Lattice algorithms and lattice-based cryptography

Natalia Kharchenko

SORBONNE UNIVERSITÉ

THÈSE DE DOCTORAT DE
SORBONNE UNIVERSITÉ

Spécialité **Informatique**

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Natalia KHARCHENKO

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Lattice algorithms and lattice-based cryptography

Thèse dirigée par Antoine JOUX

soutenue publiquement le 27 Mai 2020

aprés avis des rapporteurs :

M. Pierre-Alain FOUQUE	Prof., Université de Rennes
M. David NACCACHE	Prof., ENS

devant le jury composé de :

M. Jean-Claude BAJARD	Prof., Sorbonne Université, IMJ
M. Pierre-Alain FOUQUE	Prof., Université de Rennes
M. Louis GOUBIN	Prof., UVSQ
M. Antoine JOUX	Tenured Research Faculty, CISPA Helmholtz Center
M. David NACCACHE	Prof., ENS
Mme. Annick VALIBOUZE	Prof., Sorbonne Université, LIP6
Mme. Brigitte VALLÉE	DR, Université Caen, GREYC

Résumé

La cryptographie basée sur les réseaux est un domaine de recherche qui étudie la construction d'outils et de protocoles pour une communication sécurisée basée sur des problèmes de réseau difficiles. La cryptographie basée sur les réseaux est l'un des candidats les plus prometteurs pour la communication sécurisée post-quantique en raison de sa sécurité conjecturée contre les attaques quantiques et de son efficacité algorithmique. Une autre caractéristique intéressante de la cryptographie basée sur les réseaux est un large éventail de constructions disponibles qui incluent même des schémas de chiffrement totalement homomorphes (Fully Homomorphic Encryption ou FHE en anglais). Cette thèse étudie les algorithmes pour résoudre les problèmes de réseau difficiles et leur application à l'évaluation de la sécurité des constructions cryptographiques basées sur les réseau. Cette thèse comporte deux parties.

Dans la première partie, on introduit une nouvelle famille d'algorithmes de sieving appelé sieiving cylindrique. Le sieving heuristique est actuellement l'approche la plus rapide pour résoudre les problèmes de réseau central, le problème de vecteur le plus court (Shortest Vector Problem ou SVP en anglais) et le problème de vecteur le plus proche (Closest Vector Problem ou CVP en anglais). Dans cette thèse, on propose un nouvel algorithme de sieving pour résoudre SVP et CVP, qui fonctionne avec des vecteurs de réseau de géométrie différente par rapport aux algorithmes de sieving habituels. Le sieving cylindrique fonctionne en générant une liste de vecteurs de réseau à l'intérieur d'un hypercylindre long et étroit, puis il tamise de manière itérative les vecteurs de la liste afin d'obtenir une nouvelle liste de vecteurs à l'intérieur d'un hypercylindre beaucoup plus court mais légèrement plus large. On montre que cette approche peut être très efficace pour résoudre certaines variations de CVP et SVP. Premièrement, le sieving cylindrique améliore la complexité temporelle asymptotique pour la résolution de SVP sur des réseaux dont le volume est un nombre premier relativement petit. Par exemple, il permet de résoudre SVP pour un réseau de dimension n de volume premier d'environ 2^n dans le temps $\widetilde{O}(2^{0.229n})$. Deuxièmement, il permet d'obtenir la complexité temporelle polynomiale d'une requête pour résoudre le problème de vecteur le plus proche avec prétraitement (Closest Vector Problem ou CVPP en anglais), pour un coût en temps de $\widetilde{O}(2^{0.531n})$ et en memoire $\widetilde{O}(2^{n/2})$ lors de la phase de prétraitement pour un réseau arbitraire.

Dans la deuxième partie, on étudie la sécurité de Fast Fully Homomorphic Encryption scheme over Torus (TFHE). TFHE est l'un des schémas de chiffrement totalement homomorphe les plus rapides basé sur le problème (ring-)Learning with Errors (LWE). Dans cette thèse, on améliore l'attaque à base de réseau dual utilisée dans l'estimation de sécurité initiale du schéma. Plus précisément, on réalise l'attaque duale sur des sous-réseaux projetés, qui permet de générer des instances du problème LWE avec un bruit légèrement plus grand qui correspond à une fraction de la clé secrète. Ensuite, on recherche la fraction de la clé secrète en calculant le bruit correspondant pour chaque candidat en utilisant les échantillons LWE construits. Comme les clés de TFHE sont des vecteurs binaires, on peut effectuer l'étape de recherche très efficacement en exploitant la structure récursive de l'espace de recherche. Cette approche offre un compromis entre le coût de la réduction du réseau et la complexité de la partie recherche qui permet d'accélérer l'attaque. On implémente un script qui estime la complexité de l'attaque duale d'origine et de notre méthode hybride pour divers paramètres sous trois modèles de coûts différents pour la réduction du réseau et montre que le niveau de sécurité actuel (en mars 2020) de TFHE devrait être réévalué selon l'amélioration proposée.

Abstract

Lattice-based cryptography is an area of research that studies the construction of tools and protocols for secure communication based on hard lattice problems. Lattice-based cryptography is one of the most promising candidates for post-quantum secure communication due to its conjectured security against quantum attacks and algorithmic efficiency. Another attractive feature of lattice-based cryptography is a wide range of available constructions which includes even Fully Homomorphic Encryption (FHE) schemes. This thesis studies algorithms for solving hard lattice problems and their application to evaluating security of lattice-based cryptographic constructions. This thesis has two parts.

In the first part, we introduce a new family of lattice sieving algorithms called cylindrical sieving. Heuristic sieving is currently the fastest approach for solving central lattice problems, the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP). In this thesis, we propose a new sieving algorithm for solving SVP and CVP, that works with lattice vectors of different geometry compared to usual sieving algorithms. Cylindrical sieving works by generating a list of lattice vectors inside a long and narrow hypercylinder and then it iteratively sieves vectors from the list in order to obtain a new list of vectors inside of much shorter but slightly wider hypercylinder. We show that this approach can be very efficient for solving certain variations of CVP and SVP. First, cylindrical sieving improves asymptotical time complexity for solving SVP on lattices whose volume is a relatively small prime number. For example, it allows to solve SVP for n-dimensional lattice of prime volume about 2^n in time $\widetilde{O}(2^{0.229n})$. Second, it allows to achieve the polynomial time complexity of one query for solving the Closest Vector Problem with Preprocessing (CVPP), at the cost of spending $\widetilde{O}(2^{0.531n})$ time and $\widetilde{O}(2^{n/2})$ memory on the preprocessing for an arbitrary lattice.

In the second part, we study the security of the Fast Fully Homomorphic Encryption scheme over the Torus (TFHE). TFHE is one of the fastest FHE schemes based on the (ring-)Learning with Errors (LWE) problem. In this thesis, we improve the dual lattice attack used in the original security estimate of the scheme. More precisely, we use the dual attack on a projected sublattice, which allows to generate instances of the LWE problem with a slightly bigger noise that correspond to a fraction of the secret key. Then, we search for the fraction of the secret key by computing the corresponding noise for each candidate using the constructed LWE samples. As the TFHE keys are binary vectors, we can perform the search step very efficiently by exploiting the recursive structure of the search space. This approach offers a trade-off between the cost of lattice reduction and the complexity of the search part which allows to speed up the attack. We implement a script that estimates the complexity of the original dual attack and of our hybrid method for various parameters under three different cost models for lattice reduction and show that current (as of March 2020) security level of the TFHE scheme should be re-evaluated according to the proposed improvement.

Acknowledgements

First, I would like to thank Antoine Joux for being my thesis director. Thank you for spending a lot of time explaining me various concepts and discussing with me different ideas; thank you for always being there in spite of your busy schedule and being at another continent sometimes. I'm also very grateful to you for solving all the financial and a lot of bureaucratic problems concerning this thesis. I've learned a lot from you and it was my great luck and pleasure to work with you during these years.

I'm very grateful to Jean-Claude Bajard and Louis Goubin for participating in the last *comité de suivi* of my PhD and for very insightful discussion afterwards. Thank you for trusting in me and recommending the doctoral school to allow me an additional scientific year to work on my thesis. I would also like to thank the direction of EDITE, Habib Mehrez and Lélia Blin, for agreeing to extend my PhD.

I would like to express my deepest gratitude to Jean-Claude Bajard, Pierre-Alain Fouque, Louis Goubin, David Naccache, Annick Valibouze, and Brigitte Vallée for agreeing to be the members of my PhD defence jury. I'm very grateful to Pierre-Alain Fouque and David Naccache for agreeing to review this manuscript, for their reviews and valuable comments.

I would also like to thank all the people that I had a happy opportunity to work with during my PhD. I thank Sonia Belaïd, Mélissa Rossi, Aurélie Bauer, and Malika Izabachène for all our discussions and experiments at CryptoExperts and ANSSI. Special thanks to Thomas Espitau for all our work on this dual hybrid attack and for all the adventures around its submission. I'm very grateful to Brigitte Vallée for introducing me to sandpile models for lattice reduction and for many valuable insights about lattice reduction algorithms in general. I would also like to thank Kai-Min Chung for introducing me to the world of quantum algorithms and quantum cryptography.

It was my great fortune to be a member of the ALMASTY team during my PhD. The team has been evolving during these years, but it always remained a very warm and friendly working environment. I would like to thank Cécile Pierrot, Thomas Espitau, and Alexandre Gélin for their welcoming support during my first internship at LIP6 and the first months of my PhD; also thanks to Anand, Vincent, Jérôme, Jérémy, Charles, Lucas, and Florette for being such an amazing and friendly team. I'm very grateful to Damien Vergnaud for organizing the group seminar and to everyone who participated; and also for moving it online during the Covid confinement.

I would also like to thank Kai-Min Chung for inviting me for a two-month internship to Academia Sinica in Taiwan. I'm grateful to all the members of the Computation Theory and Algorithms group for interesting scientific discussions.

I'm very grateful to all the people who helped me with administrative tasks. I would like to thank Jean-Claude Bajard and Damien Vergnaud who helped me during my first and last year registrations to the university. Special thanks to Irphan Khan and Dany Richard who explained me how various complicated procedures are working and helped me numerous times with administrative things.

Thanks to all the PhD students and post-docs, not necessarily studying cryptography, that I've met during my PhD, for an opportunity to share our experiences and discuss

various, not necessarily mathematical, things. In particular, I'm grateful to Ilya Galanov and Luba Tupikina for all our conversations.

I'm very grateful to Sergey Dovgal for so many things that I would inevitably fail to list them all. Among those are scientific and non-scientific discussions and showing me many useful tricks in LaTeX, vim, Python, and, especially, in Valgrind. Thank you for all the typos you have found in my writing and for being my best friend for already almost 10 years.

Finally, I would like to thank my family, my parents and grandparents, for their constant and unconditional love and support that makes everything possible.

Contents

Chapter 1

Introduction

Cryptology is a study of tools and techniques for secure communication in presence of third parties. It includes two disciplines that study apparently opposite problems: *cryptography* and *cryptanalysis*. While cryptography works on design and construction of the systems for secure communication, cryptanalysis investigates methods of extracting secret information from the cryptographic constructions.

Modern cryptography encompasses a variety of protocols for secure communication in many different contexts from simple private message sending to electronic voting systems and it would be hard to describe all the existing applications in short. However, most of the protocols aim at ensuring at least one of the three basic properties: *confidentiality*, *data integrity*, and *authenticity*.

Confidentiality. The most ancient cryptographic problem is ensuring secure message transmission through the channel that can be eavesdropped. Confidentiality of messages can be achieved by applying *encryption* to the messages before sending. Encryption turns a plain text message into a sequence of letters, unreadable without an access to some secret information. Then, the message received on the other side of the channel can be easily deciphered only by the owner of the secret information. Until the XX century, confidentiality was the only problem studied by cryptography and the words "encryption" and "cryptography" are still used as synonyms sometimes.

Data integrity and authenticity. Another two problems connected to message transmission are data integrity and authenticity. Data integrity means an ability to check that the message was not modified by an adversary or by physical imperfections of the channel. Authenticity means that the sender of the message is known and there is a possibility to check that the sender is indeed the claimed person. Sealing envelopes and signing documents are the common tools for providing data integrity and authenticity used since the advent of mails. In the digital world, data integrity and authenticity can be provided by message authentication codes (MAC) and digital signatures.

1.1 Appearance of modern cryptology

Cryptology as an art of creating and breaking codes has a very long history often connected to diplomatic and military communications. Most of the known ancient civilizations developed their own methods to protect secret messages from enemy's eyes. The fascinating history of cryptography starting from ancient Egypt to the middle of the XX century can be found in [Kah96].

However, modern cryptology have appeared rather recently. The process of turning from hidden military art into modern science was dramatically accelerated in the end of

the XIX century with the development of new technologies. Appearance of new means of communication such as electrical telegraph and radio required new methods for secure communication. In 1883, Auguste Kerckhoffs have formulated 6 principles of cryptosystem's design, one of which remains relevant and very important until now: the cryptosystem must remain secure even if all the information about it, except the secret key, is public [Ker83].

A few decades later, the two world wars created a great need for both cryptographic and cryptoanalytic techniques. World War II significantly advanced the usage of automatization in cryptology. On one side, the rotor stream cipher machines were used extensively to encrypt secret messages; on the other side, electromecanical devices were used for their cryptanalysis. Probably the most impactful cryptanalytic event of World War II was the breaking of Nazi Germany's Enigma machine. Enigma ciphers were broken in 1932 by Marian Rejewski in Polish Cipher Bureau. Seven years later Polish Cipher Bureau shared their discoveries with France and Great Britain. Then, Alan Turing advanced Polish techniques and created an electromechanical machine called the bombe, which allowed the Allies to read a plethora of German's secret communication.

The cryptological experience of the two world wars together with the creation of the first digital computers in the late 40s produced the necessity of a rigorous theory encompassing existing knowledge and allowing to build new constructions. In the middle of XX century, the modern cryptology has appeared. There are two fundamental and very impactful papers, each of which can be considered as the starting point of modern cryptography. The first is written by Claude Shannon in 1949 and it considers cryptography from information-theoretical point of view [Sha49]. The second paper have appeared in 1976 [DH76]. In this paper, Diffie and Hellman connect security of a cryptosystem with the computational complexity of the underlying hard problem and introduce a revolutionary concept of *public-key cryptography*.

1.2 Information-theoretical notion of security

The one-time pad is one of the most simple encryption techniques and it is the first encryption scheme admitting provable security results. The one-time pad works as follows. Assume that there are two communicating parties, Alice and Bob. First, they share a bit string $k = (k_1, \dots, k_n) \in \{0,1\}^n$ of length n using some secure communication channel. The bit string k is the secret key for the one-time pad encryption. Then, when Alice decides to send Bob a message M, she encodes M into a bit string $m = (m_1, \dots, m_n) \in \{0,1\}^n$ of length n and encrypts each bit of the message m by adding it to the corresponding bit of the key k modulo 2:

$$e = \mathrm{Enc}(m) = (m_1 \oplus k_1, \dots, m_i \oplus k_i, \dots, m_n \oplus k_n).$$

The decryption procedure coincides with the encryption. When Bob receives the encrypted message e, he adds bit-wise the secret key k to the encryption e and recovers the original message m:

$$\mathrm{Dec}(e) = (e_1 \oplus k_1, \dots, e_i \oplus k_i, \dots, e_n \oplus k_n) = (m_1 \oplus k_1 \oplus k_1, \dots, m_n \oplus k_n \oplus k_n) = m.$$

In [Sha49], Shannon proved that if the secret key k is a random bit string, then the one-time pad is information-theoretically secure. That is, the encryption e provides no information about the plaintext m to the eavesdropper: after learning the encryption e, the probability distribution on the set of all possible messages remains the same for the eavesdropper as before seeing the encryption. This makes the one-time pad unbreakable when properly used.

However, the information-theoretical notion of security has very strong limitations which make it hard to use in practice. In order to ensure the security of the one-time pad, two requirements should be fulfilled:

- the key should be as long as the plaintext message and can never be reused;
- the key should be truly random, i.e., sampled from the uniform distribution on $\{0,1\}^n$.

As the key should be as long as the information needed to be transmitted, it might be hard to provide secure key exchange for the one-time pad. Due to the difficulties of the key management, the one-time pad is now rarely used in practice.

1.3 Public-key cryptography

A revolutionary solution to the key management problem was proposed by Diffie and Hellman in 1976. In [DH76], they proposed a concept of a cryptosystem that can establish secure communication between users without any preliminary communication through secure channels for key exchange. This is achieved by associating two keys instead of one with each user: one key is for encryption and is public, another is for decryption and is known only to its owner.

A note on secret-key cryptography. Public-key cryptography is also called asymmetric cryptography sometimes in contrast to symmetric cryptography. Before 1976, only symmetric cryptography existed. Symmetric, or secret key, cryptography studies encryption algorithms that use the same key for both encryption and decryption. For example, one-time pad is a symmetric cipher. Another two famous examples of symmetric ciphers are DES and AES. Symmetric ciphers are very efficient and widely used in practice, for example, in TLS protocol for secure network communication. Public-key and secret-key cryptography are often used together: some public key protocol (or Diffie-Hellman key exchange) is used to share the symmetric secret key and then a symmetric cipher is used for communication. As the focus of this thesis is public-key cryptography, we are not going into details about symmetric cryptography; for the introduction to symmetric ciphers see [MKVOV96, Chapters 6 and 7].

Definition. Formally, a public-key cryptosystem can be described as a set of three algorithms:

1. a probabilistic *key generation* algorithm K that takes as input the security parameter n of the scheme and returns a pair of keys (k_e, k_d), where k_e is the public key and k_d is the secret key;
2. an *encryption* algorithm Enc that takes as input a plaintext message m and the public key k_e and returns an encryption $c = \mathrm{Enc}(m, k_e)$ of the message m;
3. a *decryption* algorithm Dec that takes as input an encrypted message c and the secret key k_d and returns a plaintext $m = \mathrm{Dec}(c, k_d)$.

A public-key cryptosystem is used as follows. First, a new user Alice generates a pair of keys (k_e, k_d) by the key generation algorithm K. Alice publishes the key k_e and keeps secret the key k_d. Then, if Bob wants to send a secret message m to Alice, he encrypts it using Alice's public key k_e and send $c = \mathrm{Enc}(m, k_e)$ to Alice. Then, Alice decrypts Bob's message using her secret key k_d and gets a plaintext $m = \mathrm{Dec}(c, k_d)$.

For being correct and secure, a public-key cryptosystem must meet the following requirements:

1. *(correctness)* for any message m, decryption of $\mathrm{Enc}(m, k_e)$ under the correct key should restore the original plaintext message: [1]

$$\forall m, \quad \mathrm{Dec}(\mathrm{Enc}(m, k_e), k_d) = m;$$

2. *(semantic security [GM84], informal)* for any two messages $m_1 \neq m_2$, there is no efficient algorithm that given as input the public key k_e, the messages m_1, m_2, and the encryption c of one of the two messages chosen at random, cannot guess to which message the encryption c belongs.

The semantic security can be seen as a relaxation of Shannon's information-theoretical security definition in context of computational complexity. The information-theoretical definition of security demands that the encryption of a message leaks absolutely no information about the plaintext, which implies too strong restrictions on cryptosystems satisfying it. Diffie and Hellman in [DH76] suggested a more practical approach. Instead, they allow the encryption to contain some information about the plaintext but demand that retrieving that information from the encryption is computationally infeasible. Thus, in order to measure the security of a cryptosystem, modern cryptography uses the tools from computational complexity theory and to construct a secure cryptosystem we need algorithmic problems that are hard to solve.

RSA cryptosystem. The first concrete instantiation of a public-key encryption scheme is the RSA cryptosystem [RSA78], named by their authors Rivest, Shamir, and Adleman. Now RSA is probably the most famous and the most widely-used cryptosystem. The security of RSA relies on the integer factorization problem. Here we recall in short how the "textbook RSA" works. [2]

Key generation. In RSA, the key generation algorithm works as follows.

1. First, it generates two random prime numbers p, q of close binary length. p and q should be kept secret.
2. Then, the algorithm computes the modulus $n = p \cdot q$. All computations in RSA are in $\mathbb{Z}_n^*$ and n is a part of the public key.
3. Then, the algorithm computes Euler's phi function $\phi(n) = (p-1)(q-1)$ and chooses e such that e is coprime with $\phi(n)$.
4. Then, it computes d such that

$$d \cdot e = 1 \mod \phi(n).$$

   As e and $\phi(n)$ are coprime, d can be found using the extended Euclidean algorithm.
5. The key generation algorithm returns the pair (e, n) as public key and d as secret key.

Encryption. In order to encrypt a message, a user should encode it into a number $m \in \mathbb{Z}_n$ and raise it to the power e:

$$c = \mathrm{Enc}(m) = m^e \mod n.$$

Decryption. The decryption works similarly, but the exponent for decryption is d:

$$\mathrm{Dec}(c) = c^d = m^{e \cdot d} = m \mod n.$$

1. This requirement can be relaxed: instead of requiring the correctness equality to always be fulfilled, sometimes it is required that it holds with probability close to one.

2. Note that the "textbook RSA" is deterministic and thus it is not semantically secure. But RSA can be made semantically secure and resistant to chosen plaintext attacks by padding techniques.

The equality that ensures the correctness of decryption follows from Fermat's little theorem.

Security. The security of the RSA cryptosystem is based on the presumable hardness of the following problem.

Definition 1.1. Given a number n that is a product of two unknown prime numbers, an integer e coprime with $\phi(n)$, and $c \in \mathbb{Z}_n^*$, the RSA problem asks to find $m \in \mathbb{Z}_n^*$ such that m is a valid RSA decryption of c, i.e.,

$$m^e = c \mod n.$$

The RSA problem is not harder than the integer factorization problem. If the factorization of the modulus n is known, then $\phi(n)$ can be easily computed and the secret key d can be obtained using the extended Euclidean algorithm in the same way as in the key generation. The most efficient attacks against RSA start with factoring the modulus n, thus the practical security of RSA is based on the integer factorization problem.

Theoretically, however, RSA might be an easier problem then the integer factorization. For the moment there is no reduction showing that an algorithm solving the RSA problem can be efficiently converted into an integer factorization algorithm.

Applications besides encryption. The advent of public-key cryptography has expanded the applications of cryptographic constructions. Now, besides simple encryption of messages, there are many other cryptographic protocols used on a daily basis such as digital signatures and authentication schemes. The recent advances in public key cryptography also allow to build more powerful constructions such as identity-based encryption, attribute based encryption, and even fully homomorphic encryption.

1.4 Fully Homomorphic Encryption

In 1978, Rivest, Adleman, and Dertouzos suggested the idea of a *privacy homomorphism* [RAD+78]. They noted that existing encryption techniques are quite rigid in terms of the operations allowed on the encrypted data: the encrypted data can be stored or sent to a user, but for more complicated operations usually decryption is required. In [RAD+78] the authors posed the following question: is it possible to construct an encryption scheme that allows to perform arbitrary operations on encrypted data without decrypting it?

As an application of such an encryption scheme, Rivest, Adleman, and Dertouzos proposed private data banks. Assume that Alice has some sensitive data that is too large to be stored on her computer and that Bob is a data bank. Then, Alice can encrypt her data using a homomorphic encryption scheme and send the encrypted data to Bob. Since the encryption scheme allows to perform any computations homomorphically without decrypting, each time when Alice needs to retrieve some information from her storage, she can just send a query (i.e., a function that she needs to compute) to Bob. Then, Bob can apply the function to the encrypted data and send the encrypted answer to Alice's query without an access to the plaintext data. This is much more efficient than retrieving all the information back from the server and decrypting it each time when something is needed. Besides private data banks, homomorphic encryption can be used for solving other related problems such as cloud computing and secure multi-party computations.

For the next three decades after [RAD+78] it was an open question whether an encryption scheme that allows homomorphic computation of any function can ever be constructed. In 2009, Gentry solved the question by proposing the first fully homomorphic encryption scheme based on ideal lattices [GB09].

Definition. A homomorphic encryption scheme is a public-key encryption scheme that can be informally described as a set of three algorithms:

— – as a usual public-key cryptosystem, it has encryption and decryption procedures Enc and Dec and the corresponding keys: a public key k_e and a secret key k_d;

— – it has an efficient algorithm for homomorphic computations Eval, that takes as input a circuit C_f corresponding to a function f that user wants to evaluate, the encryptions of the arguments of the function, and the public key and returns the encryption of the value of f on the arguments.

To be correct, a homomorphic encryption scheme should satisfy the following requirement: for any allowed circuit C_f corresponding to the function f of t arguments, any valid pair of keys (e, d), and any ciphertexts $c_1 = \text{Enc}(m_1, k_e), \ldots, c_t = \text{Enc}(m_t, k_e)$, the algorithm Eval returns

$$c_f = \text{Eval}(C_f, c_1, \ldots, c_t, k_e) \quad \text{such that} \quad \text{Dec}(c_f, k_d) = f(m_1, \ldots, m_t).$$

From partially to fully homomorphic. A fully homomorphic encryption scheme should be able to evaluate any Boolean function on encrypted data. Recall that any Boolean function can be expressed as a composition of elementary functions from some functionally complete set. Common examples of functionally complete sets are $\{\oplus, \wedge, 1\}$ and $\{\text{NAND}\}$, the latter is also known as Sheffer stroke $x|y := \neg(x \wedge y)$.

Thus, in order to construct a fully homomorphic encryption scheme, it is enough to be able to evaluate unlimited number of XOR and AND operations homomorphically. There are different definitions of homomorphic encryption schemes.

Partially homomorphic schemes can evaluate only some operations homomorphically. For example, RSA in its basic version is partially homomorphic: it allows to perform homomorphic multiplication. Let $k_e = (e, n)$ be a public key and let $c_1 = \text{Enc}(m_1, k_e)$ and $c_2 = \text{Enc}(m_2, k_e)$ be ciphertexts for messages m_1 and m_2 correspondingly. Then, the product $c_1 \cdot c_2$ is a valid encryption of the message $m_1 \cdot m_2$:

$$\text{Enc}(m_1, k_e) \cdot \text{Enc}(m_2, k_e) = m_1^e \cdot m_2^e \mod n = \tag{1.1}$$

$$(m_1 \cdot m_2)^e \mod n = \text{Enc}(m_1 \cdot m_2, k_e). \tag{1.2}$$

Somewhat homomorphic schemes can perform all the operations from some functionally complete set, but only a limited number of them. For example, the homomorphic encryption scheme from [BGN05] supports an arbitrary number of additions and one multiplication.

Fully homomorphic encryption scheme is able to perform unlimited number of operations that form a functionally complete set. The first such construction is described in [GB09] and it is able to perform an unlimited number of XOR and AND.

Gentry's bootstrapping. Gentry showed that a somewhat homomorphic encryption scheme which is able to perform a limited number of operations can be turned into a fully homomorphic encryption scheme if certain conditions are satisfied. The cornerstone of this construction is the *bootstrapping* of a scheme. We informally recall how it works.

Assume that we have a somewhat homomorphic encryption scheme $\mathcal{S}$ that can homomorphicaly evaluate only circuits of a limited depth. Particularly, suppose that each ciphertext is a sum of a function of the plaintext with some noise. Initially encrypted ciphertexts have small noise, but performing homomorphic operations on ciphertexts increase the associated noise of the result. The decryption algorithm works correctly only while ciphertext's noise is smaller than some threshold. The noise level required for the correct decryption limits the depth of the circuit for homomorphic evaluation.

Gentry proposed a construction for refreshing noisy ciphertexts. Note that if we could decrypt ciphertext and re-encrypt it again, noise level would be reduced to the initial low levels, but decryption requires the secret key that should not be revealed. The idea of Gentry is to perform the decryption procedure homomorphically using the encryption of the secret key k_d. Suppose that c is a noisy ciphertext we want to refresh. Then, the bootstrapping procedure works as follows:

1. first, we compute the encrypt the ciphertext once again and get $\mathrm{Enc}(c, k_e)$;
2. then, we evaluate a decryption circuit homomorphically using the encryption of the secret key $\mathrm{Enc}(k_d, k_e)$ and get

$$c_{\text{new}} = \mathrm{Eval}\big(C_{\text{Dec}}, \mathrm{Enc}(c, k_e), \mathrm{Enc}(k_d, k_e), k_e\big).$$

The result of the homomorphic evaluation of the decryption with encrypted secret key c_{new} is a fresh encryption corresponding to the same plaintext as c. The noise associated originally with c is removed by the decryption, but evaluating the decryption circuit produces its own new noise. Assume that noise induced by the homomorphic decryption is sufficiently low to allow perform at least one homomorphic NAND gate. Then, we can perform NAND homomorphically unlimited number of times using the bootstrapping procedure. As any Boolean function can be expressed as a combination of NAND gates, the somewhat homomorphic scheme $\mathcal{S}$ can be turned into a fully homomorphic scheme $\mathcal{S}^*$ by adding gate bootstrapping. Thus, a somewhat homomorphic scheme is *bootstrappable* if it is able to evaluate its own decryption circuit and at least one NAND gate.

FHE constructions. The original Gentry's construction is based on ideal lattices. Following Gentry's work, other ideal lattice-based constructions were proposed [SV10, GH11a, SS10]. The structure of the ideal lattices seems to be very suitable for FHE as they naturally possess both additive and multiplicative homomorphic properties. However, the ideal lattice-based FHE constructions are rather mathematically involved and not very efficient in practice. The Gentry-Halevy implementation [GH11b] of Gentry's FHE scheme uses a public key of size 2.3 GB while its bootstrapping operation takes about 30 minutes.

The year after Gentry's thesis, a new FHE construction has appeared. In [VDGHV10], the authors translated Gentry's FHE scheme to integers. The FHE scheme proposed in [VDGHV10] is based on the approximate GCD problem and is conceptually much simpler than the ideal lattice-based ones. Then, more integer-based schemes were created (e.g., [CMNT11, CNT12, CLT14, CS15]). However, in terms of efficiency, the integer-based schemes do not offer improvements over ideal lattice constructions.

Starting from 2011, another lattice-related family of FHE constructions appeared. These schemes are based on the Learning With Error problem (LWE) and its ring version Ring-LWE (e.g., [BV11, BGV14, FV12, BV14, Bra12, GSW13, DM15]). Now the (Ring-) LWE branch of FHE research is the most active one and the (Ring-) LWE based constructions are the most efficient. In order to achieve efficiency, some LWE-based constructions deviate from Gentry's blueprint and avoid the costly bootstrapping operation. It can be done by creating a leveled FHE scheme, a concept, introduced in [BGV14]. A homomorphic encryption scheme is called *leveled* (LHE) if it can evaluate circuits of any depth L, where L becomes a parameter of the scheme. That is, a leveled FHE can evaluate circuits of an arbitrary depth, but the complexity of the scheme increases with L. Most of the (Ring-) LWE-based HE schemes are LHE ones, but also allow bootstrapping.

The Ring-LWE-based FHE schemes became very efficient in the recent time compared to the first FHE constructions. In 2015, Ducas and Micciancio proposed a scheme called FHEW [DM15] that can do the bootstrapping faster than in 1 second. Then, next year, the Fast Fully Homomorphic Encryption Scheme over Torus (TFHE) [CGGI16] broke the record by performing the gate bootstrapping in less than 0.1 second.

1.5 Hard problems in cryptography

Ideally, we would like to be able to prove that breaking a cryptosystem takes an infeasible amount of time under some realistic computational model. In order to produce such a proof, we need to show that breaking a cryptosystem is at least as hard as solving some hard problem, i.e., a problem that can not be solved faster than in some huge time T. Fortunately or not, for the moment there is no problem that admits a mathematical proof of such property. For now, the main source of difficult problems for cryptography are the long standing mathematical questions that were studied by the community during a long period of time and still do not admit efficient solutions. A vast amount of existing public-key constructions is based on two such problems: integer factorization and discrete logarithm. This approach implies that the security of a cryptosystem is closely connected to the fastest known algorithm for solving the underlying presumably hard problem. Therefore, choosing a hard problem to construct a cryptosystem is an important and subtle task.

Hardness on average. To turn a hard problem into a cryptographic construction, it is necessary to be able to generate many hard instances of the problem. One of the subtleties in choosing the problem is that not every worst-case hard problem is hard on average and allows to generate hard instances easily. For example, the Hamiltonian path problem is known to be NP-complete, but there is an algorithm that solves this problem in expected linear time on average [GS87] for a random $G(n, p)$ graph with a fixed positive p. Thus, just taking an NP-complete problem is not enough to guarantee security in the average case. Most of the cryptographic constructions based on the NP-complete knapsack problem were broken due to the usage of easy instances [Odl90].

Quantum computer. The hardness of a problem also depends on the model of computation. There does not yet exist a polynomial time algorithm for integer factorization and discrete logarithm on a classical computer, but in the quantum world things are different. In 1994, Peter Shor proposed an algorithm that solves integer factorization and discrete logarithm in probabilistic polynomial time on a quantum computer [Sho99]. Thus, all the existing public-key constructions based on integer factorization and discrete algorithm are insecure in the world where a large-scale quantum computer exists.

Existing quantum computers are too small to be considered as a serious threat to existing RSA-like constructions (the largest quantum computer has only 53 qubits at the moment of writing this text), but the quantum computing is a very active area of research with a lot of efforts put from both academic and industrial sides. In 2016, National Institute of Standards and Technology (NIST) of US has initiated a process of standardizing quantum-resistant public-key cryptographic algorithms. According to NIST, some engineers predict that a quantum computer large enough to break existing public key schemes may appear within about next 20 years. NIST have launched a competition for presumably quantum-secure public-key encryption and signature schemes. Competition is now in the second round, and most of the schemes that survived the first round are based on hard problems arising in lattices, codes, and multivariate polynomials.

Suchwise, now more research is needed to build efficient quantum-resistant constructions and understand their security. This thesis is focused on lattice-based cryptography and algorithms for solving hard lattice problems. Lattice-based cryptography is currently one of the most promising branches of post-quantum cryptography due to the algorithmic efficiency of the constructions (especially ring-based), connection between the worst case and average case hardness for certain lattice problems, and a wide range of cryptographic constructions lattices allow to create.

1.6 Lattices and related computational problems

A lattice is a geometrical object that can be informally described as a set of intersection points of a regular grid in an n-dimensional space. By the definition, a lattice Λ is formed by all integer linear combinations of the set of linearly independent vectors $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\} \subset \mathbb{R}^n$, called a basis of Λ:

$$\Lambda := \Big\{ \sum_{i=1}^{m} x_i \cdot \mathbf{b}_i \;\Big|\; x_1, \ldots, x_m \in \mathbb{Z} \Big\}.$$

Lattices as mathematical objects have been studied over more than two hundred years and have rich connections to number theory and convex geometry. More recently, various applications of lattices in computer science have been discovered, ranging from solving computational problems like factoring polynomial over rationals [LLL82] and combinatorial optimization to cryptographic constructions.

The two central computational lattice problems are the Shortest Vector Problem (SVP) and the Closest Vector problem (CVP). In SVP, we are given a basis $\mathbf{B}$ of the lattice Λ and the goal is to find the shortest non-zero vector of the lattice. In CVP, we are given a target vector $\mathbf{t} \in \mathbb{R}^n$ and a basis $\mathbf{B}$ of a lattice Λ and the goal is we need to find the closest approximation of the target vector by a lattice vector, i.e., to find $\mathbf{v} \in \Lambda$ such that

$$\|\mathbf{t} - \mathbf{v}\| = \min_{\mathbf{x} \in \Lambda} \|\mathbf{t} - \mathbf{x}\|.$$

Both problems are known to be computationally hard. More precisely, the Closest Vector Problem is NP-hard [DKS98] and the Shortest Vector Problem is NP-hard under randomized reduction [Ajt98].

SVP and CVP also have approximate versions. In the approximate versions, instead of looking for the shortest/the closest vector, we search for a vector that lies within some fixed distance from the origin / the target vector. That is, in γ-approximate SVP we search for a lattice vector of the norm at most $\gamma \cdot \lambda_1$, where λ_1 is the length of the shortest vector of the lattice. Approximate versions of SVP and CVP remain hard for constant and even small polynomial factors.

Lattice algorithms. While exact lattice problems and their close approximate versions are hard to solve, $2^{O(n)}$-approximate SVP can be solved in polynomial time. In 1982, Lenstra, Lenstra, and Lovász proposed first lattice reduction algorithm for high-dimensional lattices that achieves this approximation factors. The LLL algorithm works by iteratively reducing two-dimensional projected sublattices of an input lattice, which allows to bound the length of the first vector of the basis, returned by the algorithm, by $2^{O(n)}\lambda_1$. In spite of so large approximation factor, LLL allows to solve various computational problems. Moreover, the LLL algorithm was also proved to be a powerful cryptoanalytic tool (see, e.g., [Odl90] for breaking knapsack-based cryptosystems using LLL).

As for exact and close approximate lattice problems, not surprisingly, the fastest algorithms have at least exponential time complexity. For solving exact lattice problems, three main strategies can be outlined: enumeration, sieving, and discrete Gaussian sampling.

The enumeration algorithm was proposed by Kannan in [Kan83]. The algorithm works by enumerating all the lattice points inside a bounded ball and can solve both SVP and CVP. The enumeration algorithms have superexponential running time (i.e., $n^{O(n)}$) but they require only a polynomial amount of memory.

The first sieving algorithm was proposed by Ajtai, Kumar, and Sivakumar (AKS) in [AKS01]. The idea of sieving algorithms is based on a simple fact that a difference of two lattice vectors is also a lattice vector. A sieving algorithm works as follows. First,

the algorithm randomly generates a long list of lattice vectors. Then, it searches for the pairs of vectors from the list such that their difference is a little bit shorter than the vectors from the original list. Such differences form a list of slightly shorter vectors for the next iteration. Proceeding iteratively in this way, we obtain very short lattice vectors in the end. The complexity of sieving algorithms is exponential both for time and memory (i.e., $2^{O(n)}$). For a long time, sieving algorithm had been considered as impractical because of presumably high hidden constant in the exponent and because of huge memory requirements. Then, in [NV08], Nguyen and Vidick computed the exact complexity of the AKS sieve and proposed its practical heuristic variant. Since that time, various improvements for the sieving process have appeared. For the moment, the fastest heuristic algorithm for solving SVP is the sieving algorithm from [BDGL16], it has $2^{0.292n}$ time complexity and $2^{0.207n}$ memory complexity.

Another approach to solving SVP and CVP is the discrete Gaussian sampling, first proposed in [ADRSD15]. Informally, the discrete Gaussian distribution is a Gaussian distribution restricted to the lattice Λ, i.e., the probability of a lattice vector $\mathbf{v} \in \Lambda$ is proportional to the Gaussian function of $\mathbf{v}$. At a high level, the algorithms based on the discrete Gaussian sampling can be seen as sieving algorithms. They start by sampling a long list of lattice vectors from the discrete Gaussian distribution of a big width. Then, the algorithm pairs vectors from the list so that the difference of the vectors from the pair is also distributed according to the discrete Gaussian distribution, but of a smaller width. Iteratively repeating this procedure, in the end, the algorithm obtains a list of lattice vectors from the distribution of the width small enough to recover the shortest lattice vector with high probability. The algorithms, based on the discrete Gaussian sampling, have the best provable complexity for solving both exact SVP and CVP [ADSD15]. The time and memory complexity for both problems is 2^n.

In the middle between the algorithms for solving exact lattice problems and LLL there are blockwise algorithms like BKZ [CN11] or D-BKZ [MW16], that achieve subexponential approximation factors. The blockwise algorithms proceed by iteratively solving the exact SVP on projected sublattices of smaller dimension. The enumeration or heuristic sieving algorithms are usually used as an oracle for solving exact SVP in blockwise algorithms.

1.7 Lattice-based cryptography.

Lattice-based cryptography studies a usage of presumably hard lattice problems as a foundation for secure cryptographic constructions. It appeared in 1996, when Ajtai proposed a collision-resistant hash function whose security is based on a certain lattice-related problem [Ajt96].

The two central hard problems in lattice-based cryptography are the Short Integer Solution (SIS) and Learning With Errors (LWE). SIS was proposed by Ajtai in [Ajt96]. In SIS, we are given a set of m random vectors $\mathbf{a}_1, \dots, \mathbf{a}_m \in \mathbb{Z}_q^n$, and the goal is to find a short linear combination of these vectors that sums to zero, i.e., find $\mathbf{z} = (z_1, \dots, z_m)^t \in \mathbb{Z}^m$ such that

$$\sum_{i=1}^{n} z_i \cdot \mathbf{a}_i = \mathbf{0}, \ \|\mathbf{z}\| \leqslant \beta,$$

for some $\beta > 0$. In [Ajt96], Ajtai shows that, for certain choice of the parameters, solving SIS on average is at least as hard as solving poly(n)-approximate SVP in the worst-case. Then, many SIS-based cryptographic constructions appeared, for example, identification protocols [Lyu08] and digital signature schemes [GPV08].

Then, in 2005, Regev introduced another lattice-related problem that admits similar average-case hardness, called the Learning With Errors Problem [Reg05]. In LWE, we are

given m noisy linear equations, i.e.. we are given a m pairs $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ such that

$$\mathbf{a}_i^t \mathbf{s} + e_i = b_i \mod q,$$

where e_i is a small noise added to the i-th equation and $\mathbf{s} \in \mathbb{Z}_q^n$ is a secret vector. The goal is to recover the vector $\mathbf{s}$. In [Reg05], Regev proposed the first public-key encryption scheme based on LWE. Since that time, LWE have been a rich source of various cryptographic constructions, including very powerful ones like identity based encryption [GPV08] and fully homomorphic encryption (FHE) [BV11, FV12].

Both SIS and LWE have ring versions which are basically the same problems adapted to a restricted family of lattices called ideal lattices. Ring versions of SIS and LWE allow to build more compact and efficient cryptographic constructions thanks to the underlying ring structure at the cost of having more restricted hardness assumption.

SIS and LWE posses many properties that make lattice-based cryptography very attractive. First, both problems are hard on average, which means that cryptographic constructions based on SIS and LWE are as hard to break as to solve the worst-case lattice problems. Second nice property is the conjectured hardness of SIS and LWE against quantum computer. In contrast to integer factorization and discrete logarithm problems [Sho99], for the moment there is no efficient quantum algorithm for hard problems underlying lattice-based cryptography. Finally, lattice-based cryptography includes a very wide range of cryptographic constructions, from classical public key encryption and signature schemes to very sophisticated ones, like, e.g., attribute-based encryption [BGG+14] and fully homomorphic encryption [GB09, BV11, FV12].

1.8 Contribution of this thesis.

This thesis has two parts. In the first part, we introduce a new family of lattice sieving algorithms, called cylindrical sieving. In the second part, we study the security of the Fast Fully Homomorphic Encryption scheme over Torus, which is currently the FHE scheme with the fastest implementation.

1.8.1 Cylindrical sieving

The first part introduces a cylindrical sieving – a new family of heuristic sieving algorithms for solving variants of the SVP and CVP problems. In the first part of this thesis, we study the complexity of solving lattice problems using the cylindrical sieving and the connections with other lattice algorithms. We show that cylindrical sieving approach can outperform existing heuristic lattice algorithms in solving some lattice problems, namely,

- solving SVP for lattices with small prime volume,
- solving CVPP.

Sieving framework

A sieving algorithm can be divided in two steps:

- generation of an exponentially long list of lattice vectors with big but bounded norm,
- iterative sieving of the vectors from the list.

Typically, a sieving algorithm generates the initial list of lattice vectors by sampling them independently from the distribution that is concentrated on the lattice vectors with some fixed norm, for example, from the discrete Gaussian distribution of some fixed width. Then, in the sieving step, the algorithm searches for the pairs of the vectors from the list such that their differences are short, i.e., shorter then the vectors from the given list. Then, the short differences form a new list of lattice vectors for the next iteration of sieving.

It is hard to describe the exact distribution of the lattice vectors from the list during the iterative sieving process. Therefore, the behavior of sieving algorithms is usually analyzed under the following natural heuristic assumption, informally described by Assumption 1.1.

Assumption 1.1. Let L be the list of lattice vectors produced by a sieving algorithm. We assume that at any iteration of the algorithm the vectors from the list L, after rescaling, behave as if they are sampled independently from the uniform distribution on the sphere $S^{n-1} = \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x}\| = 1\}$, where n is the dimension of the input lattice.

Cylindrical sieving framework

The cylindrical sieving algorithm has the structure of a usual sieving algorithm,i.e., it also generates a long list of lattice vectors and then iteratively searches for the pairs of vectors from the list that give short differences. The main difference is the geometry of the vectors from the list, produced by the cylindrical sieving. Typically, in sieving algorithms, we assume that the vectors, produced at each iteration, lie approximately on a sphere of some fixed radius, that is, no direction is privileged. The cylindrical sieving algorithm generates and sieves vectors in such a way that the vectors from the list lie inside the hypercylinder with a huge height but with a small radius. That is, it lets the projection of the vectors on one selected direction be long at the cost of having very short projection on its orthogonal complement.

Initial generation of lattice vectors. There is a simple way to generate lattice vectors inside a long and narrow hypercylinder for integer lattices whose volume is prime. If a lattice Λ has a prime volume $\mathrm{vol}(\Lambda) = p$, then it can be represented by the basis $\mathbf{B}$ of the following shape:

$$\mathbf{H} = \begin{pmatrix} p & a_1 & \dots & a_{n-1} \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & 1 \end{pmatrix}, \tag{1.3}$$

where $a_1, \dots, a_p \in \mathbb{Z}_p$.

Using the shape of this matrix, we can generate a lattice vector $\mathbf{v} \in \Lambda$ such that the first coordinate of $\mathbf{v}$ is big, but bounded by the volume of the lattice p while the last $(n-1)$ coordinates are short. This can be done as follows:

1. choose $\mathbf{u} = (u_1, \dots, u_{n-1}) \in \mathbb{Z}^{n-1}$ such that $\|\mathbf{u}\| \leqslant R$;
2. compute $v_1 = \sum_{i=1}^{n-1} a_i \cdot u_i \mod p$;
3. return $\mathbf{v} := (v_1, u_1, \dots, u_{n-1})^t$.

This approach can be generalized for arbitrary lattices using unbalanced reduction from [GINX16]. Unbalanced reduction is a polynomial-time algorithm that transforms an arbitrary lattice basis $\mathbf{B}$ into a new basis $\mathbf{C}$ such that the Gram-Schmidt orthogonalization of $\mathbf{C}$ satisfies:

- the last $(n-1)$ Gram-Schmidt vectors are short, i.e., $\|\mathbf{c}_i^*\| \leqslant \sigma$;
- $\|\mathbf{c}_1\|$ is long, but bounded.

The unbalanced reduction reveals a basis of $(n-1)$-dimensional projected sublattice $\Lambda_{\mathbf{C}}$ of $\mathscr{L}(\mathbf{B})$ that allows to generate lattice vectors whose projection on the subspace orthogonal to $\mathbf{c}_1$ is short. This projected sublattice can be seen as an analogue of $\mathbb{Z}^{n-1}$ in case of lattices with prime volume. A lattice vector that lies inside a bounded hypercylinder for an arbitrary lattice can be generated as follows:

1. choose $\mathbf{v}$ in $\mathscr{L}(\mathbf{B})$ such that the norm of the projection of $\mathbf{v}$ on the $(n-1)$-dimensional subspace orthogonal to $\mathbf{c}_1$ is bounded by $R > \sigma$, using basis $\mathbf{C}$;

2. reduce $\mathbf{v}$ with $\mathbf{c}_1$.

Thus, we obtain a vector whose projection on the direction of $\mathbf{c}_1$ is smaller than $\frac{\|\mathbf{c}_1\|}{2}$ and the projection on its orthogonal complement is bounded by R.

The approach for arbitrary lattices is very similar to the approach for prime volume lattices, but for prime volume lattices the generation of vectors in a hypercylinder is somewhat easier, because we start with generating a short vector in $\mathbb{Z}^{n-1}$ instead of an arbitrary $(n-1)$-dimensional sublattice. Also, the parameters of the hypercylinder are different in the two cases. Thus, for all the algorithms described further in this part of the thesis, we separately consider the prime volume lattice case.

Sieving step. At a high level, one iteration of cylindrical sieving works similarly to one iteration of usual sieving process. That is, input of one iteration of cylindrical sieving is again a list of lattice vectors $L \subset \Lambda$ and the goal is to find the pairs of vectors from the list $(\mathbf{x}, \mathbf{y}) \in L \times L$ such that the difference $\mathbf{x} - \mathbf{y}$ is shorter than some threshold. The difference is the geometry of the vectors from the list.

Typically in sieving, all the vectors from the list L lie inside a hypersphere of a radius R, and the goal of one iteration is to produce a list of lattice vectors that lie inside a hypersphere of a smaller radius $R' < R$. In cylindrical sieving, we have a list of vectors that lie inside a hypercylinder of height h and radius R such that h is much bigger than R. For most of the vectors from the list L, their projection on the direction of cylinder's axis contributes to their norm much more than the projection on its orthogonal complement. Hence, in order to find pairs that give shorter differences, we consider the projection on the cylinder's axis and on its orthogonal complement separately. We look for pairs of vectors such that the difference of their projection on the cylinder's axis is short. Since the projection on the orthogonal complement is small, we can let it slightly grow. In other words, the goal of one iteration of cylindrical sieving is to produce a new list of lattice vectors that lie inside a much shorter and slightly wider hypercylinder.

Assume that we choose the coordinate system in such a way that the axis of the cylinder is parallel to the unit vector $\mathbf{e}_1 = (1, 0, \ldots, 0)^t \in \mathbb{R}^n$ and initially all the vectors from the list L lie inside a hypercylinder of height h and radius R. Then, one iteration of cylindrical sieving looks for the pairs of the vectors $(\mathbf{v}, \mathbf{w}) \in L \times L$ such that

- the first coordinate of the difference is much smaller than h, i.e. $|v_1 - w_1| \leqslant \frac{h}{N}$ for some big N;
- the vector formed by the last $(n-1)$ coordinates of the difference might be bigger than R by a constant factor: let $\mathbf{u}_v := (v_2, \ldots, v_n)^t$ and $\mathbf{u}_w := (w_2, \ldots, w_n)^t$; then, $\|\mathbf{u}_v - \mathbf{u}_w\| \leqslant \gamma \cdot R$, where $\gamma \in [1; 2]$.

The growth factor γ is never bigger than 2 because initially the projection on the orthogonal complement to $\mathbf{e}_1$ is bounded by R for all vectors from L.

Splitting vectors into the sum of two orthogonal components and letting one of the two components grow makes the search for suitable pairs easier. For example, if we let the growth factor γ be equal to 2, we can completely ignore the second component and just search for the vectors whose first coordinates are close. It can be done by sorting the vectors from L by the values of their first coordinates and then choosing the pairs that are close in the sorted list.

If we choose the growth factor γ smaller than 2, we need to take some care of the second component. But having two components allows us to reduce the search space when we look for a suitable pair for one fixed vector from the list. Instead of looking through all the vectors in the list, we can separate the search process in three parts:

- we sort the list L by the value of the first coordinate of the vectors;
- we divide the sorted list L into N lists $L_1, \ldots, L_N$ such that for any $\mathbf{v}, \mathbf{w} \in L_i$, $|v_1 - w_1| \leqslant \frac{h}{N}$ for all i;

– for each L_i, we perform usual spherical sieving, ignoring the first coordinate of the vectors, i.e., we search for pairs of vectors $\mathbf{x}, \mathbf{y} \in L_i$ such that $\|\mathbf{u}_x - \mathbf{u}_y\| \leqslant \gamma R$, where $\mathbf{u}_x = (x_2, \ldots, x_n)^t$, $\mathbf{u}_y = (y_2, \ldots, y_n)^t$.

One iteration of cylindrical sieving has two parameters: the decrease rate of the first coordinate N and the growth factor of the last two coordinates γ. The parameter γ defines how much time is spent on processing of one sublist L_i, the parameter N controls the number of sublists.

To sum up, cylindrical sieving works as follows. First, it samples a long list of lattice vectors inside a bounded hypercylinder and then iteratively applies sieving step describing above. The complexity and the length of the vectors, returned by the procedure depend on the following parameters:

– parameters of the initial hypercylinder: length h and radius R;
– size of the list of the lattice vectors L;
– parameters of one iteration of the cylindrical sieving: the decrease rate N and the growth factor γ.

Varying these parameters allows to adjust the algorithm depending on a target problem. In this thesis, we consider the following applications of cylindrical sieving: solving SVP for prime volume and arbitrary lattices, solving SVP for lattices with small prime volume (i.e., $\mathrm{vol}(\Lambda) \approx 2^{O(n)}$), and solving the Closest Vector Problem with Preprocessing for prime volume and arbitrary lattices.

Gaussian heuristic. In this thesis, in order to get an estimate on the length of shortest lattice vector and on the distance from some fixed point in the space to a lattice, we use the Gaussian heuristic. Informally, the Gaussian heuristic states that the number of lattice points inside a "nice" set $S \subset \mathbb{R}^n$ is close to the ratio of the volume of the set S and the volume of the lattice,i.e. $\frac{\mathrm{vol}(S)}{\mathrm{vol}(\Lambda)}$.

The Gaussian heuristic implies that any ball of radius $O(\sqrt{n}) \cdot \mathrm{vol}(\Lambda)^{1/n}$ contains one lattice point. Hence, we assume that the shortest vector of a lattice Λ has length close to $O(\sqrt{n}) \cdot \mathrm{vol}(\Lambda)^{1/n}$ and, more generally, the distance from the lattice Λ to a point $\mathbf{t} \in \mathbb{R}^n$ is also close to $O(\sqrt{n}) \cdot \mathrm{vol}(\Lambda)^{1/n}$.

Further, when we write "algorithm for solving SVP/CVP", if there is no more precise description, it means an algorithm that searches for lattice vectors of length $O(\sqrt{n}) \cdot \mathrm{vol}(\Lambda)^{1/n}$ / at the distance $O(\sqrt{n}) \cdot \mathrm{vol}(\Lambda)^{1/n}$ from the given target.

Solving SVP using cylindrical sieving

In Chapter 4, we consider solving the Shortest Vector Problem using the cylindrical sieving approach. Here we recall in short the results from Chapter 4.

Heuristic assumption for solving SVP. The complexity of heuristic sieving algorithms is usually analyzed under heuristic assumptions close to Assumption 1.1, i.e., it is assumed that the vectors from the list, produced by a sieving algorithm, behave as sampled from the uniform distribution on a sphere.

As the geometry of the vectors produced by the cylindrical sieving is different, we use another heuristic assumption. At each iteration, the vectors from the list are contained inside a hypercylinder of bounded height and radius. To analyse the complexity of the algorithm, we separately consider the projections of the vectors from the list on the direction of cylinder's axis and the projections on its orthogonal complement.

– We do not need any heuristic assumption of the distribution of the projections on the cylinder's axis. The fact that it is bounded at each iteration is enough to guarantee that after k iterations these projections are small using pigeonhole principle.

– The projections on the orthogonal complement of the cylinder's axis are $(n-1)$-dimensional vectors. After cutting the hypercylinder into N chunks by the value of the projection on the axis, the algorithm performs a usual spherical sieving on the $(n-1)$-dimensional projections of the vectors from one chunk. To analyze the complexity of the sieving inside one chuck, we use usual spherical sieving assumption, similar to Assumption 1.1.

Informally, the heuristic assumption that we use for analyzing the complexity of solving SVP with cylindrical sieving can be summarized as follows.

Assumption 1.2. We assume that at any iteration of the cylindrical sieving process, the projections of the vectors from the list L on the $(n-1)$-dimensional subspace orthogonal to the axis of the hypercylinder, after rescaling, behave as if they are samples independently from the uniform distribution on the sphere $S^{n-2} = \{\mathbf{x} \in \mathbb{R}^{n-1} \mid \|\mathbf{x}\| = 1\}$, where n is the dimension of the input lattice.

Difference between prime volume lattice case and general case. In Section 4.1, we describe the initial generation of lattice vectors inside a long and narrow hypercylinder. The algorithm for initial generation is different in case of prime volume lattices compared to the general case. The complexity of the initial generation of lattice vectors and the parameters of the resulting hypercylinders in the two cases are compared in Table 1.1.

Table 1.1 – Λ denotes a lattice given as input to an algorithm. The parameter β is bigger than $\sqrt{e}$. [3]

	prime volume	general case
cylinder's params.	$h = \mathrm{vol}(\Lambda)$ $R = O(\sqrt{n})$	$h = 2^{O(n^2)}\,\mathrm{vol}(\Lambda)^{1/n}$ $R = 2^{-O(n)}\,\mathrm{vol}(\Lambda)^{1/n}$
time complexity	$\widetilde{O}(N)$ to generate N vectors	$\widetilde{O}(e^{\frac{n\beta^2}{2e}})$ to generate $N = \widetilde{O}(\beta^n)$ vectors

Besides the initial generation of lattice vectors, the cylindrical sieving algorithm works identically in both prime volume lattice and general cases. But, since the parameters of the initial hypercylinder and the time complexity of the initial generation are different in the two cases, the optimal choice of algorithm's parameters (e.g., the number of iterations) are different in the two cases.

Choice of the growth rate γ. In some sense, the growth rate γ is the main parameter of the cylindrical sieving. First, when we fix the height and radius of the initial hypercylinder and the desired length of the output vectors, all the other parameters and the overall complexity of the algorithm are defined by the growth rate γ. Thus, the cylindrical sieving can be seen as a family of algorithms with different ratios between the time and memory complexities, parameterized by γ.

Second, the choice of the growth rate γ defines the complexity of the most time-consuming part of the algorithm, i.e., of one iteration of the spherical sieving of $(n-1)$-dimensional vectors inside one chunk of the hypercylinder. The number of vectors needed to perform the spherical sieving is defined by the number of random spherical caps of the radius γ. The larger the parameter γ is, the fewer caps are needed to cover the sphere and the cost of the spherical sieving part is smaller.

3. For our choice of parameters, β is always bigger than $\sqrt{e}$. However, when $\beta < \sqrt{e}$, the time complexity in the general case is the same as in the prime volume case, i.e., the time needed to generate N lattice vectors is $\widetilde{O}(N)$ (see Lemma 4.2).

The complexity of solving SVP using cylindrical sieving as a function of γ is considered in Section 4.3. We are mostly interesting in the following three choices of the parameter γ:

- choose γ to optimize the time complexity of the whole algorithm;
- choose γ to optimize the time of the cylindrical sieving part (i.e., without initial generation of lattice vectors);
- choose γ to optimize the time complexity of the spherical sieving part.

The complexity of cylindrical sieving for these choices of the parameter γ is summarized in Tables 1.2 and 1.3.

Table 1.2 describes prime volume lattice case. In the prime volume lattice case, the cylindrical sieving part takes much more time than the initial generation of lattice vectors. Therefore, optimizing the time complexity of the whole algorithm is equivalent to optimizing the complexity of the cylindrical sieving.

Table 1.2 – Complexity of solving SVP using cylindrical sieving for n-dimensional lattices whose volume is a prime number.

optimize	γ	time	memory
time complexity	$\frac{3}{\sqrt{2}}$	$2^{0.3774n}$	$2^{0.292n}$
spherical sieving time	$\sqrt{2}$	$2^{n/2}$	$2^{n/2}$

Table 1.3 describes the complexity of cylindrical sieving in the general case.

Table 1.3 – Complexity of solving SVP using cylindrical sieving for an arbitrary n-dimensional lattice.

optimize	γ	time	memory
cylindrical sieving time	$\frac{3}{\sqrt{2}}$	$2^{0.396n}$	$2^{0.292n}$
time complexity	1.10378	$2^{0.3816n}$	$2^{0.262n}$
spherical sieving time	$\sqrt{2}$	$2^{0.531n}$	$2^{n/2}$

Though the complexities presented in Tables 1.2 and 1.3 imply that cylindrical sieving is slower than the current fastest heuristic algorithm for solving SVP, they show that there are two remarkable things about cylindrical sieving which make it interesting to study.

- First, the optimal complexity of the algorithm in the prime volume lattice case (and the optimal complexity of the algorithm without initial generation step in the general case) coincides with the complexity of the overlattice sieving algorithm by Becker, Gama, and Joux [BGJ14].
- Second, in both prime volume and general cases, for $\gamma = \sqrt{2}$, cylindrical sieving achieves polynomial time complexity of the spherical sieving part. As we will see in Section 1.8.1, this property might be useful for solving other lattice problems.

Solving SVP using cylindrical sieving for lattices with small prime volume

In Chapter 5, we consider solving SVP for lattices whose volume is a relatively small prime number. Before, we implicitly assumed that the volume of a lattice is equal to $2^{f(n)}$, where $f(n) = \mathrm{poly}(n)$ and $\lim\limits_{n\to\infty} \frac{f(n)}{n} = \infty$. In Chapter 5, we consider prime volume lattices with the volume equal to 2^{cn} for some constant $c > 0$.

In case of a prime volume lattice, small values of the volume are special for the cylindrical sieving. For prime volume lattices, the height of the hypercylinder that contains

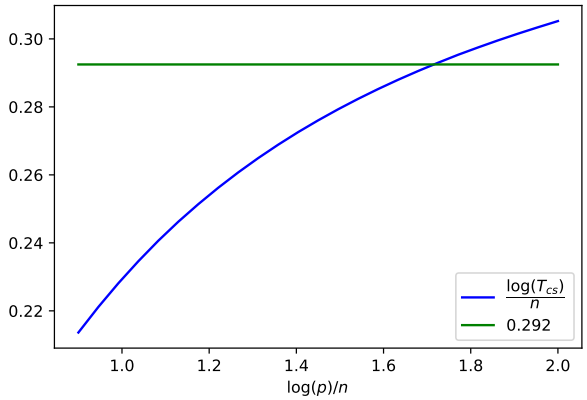


Figure 1.1 – Solving SVP using cylindrical sieving for n-dimensional prime volume lattices of the volume 2^{cn}. p denotes the volume of the lattice (i.e., $c = \frac{\log(p)}{n}$), the blue line represents the complexity of the cylindrical sieving, the green line represents the complexity of the current fastest heuristic algorithm for solving SVP [BDGL16].

initially sampled points is equal to the volume of the lattice. Thus, when the volume of an input lattice Λ is equal to 2^{cn}, the hypercylinder is initially quite short and, after a few iterations of sieving, the height of the hypercylinder is completely reduced. The number of iterations is a constant that depends on $c = \frac{\log(\mathrm{vol}(\Lambda))}{n}$ and on the ratio $\frac{R_0}{R_t}$, where R_0 is the initial radius of the hypercylinder and R_t is the desired length of the vectors returned by the algorithm.

Up to the best of our knowledge, the complexity of solving exact SVP for small volume lattices was not considered specifically before. Thus, we compare complexity of the cylindrical sieving to the current fastest sieving algorithm from [BDGL16] that works on arbitrary lattices.

Figure 1.1 represents the complexity of solving SVP using cylindrical sieving as a function of c. Figure 1.1 shows that cylindrical sieving is asymptotically faster than the algorithm from [BDGL16] for lattices of the volume smaller than $2^{1.71n}$.

However, for solving approximate SVP, there is a technique that allows to speed-up it for small volume lattices, described by Cheon and Lee in [CL15]. In Section 3.2, we adapt Cheon's technique, instantiated with the algorithm from [BDGL16], to the prime volume lattice case. Then, in Chapter 5, we compare complexities of the cylindrical sieving and of Cheon's technique (as it is described in Section 3.2) for solving approximate SVP. We find out that if the approximation factor is not too small, Cheon's technique is faster than cylindrical sieving for solving approximate SVP (see Figure 5.2).

Solving CVPP using cylindrical sieving

In Chapter 6, we adapt cylindrical sieving for solve the Closest Vector Problem. We find out that the cylindrical sieving can be very efficient for solving a variation of CVP, namely, for solving the Closest Vector Problem with Preprocessing.

Approximating target vectors using cylindrical sieving. In CVP setting, we are given a target vector $\mathbf{t} \in \mathbb{R}^n$ and a lattice $\Lambda \subset \mathbb{R}^n$ and the goal is to find a lattice vector $\mathbf{v}$ that is close to the target $\mathbf{t}$.

Consider how cylindrical sieving can be modified in order to solve this problem. First, in Section 6.3, we consider a special case of the problem, i.e., we assume that we are given

a target $\mathbf{t}$ that belongs to the initial hypercylinder of the cylindrical sieving algorithm. Then, as when solving SVP, we generate a long list $L \subset \Lambda$ of lattice vectors that lie inside the hypercylinder. If the list L is big enough, with high probability, among the points from the list L there is a point that is close to $\mathbf{t}$. More precisely, we want to find a point $\mathbf{v}_1$ such that the difference $\mathbf{t} - \mathbf{v}_1$ belongs to the next hypercylinder of the sieving process. The vector $\mathbf{t}_2 := \mathbf{t} - \mathbf{v}_1$ becomes a new target for the next iteration. Then, we sieve the lattice vectors from the list L as when solving SVP to get an updated list L that belongs to the next hypercylinder. To sum up, the i-th iteration works as follows:

1. reduce the current target point $\mathbf{t}_i$ with the list L that belongs to the i-th hypercylinder to get the target $\mathbf{t}_{i+1}$ that belongs to the next hypercylinder;
2. reduce the lattice points from L with each other to get a new list of lattice points that belong to the next hypercylinder.

During this process, we store all the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \Lambda$ that were used to reduce the target point. As the last difference $\mathbf{t}_k - \mathbf{v}_k$ belongs to the last small hypercylinder, this difference is short. Then, the sum $\sum_{i=1}^{k} \mathbf{v}_i \in \Lambda$ is a close approximation of the target $\mathbf{t}$.

The procedure described above can be extended to an arbitrary target vector. In Section 6.4.1, we show that any $\mathbf{t} \in \mathbb{R}^n$ can be decomposed into the sum of two vectors $\mathbf{t}'$ and $\mathbf{v}$ such that

- $\mathbf{v}$ belongs to the lattice Λ;
- $\mathbf{t}'$ is very close to the first hypercylinder of the cylindrical sieving.

Thus, solving CVP for $\mathbf{t}'$ implies solving CVP for the original target $\mathbf{t}$. The desired decomposition can be found by reducing the target $\mathbf{t}$ with the basis of Λ that is used for the initial generation of lattice vectors.

Heuristic assumption. In order to analyze the complexity of solving CVP using the cylindrical sieving, we need to estimate the probability to find a suitable pair for the target $\mathbf{t}$ among the vectors from the list L. To estimate this probability, we need some assumption on the behavior of the vectors from the list L. Before, when solving SVP, we assumed that the projections of the vectors from the list L on the $(n-1)$-dimensional subspace orthogonal to the hypercylinder's axis behave like uniformly distributed on a sphere, but we didn't need any assumption on the projections of the vectors on the cylinder's axis. The fact that this projection is bounded was enough to guarantee that we get short vectors in the end. But, when solving CVP, we need to consider both projections, because during the iterations of the sieving process the target vector may appear anywhere inside the hypercylinder. Therefore, the heuristic assumption for solving CVP using the cylindrical sieving has two parts for each of the two orthogonal components:

- for the projections of the vectors on the $(n-1)$-dimensional subspace orthogonal to the cylinder's axis, the heuristic assumption remains the same as Assumption 1.2;
- for the projections of the vectors on the cylinder's axis, we assume that they behave as sampled from a quasi-uniform distribution; informally, a quasi-uniform distribution is the distribution on a bounded interval such that the probability of any not too small subinterval is proportional to its length (see Definition 3.4 for a formal definition).

Solving CVPP. Consider separately reducing a target point $\mathbf{t}_i$ with the list of lattice vectors $L \subset \Lambda$ at the i-th iteration. Recall that the list L is sorted by the value of the projection of the vectors on the cylinder's axis. Reducing the target point $\mathbf{t}$ with the list L can be divided in two parts:

1. find the first vector $\mathbf{v} \in L$ such that the projection of the difference $\mathbf{t}_i - \mathbf{v}$ on the cylinder's axis is small enough;

2. subsequently consider all the vectors in L starting from $\mathbf{v}$ until the vector $\mathbf{v}'$ such that the difference $\mathbf{t}_i - \mathbf{v}'$ belongs to the next hypercylinder is found.

As the list L is sorted, the first step can be performed in time $O(\log(|L|)) = O(n)$. The second step is essentially equivalent to search for a close vector on an $(n-1)$-dimensional sphere. In Section 1.8.1, we have seen that for certain parameters, i.e., when $\gamma = \sqrt{2}$, the spherical sieving part of the cylindrical sieving can be performed in $\text{poly}(n)$ time. Thus, reducing a target point with the list L can be performed in a polynomial time when $\gamma = \sqrt{2}$.

The fact that reducing the target with the list L can be performed in a polynomial time allows to construct a very efficient algorithm based on the cylindrical sieving for solving the Closest Vector Problem with Preprocessing. The preprocessing part is basically identical to the cylindrical sieving algorithm for solving SVP. The main difference is that now, after each iteration of the cylindrical sieving, we store the state of the list L. The stored lists of lattice vectors form the description of the lattice Λ. Then, when the target appears, the decoding procedure consists in iterative reducing of the target with each of the lists of lattice vectors, produced by the preprocessing. As the number of iterations is polynomial, the overall complexity of the decoding is polynomial too.

The cylindrical sieving algorithm for solving CVPP is described in details in Section 6.4. In Table 1.4, we summarize the complexity of the algorithm. [4] As for SVP, it is different in the general case and in the case of prime volume lattices.

Table 1.4 – Complexity of solving CVPP for n-dimensional integer lattices using the cylindrical sieving.

lattice	preprocessing time	description size	decoding time
prime volume	$2^{n/2}$	$2^{n/2}$	$\text{poly}(n)$
arbitrary	$2^{0.531n}$	$2^{n/2}$	$\text{poly}(n)$

Thus, cylindrical sieving allows to solve CVPP queries in polynomial time at the cost of spending an exponential time at the preprocessing stage. This asymptotically improves the time complexity of the query cost compared to the current fastest heuristic Voronoi cell algorithm for solving CVPP from [DLdW19]. The family of Voronoi cell algorithms from [DLdW19] allows to solve a CVPP query in time $2^{\varepsilon n + o(n)}$ at the cost of spending $\left(\frac{1}{\varepsilon}\right)^{O(n)}$ for arbitrary $\varepsilon > 0$.

Implementation. Since the analysis of the cylindrical sieving algorithm for solving CVPP is based on a new heuristic assumption, it is important to verify it experimentally. To do so, we implement the cylindrical sieving algorithm in C++ using GMP library and test our implementation on random prime volume lattices of the following parameters:

- dimension $n = 30$, volume $\text{vol}(\Lambda) \approx 2^{60}$ (see Section 6.5.2);
- dimension $n = 20$, volume $\text{vol}(\Lambda) \approx 2^{100}$ (see Section 6.5.3);
- dimension $n = 40$, volume $\text{vol}(\Lambda) \approx 2^{80}$. (see Section 6.5.4);

We describe our implementation and performed experiments in details in Section 6.5. The main goal of the experiments was to check whether the distribution of the projections of the vectors from the list L on the cylinder's axis is quasi-uniform. For each set of the parameters, at each iteration of the sieving process we measured several metrics, like the

4. Note that the cylindrical sieving, by the definition, is not a polynomial-time algorithm for solving CVPP, because it produces a lattice description of an exponential size. See Section 2.4 for a formal definition of CVPP. Also, note that since the decoding procedure performs a search in a list of an exponential size, $\text{poly}(n)$ time complexity means the complexity in the RAM computational model.

percentage of the cylinder's surface, covered by the points from the list, number of collisions/unique values in the list L, and others. Our experiments show that the distribution of the vectors inside the hypercylinder is indeed quite regular.

Also, for each set of the parameters, we tried to solve 1000 random CVPP queries using the preprocessing information. In most of the cases (more than 90%), a close lattice vector was successfully recovered by the algorithm.

1.8.2 Estimating TFHE's security under hybrid dual lattice attack

In the second part, we consider the security of the Fast Fully Homomorphic Encryption scheme over Torus (TFHE). This part is based on joint work with Thomas Espitau and Antoine Joux. We propose an improvement of the dual lattice attack that was originally used in [CGGI17] to estimate the security of the scheme. We re-estimate the security of TFHE under our improved version of the attack using several models of lattice reduction algorithms.

Security of the TFHE scheme.

In Chapter 7, we recall basic information about TFHE and its security. Fast Fully Homomorphic Encryption scheme over the Torus [CGGI16, CGGI17, CGGI20], is currently the FHE scheme with the fastest implementation that we are aware of. Abstractly, all the operations in the TFHE scheme are defined on the real torus $\mathbb{T}$. The security of the TFHE scheme relies on the hardness of a variation of the LWE problem, named Torus-LWE. As a "learning a character" problem, it encompasses both the celebrated LWE and ring-LWE problems.

The security of a cryptosystem is defined by the complexity of the most efficient known attack against it. In particular, to estimate the security of an LWE-based construction, it is important to know which attack is the best for the parameters used in the construction. It can be a difficult issue; indeed, the survey of existing attacks against LWE given in [APS15] shows that no known attack would be the best for all sets of LWE parameters.

The gate bootstrapping in TFHE consists of two parts: bootstrapping and key switching. Each part uses its own secret key, thus, the overall security of the scheme is the security of the weakest of the two keys.

In the case of TFHE, the security estimation method together with the parameters of the scheme has changed over time. Originally, in [CGGI17], the the authors adapted the dual distinguishing lattice attack from [Alb17] to evaluate the security of their scheme. Then, recently, in [CGGI20, Remark 9], the authors propose an updated set of the parameters for their scheme and estimate the security of the new parameters using the LWE estimator from [ACD+18].

For a while, the default parameters of TFHE's public implementation were a composition of the two sets: the key switching key was as in [CGGI17], while the bootstrapping key was updated according to Remark 9 from [CGGI20]. Then, on February 21, 2020, the parameters of the implementation were updated for both keys. The security of the new implementation's parameters is also estimated using the LWE estimator [ACD+18]. We summarize all the choices of the parameters and recall their security claimed by TFHE's authors in Table 1.5.

Hybridizing dual lattice attack

In Chapter 8, we generalize the dual lattice attack which is currently used to evaluate the security of the TFHE scheme. First, in Section 8.1, we present a complete and detailed

analysis of the standard dual lattice attack [5] on LWE from [CGGI20]. Then, in Section 8.2, we show that applying the dual attack to a projected sublattice and combining it with the search for a fraction of the key can yield a more efficient attack.

More precisely, our attack starts by applying lattice reduction to a projected sublattice in the same way it is applied to the whole lattice in the dual attack with lazy modulus switching. This way we generate LWE instances with bigger noise but in smaller dimension, corresponding to a fraction of the secret key. Then, the freshly obtained instances are used to recover the remaining fraction of the secret key. For each candidate for this missing fraction, we compute the noise vector corresponding to the LWE instances obtained at the previous step. This allows us to perform a majority voting procedure to detect the most likely candidates. As the TFHE scheme uses binary vectors for keys, this step boils down to computing a product of a matrix composed of the LWE samples with the matrix composed of all binary strings of length equal to the dimension of the part of the secret key that we are searching for. We show that this computation can be performed efficiently thanks to the recursive structure of the corresponding search space. The number of bits of the secret key that the attack aims to guess gives an additional parameter for tuning the complexity of the attack. Hence, this hybrid approach offers a trade-off between the quality of lattice reduction in the dual attack part and the time subsequently spent in the exhaustive search part. Together with the efficient computation of the noise for each candidate, the optimal parameters for this trade-off gives an asymptotic improvement of the whole complexity.

We evaluate the complexity of the standard dual attack and of our attack for a wide range of LWE parameters. We estimate the complexities of both attacks under three different models of the lattice reduction. For all the models, our estimates show that our attack outperforms the standard dual attack.

In particular, we estimate the complexity of our attack for the parameters used in the TFHE scheme. For completeness, we re-evaluated the security of all the sets of TFHE's parameters. We describe all of these choices in Table 1.5 and showcase the corresponding estimated security within our attack framework.

Table 1.5 – Security of the parameters of TFHE. n denotes the dimension, α is the parameter of the modular Gaussian distribution. The column "λ" represents the bit-security claimed by TFHE's authors, the column "new λ" represents the security of the scheme against hybrid dual attack in the sieving model of lattice reduction.

key	n	α	λ	new λ
switching [CGGI17]	500	$2.43 \cdot 10^{-5}$	**159**	**94**
bootstrapping [CGGI17]	1024	$3.73 \cdot 10^{-9}$	198	122
switching [CGGI20]	612	2^{-15}	**128**	**118**
bootstrapping [CGGI20]	1024	2^{-26}	129	120
switching (impl. [G+16])	630	2^{-15}	**128**	**121**
bootstrapping (impl. [G+16])	1024	2^{-25}	130	125

5. We shall remark that this attack is slightly more subtle than the classical dual lattice attack, as it encompasses a continuous relaxation of the *lazy modulus switching* technique of [Alb17].

Chapter 2

Background

In this chapter, we provide notation and background on lattices and computational lattice problems common for both parts of the thesis.

2.1 Notation

Base-two logarithm is denoted by log, natural logarithm is denoted by ln. Vectors are denoted by bold lower-case letters (e.g. $\mathbf{x}$), matrices are denoted using bold upper-case letters (e.g. $\mathbf{A}$). For a vector $\mathbf{x}$, we denote the transpose of $\mathbf{x}$ as $\mathbf{x}^t$, i.e. $\mathbf{x}^t$ is a row-vector. For any two vectors $\mathbf{x}$ and $\mathbf{y}$, $\mathbf{x}^t\mathbf{y}$ denotes the scalar product of $\mathbf{x}$ and $\mathbf{y}$. For a matrix $\mathbf{A}$, its transpose is also denoted by $\mathbf{A}^t$.

We denote the n-dimensional real space as $\mathbb{R}^n$. We write $\mathbf{0}$ for the vector that consists of zeros. For a vector $\mathbf{x} \in \mathbb{R}^n$, we write $\|\mathbf{x}\|$ to represent the Euclidean norm of $\mathbf{x}$. When a coordinate system is defined, we denote the coordinates of an n-dimensional vector $\mathbf{x}$ as $x_1, \ldots, x_n$. For any set $S \in \mathbb{R}^n$, we denote a linear subspace of $\mathbb{R}^n$, spanned by the vectors from S, as $\mathrm{span}(S)$. For any subspace V of $\mathbb{R}^n$, we denote its orthogonal complement as $V^\perp$.

For a vector $\mathbf{c} \in \mathbb{R}^n$ and a positive real R, we denote as $B^n(\mathbf{c}, R) := \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x} - \mathbf{c}\| \leqslant R\}$ the closed Euclidean ball in $\mathbb{R}^n$ of radius R centered at $\mathbf{c}$, and we denote as $S^{n-1}(\mathbf{c}, R) := \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x} - \mathbf{c}\| = R\}$ the n-sphere of radius R centered at $\mathbf{c}$. If the center of a ball or a sphere is the origin or when the position of the center of a ball is not important, we omit $\mathbf{c}$ and write $B^n(R)$ for a ball of radius R and $S^{n-1}(R)$ for a sphere of radius R; we denote the n-dimensional ball of the radius 1 centered at the origin as B^n and we denote n-sphere of the radius 1 centered at the origin as S^{n-1}.

2.2 Lattices

Lattice is a discrete subgroup of the additive group $\mathbb{R}^d$. It can be pictured as a regular infinite grid in the d-dimensional space. As a lattice is a subgroup of $\mathbb{R}^d$, it consists of d-dimensional vectors and any sum or difference of two lattice vectors is a lattice vector again. A typical example of a lattice is the set $\mathbb{Z}^n$ of all vectors with integer coefficients in $\mathbb{R}^n$.

Since a lattice is discrete, it can be described by a finite number of linearly independent vectors. Thus, there is the following alternative definition of a lattice.

Definition 2.1. Let n and d be two positive integer numbers and let $n \leqslant d$. Let $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^d$ be n linearly independent vectors. Then, a *lattice* $\Lambda \subset \mathbb{R}^d$ can be de-

fined as a set of all integer linear combinations of the vectors from $\mathbf{B}$:

$$\Lambda = \mathscr{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) := \Big\{ \sum_{i=1}^{n} x_i \mathbf{b}_i \;\Big|\; x_1, \ldots, x_n \in \mathbb{Z} \Big\}. \tag{2.1}$$

The integers d and n from the definition of a lattice are called the *dimension* and the *rank* of the lattice Λ, respectively. When $n = d$, we say that the lattice Λ has *full-rank*.

The vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ form a *basis* of Λ. Denote as $\mathbf{B}$ the matrix composed of the basis vectors. Then, we can rewrite (2.1) in a more compact form:

$$\Lambda = \{\mathbf{Bx} \mid \mathbf{x} \in \mathbb{Z}^n\}.$$

Any lattice of rank bigger than 1 has infinitely many bases, that can be transformed from one to another by unimodular transformations. Let $\mathbf{U} \in \mathbb{Z}^{n \times n}$ be a unimodular matrix, i.e., a square integer matrix with determinant $+1$ or -1. Then, $\mathbf{U}$ is invertible over integers and $\mathbf{U}\mathbb{Z}^n = \mathbb{Z}^n$. Therefore, if we apply a unimodular transformation to a basis $\mathbf{B}$ of a lattice Λ, we get an equivalent basis $\mathbf{B'} = \mathbf{BU}$ of the same lattice.

When considering integer lattices, sometimes it is convenient to use the *Hermite Normal Form*.

Definition 2.2. Let H be a matrix with integer coefficients. We say that $\mathbf{H}$ is in *Hermite Normal Form* (HNF), if it satisfies the following conditions:

1. $\mathbf{H}$ is an upper triangular matrix;
2. all diagonal elements of $\mathbf{H}$ are strictly positive;
3. for all i, all the elements to the right from the diagonal are smaller than the corresponding diagonal element h_{ii}.

The HNF can be efficiently computed for any matrix with integer coefficients using a series of unimodular transformations (see [Coh13] for the algorithm). Moreover, for any integer matrix $\mathbf{A}$, there exists a unique unimodular matrix $\mathbf{U}$, such that $\mathbf{H} = \mathbf{AU}$ is the Hermite Normal Form of $\mathbf{A}$. For lattices, it means that an integer lattice $\Lambda \in \mathbb{Z}^n$ has a unique basis in the Hermite Normal Form, that can be efficiently computed starting from any basis of the lattice. It can be useful, for example, if we need to list all the integer lattices with some particular properties.

For any basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ of the lattice Λ, we call the parallelepiped, spanned by the vectors of $\mathbf{B}$, the *fundamental parallelepiped* of $\mathbf{B}$ and denote it as $\mathscr{P}(\mathbf{B})$:

$$\mathscr{P}(\mathbf{B}) = \{\mathbf{Bx} \mid x_i \in [0; 1)\}.$$

Using the notion of the fundamental parallelepiped, we can define an important lattice invariant — a lattice volume.

Definition 2.3. Let $\mathbf{B} \in \mathbb{R}^{d \times n}$ be a basis of a lattice $\Lambda \subset \mathbb{R}^d$. The *volume* of the lattice Λ, denoted $\mathrm{vol}(\Lambda)$, is the volume of the fundamental parallelepiped $\mathscr{P}(\mathbf{B})$.

The volume of a lattice is a lattice invariant, because it is the same for any basis of the lattice. This follows directly from the fact that any lattice basis can be transformed into any other lattice basis by a unimodular transformation. Geometrically, the volume of the lattice can be seen as a characterisation of the density of lattice points in the space, i.e., the smaller the lattice volume is, the denser the lattice is.

Given a basis $\mathbf{B}$ of a lattice Λ, the volume of the lattice can be computed as a square root of the determinant of the Gram matrix $\mathbf{B}^t\mathbf{B}$. In the special case of a full-rank lattice, the volume of the lattice is equal to the absolute value of the determinant of the basis,i.e., $\mathrm{vol}(\Lambda) = |\det(\mathbf{B})|$.

Another way to compute the volume of a lattice is to compute the corresponding *Gram-Schmidt orthogonalization* of its basis.

Definition 2.4. Let $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^d$ be an ordered set of n vectors. Denote the corresponding matrix as $\mathbf{B}$. For any vector $\mathbf{v}$ and any positive integer $i < n$, denote the orthogonal projection of $\mathbf{v}$ over $\text{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^\perp$ as $\pi_i(\mathbf{v})$. Then, the corresponding Gram-Schmidt vectors for $\mathbf{b}_1, \ldots, \mathbf{b}_n$ are given by

$$\mathbf{b}_i^* := \pi_i(\mathbf{b}_i) = \mathbf{b}_i - \sum_{j=1}^{i-1} \frac{\mathbf{b}_j^t \mathbf{b}_i}{\mathbf{b}_i^t \mathbf{b}_i} \cdot \mathbf{b}_j,$$

for all $i \in \{1, \ldots, n\}$.

Note that the vectors, obtained by the Gram-Schmidt orthogonalization, depend on the order of vectors in the set $\{\mathbf{b_1}, \ldots, \mathbf{b}_n\}$. We denote the matrix, composed of the Gram-Schmidt vectors of $\mathbf{B}$ as $\mathbf{B}^*$. The Gram-Schmidt vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ are pairwise orthogonal, i.e., for $i \neq j$, $(\mathbf{b}_i^*)^t \mathbf{b}_j^* = 0$. If $\mathbf{B}$ is a basis of a lattice Λ, then the volume of Λ can be computed as a product of norms of the Gram-Schmidt vectors of $\mathbf{B}$:

$$\text{vol}(\Lambda) = \prod_{i=1}^{n} \|\mathbf{b}_i^*\|.$$

An important lattice parameter is the length of its shortest non-zero vector, also called the *first minimum* of a lattice. Another way to describe the first minimum is to consider the closed ball centered at the origin. Since lattices have a discrete structure, if the radius of the ball is sufficiently small, it does not contain any lattice point except the origin.

Definition 2.5. For a lattice Λ, the minimum radius $\lambda_1(\Lambda)$ such that the closed ball of radius $\lambda_1(\Lambda)$, centered at the origin, contains at least one lattice point except the origin, is called the first minimum of Λ.

The estimation of the length of the shortest non-zero lattice vector is a central question in the study of lattices. In this preliminary section, we consider existing bounds on the length of the shortest non-zero vector of a lattice, that depend only on the value of the volume of the lattice and does not depend on its particular shape.

First, there is obviously no lower bound on the value of the first minimum. Consider the following family of two-dimensional lattices of volume 1 with a basis that consists of two orthogonal vectors, one of length x, another of length $\frac{1}{x}$. We can make one of the two vectors arbitrarily small by taking a huge value of x without changing the volume of the lattice.

On the other side, if we fix the volume of the lattice, the shortest lattice vector obviously can not be arbitrarily big. The Minkowski's theorem gives the following upper bound on the first minimum of lattices with a fixed volume.

Theorem 2.1 (Minkowski's theorem)**.** For any lattice Λ of rank n, there exists a lattice vector $\mathbf{v} \neq \mathbf{0}$, such that

$$\|\mathbf{v}\| \leqslant \sqrt{n} \cdot \text{vol}(\Lambda)^{1/n}.$$

The proof of Minkowski's theorem can be found in [Cas12].

The Minkowski's theorem gives an upper bound on the *Hermite constant*, that is defined as the supremum of the ratio $\frac{\lambda_1(\Lambda)}{\text{vol}(\Lambda)^{1/n}}$ over all full-rank n-dimensional lattices. It is tight up to the constant factor. The following asymptotic upper bound on the Hermite constant, that can be found in [KL74], is the tightest upper bound on the Hermite's constant that we aware about:

$$\gamma_n \leqslant \frac{1.744n}{2\pi e} \cdot (1 + o(1)). \tag{2.2}$$

2.3 Random lattice

For the theoretical analysis of lattice algorithms it is sometimes useful to heuristically assume that the input lattice behaves like a random one. In this section, we recall some basic facts about random lattices and their generation.

Probability measure on lattices. The natural probability measure on the set of all n-dimensional full-rank lattices of volume 1 was first introduced by Siegel [Sie45] in 1945. A precise description of the measure with all the proofs can be found in [Ajt02]. Here, for completeness, we recall the general idea of the construction and some basic properties of the measure.

Denote as $\mathscr{L}_n$ the set of all n-dimensional lattices $\Lambda \subset \mathbb{R}^n$. Denote as $\mathscr{B}_n$ the set of all sets of n linearly independent vectors:

$$\mathscr{B}_n = \{(\mathbf{b}_1, \ldots, \mathbf{b}_n) \mid \forall i \in \{1, \ldots, n\}, \mathbf{b}_i \in \mathbb{R}^n,\ \mathbf{b}_1, \ldots, \mathbf{b}_n \text{ are linearly independent}\}.$$

In other words, $\mathscr{B}_n$ is the set of all bases for n-dimensional full-rank lattices. Similarly, denote the set of all n-dimensional lattices of volume one as $\widehat{\mathscr{L}}_n$ and denote the set of all bases of the lattices from $\widehat{\mathscr{L}}_n$ as $\widehat{\mathscr{B}}_n$.

Let $G_n(\mathbb{R})$ denote the group of all invertible $n \times n$ matrices over $\mathbb{R}$, denote as $S_n(\mathbb{R})$ the group of all invertible $n \times n$ matrices over $\mathbb{R}$ with determinant ± 1, and denote as $U_n(\mathbb{Z})$ the group of all unimodular matrices, i.e. $n \times n$ integer matrices of volume ± 1.

Let $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ be a basis from $\mathscr{B}_n$. We can identify it with the $n \times n$ matrix $\mathbf{B}$ such that the i-th column of $\mathbf{B}$ is the vector $\mathbf{b}_i$. Thus, each elements from the set $\mathscr{B}_n$ can be identified with an invertible $n \times n$ matrix. Then, we can identify the set $\mathscr{B}_n$ with $G_n(\mathbb{R})$. Similarly, we can identify $\widehat{\mathscr{B}}_n$ with $S_n(\mathbb{R})$. This allows us to represent a lattice $\Lambda \in \mathscr{L}_n$ as the set of all the matrices from $G_n(\mathbb{R})$ whose columns form a basis of Λ. Two bases $\mathbf{B}_1$ and $\mathbf{B}_2$ of the same lattice Λ can be transformed into each other using a unimodular transformation, i.e., there exists $\mathbf{U} \in U_n(\mathbb{Z})$ such that $\mathbf{B}_1 = \mathbf{B}_2\mathbf{U}$. Then, two bases correspond to the same lattice if and only if they lie in the same left coset of $U_n(\mathbb{Z})$ in $G_n(\mathbb{R})$. Therefore, the set of all lattices $\mathscr{L}_n$ can be identified with the set of left cosets $G_n(\mathbb{R})/U_n(\mathbb{Z})$. Similarly, the set $\widehat{\mathscr{L}}_n$ of all lattices of volume 1 can be identified with $S_n(\mathbb{R})/U_n(\mathbb{Z})$.

For any matrix $\mathbf{B} \in G_n(\mathbb{R})$, denote as $\phi(\mathbf{B})$ the lattice whose basis is formed by the columns of $\mathbf{B}$. Also, for any lattice $\Lambda \in \mathscr{L}_n$ denote as $\phi^{-1}(\Lambda)$ the set of all matrices that correspond to the bases of Λ. For a set $V \subset \mathscr{L}_n$, we write $\phi^{-1}(V)$ for the set $\{\phi^{-1}(\Lambda) \mid \Lambda \in V\}$.

Then, we can define topology and measure on $\mathscr{L}_n$ and $\widehat{\mathscr{L}}_n$ by defining it for the set of all left cosets of the subgroup $U_n(\mathbb{Z})$ in $G_n(\mathbb{R})$ and $S_n(\mathbb{R})$, correspondingly.

$G_n(\mathbb{R})$ can be viewed as an open subset of $\mathbb{R}^{n^2}$. Then, $G_n(\mathbb{R})$ inherits the Euclidean topology of $\mathbb{R}^{n^2}$. It allows to define a topology on $\mathscr{L}_n$ in the following way: a set $V \in \mathscr{L}_n$ is open if and only if $\phi^{-1}(V)$ is open in $G_n(\mathbb{R})$.

Also, the set of all bases $G_n(\mathbb{R})$ has measure inherited from $\mathbb{R}^{n^2}$. The measure of a Borel set A in $G_n(\mathbb{R})$ is the n-dimensional volume of A. Then, we can define the measure on the Borel sets of the set of all lattices. If A is the Borel subset of $\mathscr{L}_n$, then its measure is equal to the measure of $\phi^{-1}(A)$.

As $S_n(\mathbb{R})$ is an open subset of $G_n(\mathbb{R})$, the topology and measure can be similarly defined for the set $\widehat{\mathscr{L}}_n$ of all n-dimensional lattice of volume 1. Denote this measure as μ_n.

μ_n is a probability measure, i.e., $\mu_n(\mathscr{L}_n) \leqslant \infty$ (see, e.g., [Ajt02, Lemma 12] for the proof). It is invariant under linear transformations with determinant ± 1 and it is a unique probability measure that possesses this property.

Using the described probability measure on lattices, one can derive various results about the average behavior of random lattices. One of the earliest results of that kind if the following theorem due to Siegel [Sie45].

Theorem 2.2 (Siegel)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a compactly supported Borel measurable function. Then,

$$\int\limits_{G_n} \sum_{\mathbf{v}\in\Lambda\setminus\{\mathbf{0}\}} f(\mathbf{v})d\mu_n = \int\limits_{\mathbb{R}^n} f(\mathbf{x})d\mathbf{x}.$$

We can apply Siegel's theorem to the characteristic function of some measurable subset S of $\mathbb{R}^n$. Then, we obtain that, for a random lattice, the expectation of a number of lattice points inside S is equal to the volume of S in $\mathbb{R}^n$.

The heuristic assumption, based on this corollary from Siegel's theorem, is called the Gaussian Heuristic. It is often used in the analysis of lattice algorithms (see, e.g. [HS07] for the usage of the Gaussian Heuristic for the analysis of the lattice enumeration algorithm).

Assumption 2.1 (Gaussian Heuristic)**.** For a set $S \subset \mathbb{R}^n$ and a full-rank n-dimensional lattice Λ, we assume that the number of lattice points inside S is given by

$$|S \cap \Lambda| = \frac{\mathrm{vol}(S)}{\mathrm{vol}(\Lambda)}.$$

Recall that the volume of a ball of radius R in the n-dimensional space is given by:

$$\mathrm{vol}(B^n(R)) = \frac{\pi^{n/2}}{\Gamma\left(\frac{n}{2}+1\right)}R^n, \tag{2.3}$$

where Γ is Euler's gamma function. Using Stirling's approximation for the gamma function, we obtain the following asymptotic formula:

$$\mathrm{vol}(B^n(R)) \sim \frac{1}{\sqrt{\pi n}} \cdot \left(\frac{2\pi e}{n}\right)^{n/2} R^n. \tag{2.4}$$

The Gaussian Heuristic together with the asymptotic formula for the volume of an n-dimensional ball imply the following important properties of the behavior of random lattices:

- for a random n-dimensional lattice Λ, the length of the shortest non-zero vector is equal to $\Theta(\sqrt{n})\,\mathrm{vol}(\Lambda)^{1/n}$, i.e., the upper bound for the first minimum, given by Minkowski's theorem, becomes also a lower bound under the Gaussian Heuristic;
- for any vector $\mathbf{t} \in \mathbb{R}^n$, the expected distance from $\mathbf{t}$ to a random lattice Λ is equal to $\Theta(\sqrt{n})\,\mathrm{vol}(\Lambda)^{1/n}$;
- for an n-dimensional random lattice Λ, the expected number of lattice points inside the ball of radius $O(\sqrt{n})\,\mathrm{vol}(\Lambda)^{1/n}$ is exponential in n.

Although the Gaussian Heuristic seems to predict well the behavior of lattices that are "random enough" (see, e.g., [BGJ14, Section 4] for the experimental results on the Gaussian Heuristic), it does not necessarily hold for any lattice. One notable exception is the integer lattice. In [MO90], the authors estimate the number of points in $\mathbb{Z}^n$ inside a ball of radius, proportional to $\sqrt{n}$. They show that for the integer lattice, the number of the points inside the ball varies by exponential factors depending on the position of the center of the ball, which is not consistent with the Gaussian Heuristic.

In [GM03], Goldstein and Mayer have proposed the following simple strategy to sample a random lattice. First, notice that if we fix the dimension and the volume of a lattice, we are left only with a finite number of integer lattices. The algorithm selects an integer lattice with a fixed volume uniformly at random and then rescales it in order to obtain a

lattice of a volume 1 with rational coefficients. In [GM03], it is shown that the distribution, generated by that algorithm, tends to the natural probability distribution on the set of all lattices in $\mathbb{R}^n$ of volume 1 that was defined by Siegel.

Generation of a random integer lattice with a fixed volume is very easy when the volume is a prime number. Consider lattices with a prime volume p. In that case, the Hermite Normal Form of a lattice has one of the following n shapes:

$$\begin{pmatrix} 1 & & & 0 \\ & \ddots & & \\ & & 1 & \\ 0 & & & p \end{pmatrix}, \begin{pmatrix} 1 & & & & 0 \\ & \ddots & & & \\ & & 1 & & \\ & & & p & a_1 \\ 0 & & & 0 & 1 \end{pmatrix}, \dots, \begin{pmatrix} p & a_1 & \dots & a_{n-1} \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & 1 \end{pmatrix}, \tag{2.5}$$

where a_i's are integers in the range $\{0, \dots, p-1\}$. Then, sampling a random lattice with a prime volume p can be done in the following two steps:

1. choose one of n possible types of HNFs, presented by (2.5);
2. assume that at the previous step i-th type of the HNF was chosen; then, sample each of the numbers $a_1, \dots, a_{n-i}$ independently from the uniform distribution on $\{0, \dots, p-1\}$.

In order to obtain the uniform distribution on the set of all lattices, the probability to choose i-th type of the lattice should be equal to the fraction of the HNFs of i-th type among all the integer lattices with the volume p. There is one lattice of the first type, there are p lattices of the second type , p^2 lattices of the third type, and so on. The whole number of integer lattices whose volume is equal to p is $\sum_{k=0}^{n-1} p^k$. Then, to obtain the desired distribution, the first step of sampling can be performed in the following way. First, we sample an integer number from the uniform distribution on $\left\{1, 2, \dots, \sum_{k=0}^{n-1} p^k\right\}$. If the sampled number falls into the interval from $(\sum_{k=0}^{i-1} p^k + 1)$ to $\sum_{k=0}^{i} p^k$, then we choose i-th type of the HNF.

The algorithm that we have just described samples exactly the uniform distribution on the integer lattices with a fixed prime volume. When the dimension n and the volume p are large, the overwhelming amount of lattices have the HNF that is given by the last matrix from (2.5). In that case, the algorithm can be simplified: since the probability of choosing the n-th type of HNF is close to 1, we can omit the first step, i.e. we can always choose the n-th type of HNF and the resulting distribution will be close to the uniform. Then, the problem of sampling a random integer lattice of dimension n with a large prime volume p boils down to sampling $n-1$ integer numbers from the set $\{0, \dots, p-1\}$ uniformly at random.

2.4 Computational lattice problems

In this section, we give the formal definitions of the two fundamental computational lattice problems: the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP), and their variations, and recall some results on their theoretical and practical complexity.

For all computational lattice problems and lattice algorithms that we consider, the input of the problem is a basis $\mathbf{B}$ of a d-dimensional integer lattice Λ, i.e., a matrix with at most d^2 integer coefficients. We always assume that the size of the input and, therefore, the size of the coefficients of the matrix is polynomial in d, i.e., the $\|\mathbf{B}\|_{\max} = 2^{\mathrm{poly}(n)}$.

The Shortest Vector Problem. The most basic lattice problem is the Shortest Vector Problem. In SVP, we are given a basis of a lattice, which may consist of arbitrarily long vectors, and the goal is to find the shortest non-zero vector of the lattice.

Definition 2.6. Given a basis $\mathbf{B}$ of a lattice Λ, the *Shortest Vector Problem (SVP)* asks to find a shortest non-zero lattice vector, i.e., a vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{v}\| = \lambda_1(\Lambda)$.

The decision version of SVP is NP-hard under randomized reduction [Ajt98]. There is also a relaxation of SVP, that, instead of finding the shortest vector, asks to find its close approximation.

Definition 2.7. Let $\gamma > 1$. Given a basis $\mathbf{B}$ of a lattice Λ, the *γ-approximate Shortest Vector Problem (γ-SVP)* asks to find a non-zero lattice vector of length at most $\gamma\lambda_1(\Lambda)$, i.e., a vector $\mathbf{v} \in \Lambda$ such that $0 < \|\mathbf{v}\| \leqslant \gamma\lambda_1(\Lambda)$.

Another variation of SVP is called Hermite-SVP. It also asks to find a short lattice vector, but now, following the definition of the Hermite's constant, the length of the vector is compared to the d-th root of the volume of the lattice instead of the first minimum.

Definition 2.8. Let $\gamma > 1$. Given a basis $\mathbf{B}$ of a d-dimensional lattice Λ, the *γ-Hermite approximate Shortest Vector Problem (γ-Hermite-SVP)* asks to find a a vector $\mathbf{v} \in \Lambda$ such that $0 < \|\mathbf{v}\| \leqslant \gamma \operatorname{vol}(\Lambda)^{1/d}$.

It can be shown, that poly(n)-SVP and poly(n)-Hermite-SVP are equivalent problems. It is straightforwardly follows from Minkowski's theorem, that the solution of γ-SVP is also a solution of $\gamma\sqrt{n}$-Hermite-SVP. There is also an inverse reduction by Lovász. In [Lov86, Chapter 1.2], he showed that if there is an algorithm, that solves γ-Hermite-SVP in polynomial time, then there is an algorithm that solves γ^2-SVP in polynomial time.

Hardness of solving the approximate problems depends on the approximation factor. Approximating SVP within constant and even small polynomial factors remains a hard problem. In [Kho05], it is shown that, under certain assumptions on the polynomial hierarchy of complexity classes, there is no polynomial time algorithm that solves $n^{\frac{1}{2}-\varepsilon}$-approximate SVP for any ε. In the same time, it is very unlikely that the problem remains NP-hard for approximation factors bigger than $\sqrt{n}$, because in [AR05] it is shown that these problems lie in the intersection of NP and coNP.

Algorithms for SVP. For exponential approximation factors, the problem can be solved in polynomial time by the LLL algorithm [LLL82]. However, for the moment the polynomial-time algorithm is available only for exponential and slightly subexponential factors, i.e., the best achieved approximation factor by a polynomial time algorithm is $2^{\Theta(\frac{d\log(\log(d)))}{\log(d)})}$.

The fastest provable algorithms for approximating SVP within polynomial and better than polynomial factors require either superexponential running time (lattice enumeration algorithms [Kan83]) or both exponential time and space (the algorithm based on the Discrete Gaussian sampling [ADRSD15]). There are also heuristic algorithms for SVP, that are based on certain assumptions on behavior of random lattices and random lattice points. For the moment, there is a huge gap between the complexity of provable and heuristic algorithms for solving SVP. The fastest heuristic algorithm for SVP runs in time $\widetilde{O}(2^{0.292d})$, while the fastest provable algorithm has the time complexity $\widetilde{O}(2^d)$.

There are also blockwise lattice algorithms (e.g. [CN11, HPS11]) that can be seen as a mixture of the two approaches described above, which proceed by solving SVP on projected sublattices of dimension $\beta < d$. Such algorithms achieve approximation factors $\Theta(\beta^{\frac{d}{2\beta}})$ in time exponential in β.

The Closest Vector Problem. Another related hard lattice problem is the Closest Vector Problem. In CVP, we are given a basis of a lattice together with an arbitrary target point in the space, and the goal is to find the lattice point that is the best approximation of the target point.

Definition 2.9. Given a basis $\mathbf{B}$ of a lattice $\Lambda \subset \mathbb{R}^n$ and a target point $\mathbf{t} \in \mathbb{R}^n$, the *Closest Vector Problem (CVP)* asks to find a lattice vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{v}-\mathbf{t}\| = \min\limits_{\mathbf{x}\in\Lambda} \|\mathbf{x}-\mathbf{t}\|$.

The Closest Vector Problem, as SVP, also has an approximate version.

Definition 2.10. Let $\gamma > 1$. Given a basis $\mathbf{B}$ of a lattice $\Lambda \subset \mathbb{R}^n$ and a target point $\mathbf{t} \in \mathbb{R}^n$, the γ-*approximate Closest Vector Problem* asks to find a lattice vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{v} - \mathbf{t}\| \leqslant \gamma \cdot \min\limits_{\mathbf{x}\in\Lambda} \|\mathbf{x} - \mathbf{t}\|$.

In some sense, we can say that CVP is a "harder" problem than SVP, because there is a reduction from SVP to CVP, that preserves the dimension of the lattice [GMSS99], while there is no analogous reduction in the opposite direction. The decision version of the Closest Vector Problem is known to be NP-complete (see [MG12, Chapter 3.2]), and approximating CVP remains NP-hard to within almost polynomial factors. In [DKS98], it is shown that, for some $c > 0$, $n^{c/\log(\log(n))}$-approximate CVP is NP-hard. But, as for SVP, the $\sqrt{n}$-approximate problem lies in $\mathrm{NP} \cap \mathrm{coNP}$.

Algorithms for CVP. The dependence of the complexity of the algorithms for solving CVP on the desired approximation factor is very similar to the one of SVP.

For big approximation factors, there is the Babai Nearest Plane algorithm [Bab86]. The Babai Nearest Plane algorithm takes a basis of a lattice and a target point as an input and outputs the approximation of the target point. The quality of the approximation depends on the basis used. For example, the Babai Nearest Plane algorithm, given an LLL-reduced basis of a lattice, can solve $2^{\Theta(n)}$-CVP.

The fastest provable algorithms for solving exact SVP can be adapted for solving CVP as well. The lattice enumeration initially was suitable for solving both problems; the adaptation for CVP of the algorithm, based on the Discrete Gaussian sampling can be found in [ADSD15]. Also, as well as in case of SVP, there are heuristic sieving algorithms for solving CVP [Laa16] with much smaller, but still exponential complexity.

The Closest Vector Problem with Preprocessing. There is a variation of the CVP that is called the Closest Vector Problem with Preprocessing (CVPP). In CVPP, there are two stages. At the first stage, we are given a basis of a lattice and we can preprocess it in advance for an arbitrarily long amount of time. Then, at the second stage, target vectors are revealed. The goal is, using the information obtained at the preprocessing stage, efficiently find close vectors to the target points.

Definition 2.11. The *Closest Vector Problem with Preprocessing (CVPP)* asks to find a preprocessing function P and a decoding algorithm D that satisfy the following properties. Given as an input a basis $\mathbf{B}$ of a lattice $\Lambda \in \mathbb{R}^n$, the preprocessing function P returns a description $L = P(\mathbf{B})$ of the lattice Λ, such that, the decoding algorithm D, given as input the description L of the lattice Λ and a target point $\mathbf{t} \in \mathbb{R}^n$, returns a vector $\mathbf{v} \in \Lambda$ that minimizes the distance from $\mathbf{v}$ to the lattice: $\|\mathbf{v} - \mathbf{t}\| = \min\limits_{\mathbf{x}\in\Lambda} \|\mathbf{x} - \mathbf{t}\|$. The computation of the preprocessing function P can take arbitrary long time.

In the definition of CVPP, there is no restriction on the complexity of the computation of the preprocessing function. In CVPP, we assume that we are given unlimited computational resources for computing the function P. The complexity of the algorithm, that

solves CVPP, is measured by the complexity of the decoding algorithm D. We say that an algorithm solves CVPP in polynomial time, if it satisfies the following two conditions:

1. the description of a lattice, returned by the preprocessing function P has polynomial size;
2. the decoding algorithm D has polynomial time complexity.

Although it is very unlikely that such an algorithm exists, as in [Mic01], it is shown that, unless NP $\subset$ P/poly, there is no polynomial-time algorithm for CVPP.

The Closest Vector Problem with Preprocessing also has an approximate version, that can be defined similarly to the approximate CVP. The exact CVPP and the γ-approximate CVPP up to factor $\gamma = 2^{\log(d)^{1-\varepsilon}}$ are known to be hard to approximate under certain complexity assumptions [KPV12]. However, already for $O\left(\sqrt{\frac{d}{\log(d)}}\right)$-CVPP there is a polynomial time algorithm, described in [AR05].

Algorithms for CVPP. Besides the polynomial-time algorithm for the $O\left(\sqrt{\frac{d}{\log(d)}}\right)$-approximate Closest Vector Problem with Preprocessing, there are not so much other algorithms that separately consider CVPP. There is an heuristic algorithm for CVPP by Doulgerakis, Laarhoven, and de Weger [DLdW19], that precomputes the approximate Voronoi cell of an input lattice as the preprocessing step. This algorithm offers a trade-off between the time, spent on the preprocessing and the time of one query: it can solve one query in $\widetilde{O}(2^{\varepsilon d})$ time at the cost of spending $\left(\frac{1}{\varepsilon}\right)^{O(d)}$ on the preprocessing of the input lattice, for any $\varepsilon > 0$.

2.5 Lattice-related hard problems in cryptography

The two central average-case hard problems in lattice-based cryptography are the Short Integer Solution (SIS) and the Learning With Errors (LWE). Both problems were proved to be as hard on average as worst-case approximate lattice problems. SIS was introduced by Ajtai in 1996 [Ajt96], almost ten years later Regev proposed the LWE problem [Reg05]. Since their introduction, SIS and LWE have served as a foundation for a range of cryptographic constructions, like, e.g., collision resistant hash functions, signature schemes, public key encryption schemes, and fully homomorphic encryption schemes.

Both problems have ring-based analogues [Mic02, LPR10]. The ring-based problems are also as hard on average as certain approximate worst-case lattice problems, but for a restricted class of lattices called ideal lattices. Most of the SIS and LWE-based construction can be adapted to the ring settings. The ring-based problems allow more compact key sizes and more efficient encryption and decryption procedures then standard SIS/LWE-based constructions at the cost of having more restricted theoretical security foundations.

In this section, we recall the basic definitions, informally describe the hardness of the problems, and recall some basic cryptographic constructions based on LWE, SIS, and their ring analogues.

2.5.1 Short Integer Solution (SIS)

In 1996, Ajtai [Ajt96] described the first worst-case to average-case reduction for lattice problems and introduced the first cryptographic object based on hardness of lattice problem. Namely, Ajtai introduced the Short Integer Solution problem (SIS) together with related one-way function. Informally, in SIS, we are given many random vectors from a certain distribution, and the goal is to find a "short" integer combination them that is equal to zero.

Definition 2.12. Let $n, m, q \in \mathbb{Z}$, let $\beta > 0$ be a real number. For all $i \in \{1, \ldots, m\}$, let $\mathbf{a}_i$ be sampled from the uniform distribution on $\mathbb{Z}_q^n$. Let $\mathbf{A}$ be a matrix whose i-th column is $\mathbf{a}_i$. Given $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, the *Short Integer Solution problem (SIS$_{n,q,\beta,m}$)* asks to find a non-zero integer vector $\mathbf{z} = (z_1, \ldots, z_m)^t \in \mathbb{Z}^m$ of norm at most β such that

$$f_{\mathbf{A}}(\mathbf{z}) := \mathbf{A}\mathbf{z} = \sum_{i=1}^{m} z_i \cdot \mathbf{a}_i = \mathbf{0} \mod q. \tag{2.6}$$

The problem depends on the choice of the parameters n, q, β and m. The following corner-case observations illustrate the dependence:

- If the number of vectors m is too small, there might be no solution. For example, if $m = 1$ and we are given only one non-zero vector $\mathbf{a}_1$, there is no non-zero z such that $z \cdot \mathbf{a} = \mathbf{0}$.
- If the bound on the norm of the solution β is too big, the solution is easy to find. If there is no bound at all, i.e., $\beta = \infty$, we can find $\mathbf{z}$ using Gaussian elimination. For $\beta > q$, we always have a trivial solution $\mathbf{z} = (q, 0, \ldots, 0)^t \in \mathbb{Z}^m$.

Also, if we know a non-zero solution $\mathbf{z}'$ for some non-empty subset $\mathbf{A}' = \{\mathbf{a}'_1, \ldots, \mathbf{a}_{m'}\}$ of $\mathbf{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$, we can transform this solution into a solution for the original problem by zeroing out the vectors that are not presented in $\mathbf{A}'$. Thus, when m increases, the problem becomes easier. On the other hand, increasing n implies a harder problem.

In [Ajt96], Ajtai has shown that the Short Integer Solution problem with certain parameters is at least as hard as certain approximate lattice problems. More precisely, [Ajt96] provides polynomial-time reduction, i.e., it is shown that if there is a probabilistic polynomial time algorithm that solves SIS for certain parameters with a non-negligible probability, then, using this algorithm as an oracle, it is possible to construct a probabilistic polynomial time algorithm that solves approximate SVP and some other related problems for any lattice. The following theorem is an informal adaptation of this result.

Theorem 2.3 ([Ajt96, Theorem 1])**.** For any $m = \text{poly}(n)$, any $\beta > 0$ and $q \geqslant \beta \cdot \text{poly}(n)$, solving SIS$_{n,q,\beta,m}$ is at least as hard as solving the following three problems on arbitrary n-dimensional lattices with high probability:

- solving γ-approximate SVP,
- solving γ-unique SVP,
- solving γ-approximate SIVP,

where $\gamma = \beta \cdot \text{poly}(n)$.

Since the work of Ajtai, the parameters of the reduction have been significantly improved (see [MR07, GPV08, MP13]). The work of Gentry, Peikert, and Vaikuntanathan [GPV08] provides the reduction for the modulus $q = \beta \cdot \widetilde{O}(n)$ and the approximation factor $\gamma = \beta \cdot \widetilde{O}(n)$. In 2013, Micciancio and Peikert [MP13] achieved an almost optimal value for the modulus q. They proved the reduction for $q = \beta \cdot n^{\varepsilon}$ for any $\varepsilon > 0$, but the approximation factor γ of their reduction depends on the infinity norm of the SIS solution.

The SIS problem allows to build various cryptographic constructions. A simple example is the collision-resistant hash function. Consider the function $f_{\mathbf{A}}$, given by (2.6), defined on the set of all binary vectors. Assume that we can efficiently find a collision for $f_{\mathbf{A}}$, i.e., there is an efficient algorithm that finds two different binary vectors $\mathbf{z}_1, \mathbf{z}_2 \in \{0, 1\}^n$ such that $f_{\mathbf{A}}(\mathbf{z}_1) = f_{\mathbf{A}}(\mathbf{z}_2)$. Then, the difference $\mathbf{z}_1 - \mathbf{z}_2$ is the solution for the SIS problem with $\beta = O(\sqrt{n})$. Thus, finding a collision for $f_{\mathbf{A}}$ is as hard as solving the corresponding SIS problem. Besides collision-resistant hash functions, the SIS problem serves as a foundation for one-way functions, digital signatures and many other cryptographic primitives. See [P+16, Chapter 5] for an overview.

2.5.2 Learning With Errors (LWE)

In 2005, Regev introduced another important average-case problem called Learning With Errors (LWE) [Reg05]. In this work, Regev also proposed a public-key encryption scheme, based on LWE. Since that time, many other cryptographic constructions based on LWE have appeared, including very sophisticated ones, like fully homomorphic encryption schemes.

Informally, the LWE problem can be described as follows. We are given a list of m noisy equations, i.e., we are given m pairs $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ such that

$$\mathbf{a}_1^t \mathbf{s} = b_1' \mod q, \quad b_1 = b_1' + e \mod q, \tag{2.7}$$

$$\vdots \tag{2.8}$$

$$\mathbf{a}_m^t \mathbf{s} = b_m' \mod q, \quad b_m = b_m' + e \mod q. \tag{2.9}$$

The goal is to find the unknown vector $\mathbf{s} \in \mathbb{Z}_q^n$.

In order to define the LWE problem, we need to define the LWE distribution. It is parameterized by the three parameters: the dimension $n \in \mathbb{Z}$, the modulo $q \in \mathbb{Z}$, and an error distribution ξ. The error distribution ξ is some centered distribution over $\mathbb{Z}$ with variance $\alpha \cdot q$, where $\alpha \in (0; 1)$ is some constant. Often ξ is chosen to be the discrete Gaussian distribution on $\mathbb{Z}$ of width $\alpha \cdot q$.

Definition 2.13. Let $n, q \in \mathbb{Z}$ and let ξ be a distribution on $\mathbb{Z}$ with such that $\mathbb{E}\xi = 0$ and $\mathrm{Var}(\xi) = \alpha \cdot q$ for some $\alpha \in (0; 1)$. Let $\mathbf{s} \in \mathbb{Z}^q$.

Then, the LWE distribution with the parameters $\mathbf{s}$ and ξ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$, denoted as *the* $LWE_{\mathbf{s},\xi}$ *distribution*, is the distribution obtained by sampling $\mathbf{a}$ uniformly at random from $\mathbb{Z}^q$, sampling an error e from ξ and returning the following pair:

$$(\mathbf{a}, b) = (\mathbf{a}, \mathbf{a}^t \mathbf{s} + e). \tag{2.10}$$

There are two versions of the LWE problem: search-LWE and decision-LWE. In search-LWE, we are given many LWE samples and the goal is to recover the underlying secret vector $\mathbf{s}$. In decision-LWE, we need to distinguish the LWE distribution from the uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

Definition 2.14. Let $\mathbf{s}$ be sampled uniformly at random from $\mathbb{Z}_q^n$. The *search-*$LWE_{n,q,\xi,m}$ *problem* asks to recover the secret vector $\mathbf{s}$, given m independent samples from the $\mathrm{LWE}_{\mathbf{s},\xi}$ distribution.

Definition 2.15. Given m independent samples $\{(\mathbf{a}_i, b_i)\}_{i=1}^m \subset \mathbb{Z}_q^n \times \mathbb{Z}_q$ such that either each sample is drawn from the uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ or from the $\mathrm{LWE}_{\mathbf{s},\xi}$ distribution, the *decision-*$LWE_{n,q,\xi,m}$ *problem* asks to guess with a non-negligible probability the underlying distribution of the samples.

As in the SIS problem, the hardness of LWE also depends on the choice of the parameters. For example, notice that without the additional noise e_i, the search-LWE can be solved using Gaussian elimination. In general, if the variance of the noise is too small, the problem might be easy to solve.

However, there are sets of parameters for which LWE is hard. In his original work [Reg05], Regev have showed that solving the decision version of the LWE problem for a certain set of the parameters is at least as hard as solving approximate worst-case lattice problems on a quantum computer.

Theorem 2.4 ([Reg05, Theorem 1.1])**.** Let $n \in \mathbb{Z}$, $m = \mathrm{poly}(n)$, and $q = \mathrm{poly}(n)$. Let $\alpha \in (0; 1)$ be such that $\alpha \cdot q \geqslant 2\sqrt{n}$. Let ξ be the discrete Gaussian distribution of width

$\alpha \cdot q$. Then, if there exists an efficient algorithm for solving decision-LWE$_{n,q,\xi,m}$, then there exists an efficient quantum algorithm that solves γ-GapSVP and γ-approximate SIVP for $\gamma = \widetilde{O}\left(\frac{n}{\alpha}\right)$.

As for the moment there is no efficient algorithm that solves $\widetilde{O}(n/\alpha)$-approximate lattice problems on quantum computer, such a result implies that the LWE problem is hard. Besides the original quantum reduction, since 2005, many other results on the hardness of the LWE problem have appeared, including the classical reduction to worst-case lattice problems [Pei09], the results on hardness of LWE for various choices of the parameters n and q (see, e.g., [LM09, BLP$^+$13]), and the results that consider alternative distributions of the error and of the secret key (e.g., [MP13, Mic18]). These results show that LWE is quite robust problem and allows some flexibility in the choice of the parameters. For example, in [BLP$^+$13, Mic18], it is shown that LWE with a binary secret is as hard as usual LWE problem of slightly smaller dimension.

Theorem 2.5 ([Mic18]). The LWE problem with a binary secret $\mathbf{s} \in \{0,1\}^n$ of size $n = O(k \cdot \log(q))$ is as hard as the LWE problem with uniformly random secret in $\mathbb{Z}^q$.

Such a result is important for fully homomorphic encryption schemes based on LWE, because mostly they use binary secrets.

In [Reg05], Regev introduced the first public key encryption scheme based on the LWE problem. Then, many other cryptographic applications of LWE were proposed, for example, other public key cryptosystems, identity-based encryption schemes, and fully homomorphic encryption schemes. Here, to give a simple illustration of LWE applications in cryptography, we recall how basic LWE-based public-key cryptosystem from [Reg05, Section 5] works. Let n be the security parameter of the cryptosystem. Besides n, the cryptosystem is also parameterized by $N, q \in \mathbb{Z}$ and a probability distribution ξ with the support $\mathbb{Z}_q$.

- The **private key s** is a vector, sampled uniformly at random from $\mathbb{Z}_q^n$.
- The **public key** is a set of N samples from the LWE$_{\mathbf{s},\xi}$ distribution:

$$\{(\mathbf{a}_i, b_i)\}_{i=1}^N \subset \mathbb{Z}_q^n \times \mathbb{Z}_q.$$

- In order to encrypt one bit $m \in \{0,1\}$, we choose a random subset S of the set $\{1,\dots,N\}$. Then the **encryption** of the bit m is given by the pair

$$\left(\sum_{i\in S} \mathbf{a}_i, m + \sum_{i\in S} b_i\right) \in \mathbb{Z}_q^n \times \mathbb{Z}_q.$$

- Let $(\mathbf{a}, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ be an encryption of the bit $m \in \{0,1\}$. Then, to decrypt it, we compute $(b - \mathbf{a}^t\mathbf{s})$. If $(b - \mathbf{a}^t\mathbf{s})$ is closer to 0 than to $\lfloor \frac{q}{2} \rfloor$ modulo q, the **decryption** returns 0, otherwise, it returns 1.

The security of the cryptosystem follows from the fact that for a proper choice of the parameters, samples from the LWE distribution are computationally indistinguishable from the pairs of the same shape $(\widehat{\mathbf{a}}_i, \widehat{b}_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ such that $(\widehat{b}_i)_{i=1}^N$ are sampled from the uniform distribution on $\mathbb{Z}_q$.

2.6 Ring versions of SIS and LWE

Ring-SIS

There is a ring-based analogue of the SIS problem, described in Section 2.5.1, called Ring-SIS. Ring-SIS was also proved to be as hard as worst-case lattice problems, but for

some restricted families of lattices with additional structure. The cryptographic constructions based on Ring-SIS are more efficient and allow more compact representation thanks to the underlying ring structure.

To define Ring-SIS, we need to choose a ring. Most of the time, the Ring-SIS problem is defined using a quotient ring of the form $R := \mathbb{Z}[X]/(f)$, where f is a polynomial of degree n over integers. Each element of R can be represented by its residue modulo f, which is a polynomial of degree smaller than n with integer coefficients. Also, we assume that R is equipped with some norm $\|\cdot\|$. If $\boldsymbol{z} = (z_1, \ldots, z_m)^t \in R^m$ is a vector of elements from R, the norm of $\boldsymbol{z}$ is given by $\sqrt{\sum_{i=1}^{m} \|z_i\|^2}$.

Let q be a positive integer number. Define another ring $R_q := R/qR = \mathbb{Z}_q[X]/(f)$. The elements of R_q can be represented by polynomials of degree less than n with coefficients in $\mathbb{Z}_q$. Now, we can define the Ring-SIS problem that corresponds to the ring R and the modulus q.

Definition 2.16. Let $m \in \mathbb{Z}$. Let $a_1, \ldots, a_m \in R_q$ be independent uniformly random elements from R_q. The $R\text{-SIS}_{q,\beta,n,m}$ problem asks to find a vector $\boldsymbol{z} \in R^m$ such that

- $\boldsymbol{z}$ is short, i.e., $\|\boldsymbol{z}\| \leqslant \beta$;
- the scalar product of the vectors $\boldsymbol{a} = (a_1, \ldots, a_m)^t \in R_q^m$ and $\boldsymbol{z}$ is equal to zero in R_q:

$$f_{\boldsymbol{a}}(\boldsymbol{z}) := \boldsymbol{a}^t \boldsymbol{z} = \sum_{i=1}^{m} a_i \cdot z_i = 0 \in R_q. \tag{2.11}$$

Connection to SIS. The function of the form $f_{\mathbf{a}}$ given by (2.11) was first introduced by Micciancio in [Mic02]. In [Mic02], the ring R is defined as $\mathbb{Z}[X]/(X^n - 1)$. For this ring, the R-SIS problem can be seen as a special case of the usual SIS problem. Any polynomial $a(X) = a_0 + a_1 \cdot X + \cdots + a_{n-1} \cdot X^{n-1} \in R$ can be represented as an n-dimensional vector $\mathbf{a} = (a_0, \ldots, a_{n-1})^t \in \mathbb{Z}^n$ of its coefficients. For any $z(X) = z_0 + z_1 \cdot X + \cdots + z_{n-1} \cdot X^{n-1} \in R$, we define a circulant matrix $\mathbf{C}(z)$ as

$$\mathbf{C}(z) := \begin{pmatrix} z_0 & z_{n-1} & \cdots & z_1 \\ z_1 & z_0 & \cdots & z_2 \\ \vdots & \vdots & \vdots & \vdots \\ z_{n-2} & z_{n-3} & \cdots & z_{n-1} \\ z_{n-1} & z_{n-2} & \cdots & z_0 \end{pmatrix}. \tag{2.12}$$

Let p_{ab} be a product of two polynomials $a, b \in R = \mathbb{Z}/(X^n - 1)$:

$$p_{ab}(X) := a(X) \cdot b(X) = (a_0 + a_1 \cdot X + \cdots + a_{n-1} \cdot X^{n-1}) \cdot (b_0 + b_1 \cdot X + \cdots + b_{n-1} \cdot X^{n-1}) \in R. \tag{2.13}$$

Then, the product of two polynomials $a, b \in R$ can be computed as the product of a circulant matrix, corresponding to a, with the vector of coefficients of $\boldsymbol{b}$, i.e.,

$$\boldsymbol{p}_{ab} = \mathbf{C}(\boldsymbol{a})\mathbf{b}. \tag{2.14}$$

Recall that in SIS we are given a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and the goal is to find a short vector $\mathbf{z} \in \mathbb{Z}^m$ such that $f_{\mathbf{A}}(\mathbf{z}) = \mathbf{A}\mathbf{z} = 0 \in \mathbb{Z}_q$. Thus, computing the function $f_{\boldsymbol{a}}(\boldsymbol{z})$ (see (2.11)) is equivalent to computing the function $f_{\mathbf{A}'}(\mathbf{z})$, where $\mathbf{A}'$ is the concatenation of the circulant matrices, corresponding to the polynomial $a_1, \ldots, a_m$, i.e.,

$$\mathbf{A}' = \left(\mathbf{C}(a_1) \mid \mathbf{C}(a_2) \mid \ldots \mid \mathbf{C}(a_m) \right) \in \mathbb{Z}^{n \times nm}. \tag{2.15}$$

Then, the Ring-SIS problem, given by m polynomials $a_1, \ldots, a_m \in R_q$, can be seen as a structured SIS problem, given by the matrix $\mathbf{A}' \in \mathbb{Z}_q^{n \times nm}$.

One-way function. In [Mic02], Micciancio showed that for certain choice of parameters n, q, m and β and for the ring $R = \mathbb{Z}[X]/(X^n - 1)$, the function $f_{\boldsymbol{a}}$ is one-way, assuming that approximate lattice problems are hard in the worst case for cyclic lattices, i.e., lattices, that are invariant under cyclic rotations of the coordinates. That is, he showed that for a random $r \in R_q$ it is hard to find $\boldsymbol{z} \in R$ such that $f_{\boldsymbol{a}}(\boldsymbol{z}) = r$, unless solving approximate SIVP on cyclic lattices is easy. The paper [Mic02] lefts open the question whether it is possible to prove that $f_{\boldsymbol{a}}$ is collision resistant, like Ajtai's function $f_{\mathbf{A}}$ is.

Collision resistance. Later, in [PR06, LM06], it was proved that for the ring $\mathbb{Z}[X]/(X^n - 1)$ the function $f_{\boldsymbol{a}}$ is actually not collision resistant and the homogeneous Ring-SIS problem described in Definition 2.16 is easy to solve, because the ring has divisors of zero which can be used to construct collisions. On the other hand, the same works [PR06, LM06] show that this problem can be fixed by taking a ring of polynomials $R = \mathbb{Z}[X]/(f)$, where f is a carefully chosen irreducible polynomial. For example, the cyclotomic ring $\mathbb{Z}[X]/(X^n + 1)$, where n is a power of 2, allows to build a collision-resistant $f_{\boldsymbol{a}}$.

Ideal lattices. In [LM06], the authors also introduce a notion of ideal lattices, which can be seen as a generalization of the cyclic lattices. Let σ be a function that identifies polynomials from $R = \mathbb{Z}[X]/(f)$ with vectors in $\mathbb{Z}^n$, i.e., σ identifies a polynomial $g \in R$ with a vector of coefficients of $g \mod f$.

Definition 2.17. An *ideal lattice* is an integer lattice $\Lambda \subset \mathbb{Z}^n$ such that $\Lambda = \{\sigma(g) \mid g \in I\}$ for some monic polynomial f of degree n and an ideal $I \in R = \mathbb{Z}[X]/(f)$.

In the special case when $R = \mathbb{Z}[X]/(X^n - 1)$, the ideal lattices are cyclic lattices. In [LM06], the authors proved that, for a proper choice of the underlying ring, the function $f_{\boldsymbol{a}}$ is collision resistant, unless certain approximation lattice problems on ideal lattices are easy to solve.

Ring-LWE

In 2010, Lyubashevsky, Peikert, and Regev introduced the ring-LWE problem [LPR10], a ring-based analogue of LWE, and showed that solving ring-LWE is as hard as solving some approximate problems for ideal lattices.

Definitions and hardness. Let rings R and R_q be as defined in Section 2.6. That is, $R = \mathbb{Z}[X]/(f)$ and $R_q = R/qR$, where f is some irreducible polynomial,e.g. $f = X^n + 1$, where n is a power of 2. Let $\|\cdot\|$ be some norm defined on the elements of R and let ξ be a distribution on R that is concentrated on the small elements of R with regards to the chosen norm. Then, informally, the ring-LWE distribution can be defined as follows.

Definition 2.18. Let $s \in R_q$. Then, the R-LWE$_{s,\xi}$ distribution with support $R_q \times R_q$ is the distribution, obtained by sampling $a \in R_q$ uniformly at random from R_q, sampling $e \in R$ from ξ and returning the pair:

$$(a, b) = (a, (a \cdot s + e) \mod f) \in R_q \times R_q.$$

The ring-LWE problem, as LWE, has search and decision version. In the search version, we are asked to recover the secret s given a polynomial in n number of samples from the ring-LWE distribution. In decision version, we are given polynomially many independent samples either all from the uniform distribution on $R_q \times R_q$ or all from the ring-LWE distribution and the goal is to guess the distribution of the samples.

In [LPR10], the authors proved that the ring-LWE distribution for certain parameters is pseudorandom under the assumptions that approximate SVP is hard for ideal lattices. The following theorem informally recalls this result.

Theorem 2.6 ([LPR10, Theorem 1]). Assume that there is no polynomial-time quantum algorithm that solves approximate SVP in the worst case on ideal lattices for a fixed $\text{poly}(n)$ approximation factors. Then any $\text{poly}(n)$ number of samples drawn independently from the R-LWE distribution are pseudo-random, i.e., there is no polynomial-time algorithm that distinguish them from the uniform distribution with a non-negligible advantage.

Theorem 2.6 implies that solving the decision version of ring-LWE is hard. The proof of this result from [LPR10] has two parts. The first part is a quantum reduction from the worst-case approximate SVP to the search version of the ring-LWE problem; the second part shows that the ring-LWE distribution is pseudorandom under the assumption that the search version of ring-LWE is hard.

Basic cryptosystem. Many of the cryptographic constructions based on LWE can be adapted to ring-LWE settings with a gain in efficiency. To give some illustration of usage of ring-LWE in cryptography, we recall the idea of the very basic ring-LWE based cryptosystem, introduced in [LPR10].

- The **key generation** algorithm samples three random polynomials:
  - a from the uniform distribution on R_q;
  - $e \in R$ from the distribution ξ;
  - the secret s from some distributions on R concentrated on the elements with "small" norms.

  The key generation algorithm return $s \in R$ as a secret key and a pair $(a, b := a \cdot s + e) \in R_q \times R_q$ as a public key.
- The **encryption** algorithm samples three random polynomials $r, e_1, e_2 \in R$ with small norms from the distribution ξ. Let $M \in \{0,1\}^n$ be an n-bit message and let m be the polynomial whose coefficients correspond to the bits of M. Then, the encryption of the message M is a pair of polynomial $(u, v) \in R_q \times R_q$, given by

$$u = a \cdot r + e_1, \tag{2.16}$$

$$v = b \cdot r + e_2 + \lfloor \frac{q}{2} \rceil \cdot m. \tag{2.17}$$

- The **decryption** algorithm computes

$$v - u \cdot s = r \cdot e - s \cdot e_1 + e_2 + \lfloor \frac{q}{2} \rceil \cdot m. \tag{2.18}$$

  Under the appropriate choice of the distributions of the secret s and of the error vectors r, e_1, e_2, e, the coefficients of the polynomial $(r \cdot e - s \cdot e_1 + e_2) \in R$ have absolute values less than $\frac{q}{4}$ with high probability. Then, the i-th bit of the message M can be decrypted in the following way: $M_i = 0$ if the i-th coefficient of the polynomial $(v - u \cdot s)$ is closer to 0 than to $\lfloor \frac{q}{2} \rceil$, otherwise, $M_i = 1$.

The semantic security of the cryptosystem follows from the pseudo-randomness of the corresponding R-LWE distribution.

The complexity of the encryption and decryption algorithms is essentially the complexity of performing several polynomial multiplications. This operation can be performed very efficiently using FFT-techniques, which makes ring-LWE based constructions much faster than their standard LWE analogues.

Part I

Cylindrical sieving

Chapter 3

Necessary background on lattice algorithms

In this chapter, we provide all necessary background on lattice algorithms needed for the first part of this thesis. This chapter is organised as follows. First, in Section 3.1, we recall how sieving algorithms work. Second, in Section 3.2, we describe existing algorithms for finding short lattice vectors for lattices of small volumes. Then, in Sections 3.3 to 3.6, we describe various algorithmic techniques that we use later in this part. Finally, in Section 3.7, we provide some results on covering n-dimensional spheres and hypercylinders that underlie the estimation of the complexity of most of the sieving algorithms, and, in particular, of the algorithms, described in this thesis.

Notation. For any two vectors $\mathbf{x}, \mathbf{a} \in \mathbb{R}^n$, we write $\pi_{\mathbf{x}}(\mathbf{a})$ for the projection of $\mathbf{a}$ on the $(n-1)$-dimensional subspace of $\mathbb{R}^n$, orthogonal to $\mathbf{x}$, i.e., $\pi_{\mathbf{x}}(\mathbf{a}) = \mathbf{a} - \frac{\mathbf{a}^t\mathbf{x}}{\mathbf{x}^t\mathbf{x}}\mathbf{x}$.

Let $\mathbf{u} \in \mathbb{R}^n$ be a vector of norm 1. We denote as $C^n_{\mathbf{u}}(h, R)$ a n-dimensional hypercylinder of height h and radius R such that its axis is parallel to $\mathbf{u}$:

$$C^n_{\mathbf{u}}(h, R) = \Big\{\mathbf{x} \in \mathbb{R}^d \,|\, \mathbf{x}^t\mathbf{u} \in \big(-\frac{h}{2}, \frac{h}{2}\big),\ \|\pi_{\mathbf{u}}(\mathbf{x})\| \leqslant R\Big\}.$$

$C^n_{\mathbf{u}}$ denotes a hypercylinder with h and R both equal to one. If only h (or only R) is omitted, it means that only h (only R) is equal to one. When the value of $\mathbf{u}$ is clear from the context or irrelevant, we omit the subscript $\mathbf{u}$.

When we consider vectors in hypercylinders, sometimes it is convenient to rescale them in such a way that the projection of the vector on the cylinder axis is inside the interval $[0; 1)$ and the length of the projection on the subspace, orthogonal to the cylinder axis, is also equal to 1. Let $\mathbf{u} \in \mathbb{R}^n$ be a vector of norm 1, parallel to the cylinder axis and let $h > 0$ be the height of the hypercylinder. For any vector $\mathbf{x} \in \mathbb{R}^n$, we denote as $\mathrm{Rescale}_{\mathbf{u},h}(\mathbf{x})$ a vector which is given by

$$\mathrm{Rescale}_{\mathbf{u},h}(\mathbf{x}) = \frac{\mathbf{t}^t\mathbf{u}}{h} \cdot \mathbf{u} + \frac{1}{\|\pi_{\mathbf{u}}(\mathbf{x})\|} \cdot \pi_{\mathbf{u}}(\mathbf{x}).$$

All the computational problems and algorithms that we consider in this part have a dimension n as a parameter. We always assume that the input of an algorithm can be written using polynomial in n number of bits. In order to simplify the description of the complexity of algorithms, we use $\widetilde{O}$-notation, i.e., for any function f, $\widetilde{O}(f(n))$ denotes $O(f(n) \cdot \log(f(n))^c)$ for any constant c.

3.1 Sieving algorithms

The first sieving algorithm for solving the Shortest Vector Problem [AKS01] was proposed by Ajtai, Kumar, and Sivakumar (AKS) in 2001. This algorithm, as the most of its followers, is based on the following two statements about lattices:

1. a difference of two lattice vectors is also a lattice vector;
2. given any basis of a lattice, we can efficiently generate a long list of lattice vectors of huge, but bounded norm.

The first statement follows directly from the definition of a lattice. The second statement is true thanks to the LLL algorithm. Namely, given any basis of a lattice Λ, we can find a lattice basis, that consists of vectors of the norm, bounded by $2^{O(n)} \cdot \lambda_1(\Lambda)$, in a polynomial time by the LLL algorithm. Then, a long list of lattice vectors of bounded norm can be produced by taking various linear combinations of the vectors from the LLL-reduced basis. This can be done, for example, using Klein's algorithm [Kle00].

The two ideas, described above, were composed by Ajtai, Kumar, and Sivakumar into an algorithm for solving SVP that can be roughly described in the following way. The algorithm starts with generating a long list L of lattice vectors. Then, it considers all the pairwise differences $\mathbf{v} - \mathbf{w}$ of the vectors from the list L. Most of the time, the obtained difference $\mathbf{v} - \mathbf{w}$ is longer than both vectors $\mathbf{v}$ and $\mathbf{w}$, but if the list L is very long, some non-negligible proportion of the differences has a smaller norm. The algorithm keeps the shorter differences and stores them in a new list L'. Once the algorithm have considered all the differences, it is left with the list L' of lattice vectors that are shorter than the vectors from the initial list L. If the list L' is sufficiently long, the algorithm can repeat the procedure and obtain even shorter vectors. In [AKS01], it is shown that if the size of the initial list is $2^{\Theta(n)}$, then the algorithm can perform $\Theta(n)$ iterations of sieving which allows to find a vector of length $\lambda_1(\Lambda)$ in the end.

The algorithm of Ajtai, Kumar, and Sivakumar was the first simply exponential algorithm for solving SVP (the previous fastest SVP algorithm was Kannan's enumeration of the time complexity $2^{O(n \log(n))}$), but for a long time it was considered as not practical due to presumably high constant in the exponent and due to huge memory requirements.

Then, in 2008, the things have changed with the appearance of the work [NV08] of Nguyen and Vidick. In [NV08], the authors, first, have shown that the constant in the exponent of the complexity of the AKS sieve is actually not so huge, and, second that the worst-case analysis of the AKS sieve does not predict well the behavior of the algorithm in practice. For the average-case analysis, they have proposed to use a natural heuristic assumption on the distribution of lattice vectors. The heuristic complexity of the algorithm, described in [NV08], is $(4/3)^{n+o(n)}$ for the time and $(4/3)^{n/2+o(n)}$ for the memory.

We give a more detailed description of the Nguyen-Vidick (NV) sieve here, as further in this thesis its variation is used as a building block for constructing other lattice algorithms.

3.1.1 Nguyen–Vidick sieve

The Nguyen–Vidick sieve starts with generating a long list of lattice vectors by sampling them from the discrete Gaussian distribution with a huge variance. In order to do so, Klein's algorithm is used (see [GPV08] for a detailed description of the distribution, generated by Klein's algorithm). Then, the algorithm proceeds by iteratively applying the sieving step, described in Algorithm 3.1, to the obtained list of lattice vectors.

One iteration of the Nguyen–Vidick sieve separates the vectors from the input list L into two categories: centers and all the other vectors. At the beginning, the algorithm creates an empty list of centers C. Then, for each vector $\mathbf{v}$ from the input list L, the algorithm searches in the list C for a vector $\mathbf{c}$, such that the difference $\mathbf{v} - \mathbf{c}$ is short. If

such a center is found, the corresponding difference goes to the new list L', otherwise, if there is no suitable center for $\mathbf{v}$, the vector $\mathbf{v}$ goes to the list C and becomes a new center itself. The size of the new list of shorter vectors L', returned by the algorithm, is equal to $|L| - |C|$. Therefore, in order to find out how many lattice vectors we need to perform a certain number of iterations and obtain a non-empty list in the end, it is important to estimate the number of centers, lost at one iteration.

Algorithm 3.1: One iteration of the Nguyen–Vidick sieve

input : a list $L \subset \Lambda$ of lattice vectors, such that for all $\mathbf{v} \in L$, $\|\mathbf{v}\| \leqslant R$, a parameter $\gamma \in (0; 1)$.

output: a list $L' \subset \Lambda$ of lattice vectors, such that for all $\mathbf{v} \in L'$, $\|\mathbf{v}\| \leqslant \gamma \cdot R$.

```
NVsieve(L, γ):
    L' ← ∅, C ← ∅
    for v ∈ L' do
        if (∃c ∈ C such that ‖v − c‖ ⩽ γ · R) then
            L ← L ∪ {v − c}
        else
            C ← C ∪ {v}
    return L'
```

In order to estimate the size of the list of centers, Nguyen and Vidick proposed to assume that the vectors from the list L behave like uniformly distributed on a sphere. Assumption 3.1 presents the simplified version of the original assumption (see [NV08, Heuristic, page 14]).

Assumption 3.1. The vectors from the list L in Algorithm 3.1, after being normalized, behave as if they were sampled independently from the uniform distribution on the sphere S^{n-1}.

In [NV08], it is shown that, if the decrease rate γ is close to 1, then the number of centers, lost at one iteration, is equal to $(4/3)^{n/2}$. This is the number of random spherical caps of the angular radius $\frac{\pi}{3}$, needed to cover the surface of the sphere S^{n-1} with high probability.

The algorithm starts with generating a list of lattice vectors of length $2^{O(n)} \cdot \lambda_1(\Lambda)$, each iteration decreases the length of vectors from the list by the constant factor γ, therefore, the number of iterations is linear in n. Then, in order to obtain a vector of length λ_1, the size of the initial list should be $O(n) \cdot (4/3)^{n/2}$.

The time complexity of one iteration of the Nguyen–Vidick sieve is equal to $|L| \cdot |C|$. Since the number of iterations is polynomial and since $|L| = (4/3)^{n/2+o(n)}$, the time complexity of the algorithm is $(4/3)^{n+o(n)}$.

After the seminal work of Nguyen and Vidick, various improvements of the heuristic sieving were proposed (see, e.g. [WLTB11, ZPH13, LdW15, Laa15b, BDGL16, BLS16, HKL18]). We briefly describe the following four lines of research in heuristics sieving.

Leveled sieving. The Nguyen-Vidick sieve works by creating a list of centers. In [WLTB11], in order to decrease the time, spent on looking for a suitable pair for a lattice vector, the authors proposed to use two levels of centers instead of one. The goal of the centers of the first level is to divide the search space into the smaller chunks. Now the algorithm has two parameters γ_1 and γ_2 for each level of sieving, which allows to obtain a time-memory trade-off. Then, a faster three-leveled-sieve, based on a similar idea, was proposed in [ZPH13]. Both algorithms are faster than the Nguyen–Vidick sieve, but at the cost of having bigger

memory complexity. In [Laa15a, Chapter 9], it is shown that taking more levels than three does not improve further the time complexity of sieving.

Overlattice sieving. The overlattice sieving algorithm, introduced in [BGJ14], also can be considered as a sieving algorithm, though it is rather far from all the other algorithms, described in this section. It creates a tower of overlattices, related to the input lattice, such that in the first overlattice we can efficiently enumerate all the short vectors, and the last lattice in the tower is the input lattice. Then, the algorithm enumerates the short vectors in the first lattice in the tower, considers the sums of the obtained vectors, and keeps those that lie in the overlattice of the next level. Proceeding iteratively in such a way, the algorithm obtains the short vectors in the original lattice in the end. The algorithm is based on the Gaussian Heuristic. The time complexity of the algorithm is $2^{0.3774n+o(n)}$, the memory complexity is $2^{0.2925n+o(n)}$, which makes the overlattice sieving faster than leveled sieving algorithms.

Nearest-neighbor search. The time complexity of sieving is defined by the time, needed to find a vector, close to a given one, in the list of N points. The Nguyen-Vidick sieve performs this step just by going through all the vectors in the list, until the desired pair is found or until all the list is checked, which takes $O(N)$ time. There is a line of research that studies how the process of finding a suitable pair in the list of vectors can be accelerated using the techniques for the nearest-neighbor search on a sphere, see, e.g., [LdW15, Laa15b, BDGL16]. The currently fastest heuristic sieving algorithm [BDGL16] follows this line of research. It uses locality-sensitive filtering for the nearest neighbor search on a sphere and achieves the time complexity $(3/2)^{n/2+o(n)} \approx 2^{0.292n+o(n)}$.

Tuple sieving. In order to decrease the memory requirement of sieving algorithms, in [BLS16], the authors propose, instead of checking only differences of vectors from the list, consider linear combinations of more than only two vectors. They show that considering triplets of vectors allows to decrease the memory complexity up to $2^{0.1887n+o(n)}$, while the running time of the algorithm is $2^{0.4812n+o(n)}$. For further improvements in tuple sieving, see, for example, [HKL18].

3.2 Solving hard lattice problems for lattices with a small volume

In this section, we recall the algorithm by Cheon and Lee [CL15] that solves approximate SVP for lattices with a small volume. The technique, described in [CL15], exploits the structure of the Hermite Normal Form of a lattice with a small volume in order to speed up lattice algorithms.

Computing the Hermite Normal Form allows to find a sublattice with a volume that does not exceed the volume of the initial lattice. Then, if the volume of the initial lattice is not very huge, a relatively short vector in the initial lattice can be found by solving SVP for the sublattice of smaller dimension.

In this thesis, we are interested in applying Cheon's technique to a lattice with a small prime volume. The Hermite Normal Form of a lattice with a prime volume p is given by (2.5). By taking a sublattice, spanned by the first $k < n$ columns of the HNF of a

prime volume lattice, we obtain a sublattice of smaller dimension with the same volume p:

$$\left(\begin{array}{cccc|ccc} p & a_1 & \dots & a_{k-1} & a_k & \dots & a_{n-1} \\ 0 & 1 & & 0 & 0 & \dots & 0 \\ \vdots & & \ddots & & \vdots & & \vdots \\ 0 & 0 & & 1 & 0 & \dots & 0 \\ \hline 0 & 0 & \dots & 0 & 1 & & 0 \\ \vdots & \vdots & & \vdots & & \ddots & \\ 0 & 0 & \dots & 0 & 0 & & 1 \end{array}\right). \tag{3.1}$$

In [CL15], the authors consider approximate SVP and they use the LLL algorithm for finding short vectors in the sublattice. In this work, our goal is to find a close approximation of the shortest vector of the lattice, so we replace LLL with the fastest heuristic algorithm for solving SVP that we are aware of [BDGL16].

Algorithm 3.2: Cheon and Lee technique [CL15] for a lattice with a small prime volume

input : a basis $\mathbf{B}$ of an n-dimensional lattice Λ with a prime volume, $\theta \in (0;1)$.
output: $\mathbf{v} \in \Lambda$ such that $0 < \|\mathbf{v}\| \leqslant \sqrt{\gamma_{\theta n}} \cdot \mathrm{vol}(\Lambda)^{\frac{1}{\theta n}}$

1 $(p, a_1, \dots, a_{n-1}) \leftarrow \mathrm{HNF}(\mathbf{B})$
2 $k = \lfloor \theta n \rfloor$
3 $\Lambda_k = \mathscr{L}(p, a_1, \dots, a_{k-1})$
4 $\mathbf{v} \leftarrow \mathrm{SVP}(\Lambda_k)$ ▷ sieving algorithm from [BDGL16]
5 **return** $\mathbf{v}$

Theorem 3.1 (see [CL15, Theorem 2])**.** Let n be a positive integer number and let $\theta \in (0;1)$. Let p be a prime number such that $p = 2^{cn}$ for some $c > 0$. Let Λ be an n-dimensional integer lattice with a prime volume p. Then, Algorithm 3.2, given as input a basis of Λ and the parameter θ, outputs a non-zero lattice vector $\mathbf{v}$ of length at most $\sqrt{\gamma_{\theta n}} \cdot 2^{c/\theta}$, where $\sqrt{\gamma_{\theta n}}$ is the Hermite constant of dimension θn. The time complexity of the algorithm is $\widetilde{O}\left(\left(\frac{3}{2}\right)^{\theta n/2}\right)$.

Proof. The time complexity and the length of the vector returned by the algorithm can be obtained straightforwardly from the complexity of the algorithm [BDGL16] and from the Minkowski's bound for an $n\theta$-dimensional lattice. □

3.3 Enumeration of lattice points in easy special cases

In this section, we consider the following problem. Let $R > 0$, $\mathbf{c} \in \mathbb{R}^n$ and let Λ be an n-dimensional full-rank lattice. The goal is to enumerate all the lattice points inside the ball $B^n(\mathbf{c}, R)$. In general, for an arbitrary lattice, this problem can be solved by Schnorr-Euchner's enumeration algorithm in a superexponential time $\widetilde{O}(2^{n\log(n)})$ using polynomial amount of memory. However, for some species of lattices, enumeration can be performed much more efficiently. In this section, we consider two such special cases: the integer lattice $\mathbb{Z}^n$ and a lattice with a quasi-orthonormal basis.

3.3.1 Integer lattice

Consider the complexity of enumerating all integer points in the n-dimensional space that lie inside a ball, centred at the origin, of some fixed radius.

As an integer lattice is symmetric, we have to enumerate only points in with non-negative coordinates, all the other points can be obtained by changing the corresponding signs of the coordinates.

The points with non-negative coordinates in a set $\mathbb{Z}^n \cap B^n(R)$ can be enumerated recursively. First, we enumerate all integer points inside the ball of radius R in dimension one, i.e., we just list all non-negative integers smaller than R. Denote the obtained list as $S_1 := \{0, \dots \lfloor R \rfloor\}$. Assume that we have a list $S_{n-1} = \mathbb{Z}^{n-1} \cap B^n(R)$ of all the points inside the ball in dimension $n-1$. Then, we can construct $S_n = \mathbb{Z}^n \cap B^n(R)$ in the following way. Let $\mathbf{x} = (x_1, \dots, x_n)^t \in S_{n-1}$. Denote as $\mathbf{x}_k$ the n-dimensional vector formed by concatenating $\mathbf{x}$ with k: $\mathbf{x}_k := (x_1, \dots, x_n, k)^t$. Then, $\mathbf{x}_0 \in S_n$. For all $k \in S_1$, until the norm of $\mathbf{x}_k$ is smaller than R, we add $\mathbf{x}_k$ to S_n. Once we reach the value of k, such that $\|\mathbf{x}_k\|$ exceeds R, we switch to another point from S_{n-1}. The approach is summarized in Algorithm 3.3.

Algorithm 3.3: Enumerate integer points inside a ball.

input : a positive integer n, $R > 0$
output: all points from $B^n(R) \cap \mathbb{Z}^n$ with non-negative coordinates

```
1  enumerateIntegerPoints(n, R):
2      S_1 ← {0, 1, ..., ⌊R⌋}
3      for i ∈ {2, ..., n} do
4          S_i ← ∅
5          for (x = (x_1, ..., x_{i-1})^t ∈ S_{i-1}) do
6              k ← 0
7              x_k ← (x_1, ..., x_{i-1}, k)^t
8              while (‖x_k‖ < R) do
9                  S_i ← S_i ∪ {x_k}
10                 k = k + 1
11                 x_k = (x_1, ..., x_{i-1}, k + 1)^t
12     return S_n
```

In the following lemma, we show that the complexity of enumerating integer points inside a ball $B^n(R)$ essentially is defined by the number of points inside the ball $|\mathbb{Z}^n \cap B^n(R)|$.

Lemma 3.1. Algorithm 3.3, given as input a positive integer number n and $R > 0$, outputs a list $S_n = \mathbb{Z}^n \cap B^n(R)$ in time $\text{poly}(n) \cdot C_n(R)$ using $\text{poly}(n) \cdot C_n(R)$ memory, where $C_n(R)$ is equal to $|\mathbb{Z}^n \cap B^n(R)|$.

Proof. For any positive integer i denote $|\mathbb{Z}^i \cap B(R)|$ as $C_i(R)$. The memory complexity is bounded by the memory required to store $S_n = \mathbb{Z}^n \cap B^n(R)$, which contains $C_n(R)$ n-dimensional vectors. Therefore, is we assume that storing one integer number takes $\text{poly}(n)$ memory, the memory complexity of the algorithm is $\text{poly}(n) \cdot C_n(R)$.

The time complexity of the i-th iteration is $O(|S_i|)$. As size of all S_i's are smaller than the size of the final list S_n and since the number of iterations is n, the time complexity of the algorithm is $\text{poly}(n) \cdot C_n(R)$. □

Further in this part of the thesis, we need to enumerate points in $\mathbb{Z}^n$ inside a ball, centered at the origin, of radius, proportional to $\sqrt{n}$. In [MO90], the authors show that asymptotically size of the set $\mathbb{Z}^n \cap B^n(\sqrt{\alpha n})$ for any $\alpha > 0$ can be numerically estimated using the saddle-point method.

Theorem 3.2 (see [MO90, Section 3])**.** Let $\alpha > 0$ and let $n \to \infty$. Then, the number of n-dimensional integer points of norm smaller then $\sqrt{\alpha n}$ tends to $\nu(\alpha)^n$, where ν is a constant that depends only on α.

In this thesis, we are interested in the following two particular values of α:

$$|\mathbb{Z}^n \cap B^n(\sqrt{0.084n})| \sim 2^{n/2}, \quad |\mathbb{Z}^n \cap B^n(\sqrt{0.0418n})| \sim 2^{0.292n}. \tag{3.2}$$

3.3.2 Lattice with quasi-orthonormal basis

In this section, we recall that the Schnorr-Euchner's enumeration algorithm can be very efficient, if it is given a "nice" lattice basis as an input, i.e., a basis that is composed of almost orthogonal vectors. In general, obtaining such a basis for an arbitrary lattice is a hard problem. But, for any lattice, we can efficiently compute an almost orthogonal basis for a projected sublattice of a smaller dimension (see Section 3.6). Efficient enumeration of the projected sublattice can be a useful subroutine for more complicated lattice algorithms.

This section is organized as follows. First, we shortly describe how the complexity of Schnorr-Euchner's enumeration algorithm depends on the quality of an input basis. For a more detailed analysis of the enumeration algorithms, see, e.g., [HS07, GNR10]. Then, we give a formal definition of a quasi-orthonormal basis, and, finally, we compute the complexity of the enumeration, assuming that the algorithm obtains such a basis as input.

Schnorr-Euchner's enumeration. Given a basis $\mathbf{B}$ of an n-dimensional lattice Λ, Schnorr-Euchner's enumeration algorithm [SE94] lists all lattice vectors that lie inside the ball $B^n(R)$.

The algorithm enumerates lattice vectors inside the ball by creating the enumeration tree. The i-th level of the tree consists of all the vectors in the projected sublattice $\pi_{n-i+1}(\Lambda)$ of norm at most R. The time complexity of the enumeration is N polynomial-time operations, where N is the total number of the tree nodes:

$$N = \sum_{i=1}^{n} |\pi_{n-i+1}(\Lambda) \cap B^i(R)|. \tag{3.3}$$

The number of vectors of bounded norm in the projected sublattices can be estimated using the Gaussian Heuristic. As the volume of the i-th projected sublattice is $\mathrm{vol}(\pi_i(\Lambda)) = \prod_{k=i}^{n} \|\mathbf{b}_k^*\|$, for the number of tree nodes we obtain the following estimate:

$$N = \sum_{i=1}^{n} \frac{\mathrm{vol}(B^i(R))}{\prod_{k=n-i+1}^{n} \|\mathbf{b}_k^*\|} \tag{3.4}$$

Therefore, the complexity of Schnorr-Euchner's enumeration algorithm depends on the shape of the given basis of the lattice, more precisely, on the norms of the Gram-Schmidt vectors corresponding to the input basis.

Quasi-orthonormal basis. There is a generalisation of the Hermite's constant, introduced by Rankin in 1953 [Ran53]. Let Λ be an n-dimensional lattice. We can say that Hermite's constant gives a bound on the smallest volume of a one-dimensional sublattice of a lattice. Then, Rankin's constant gives a similar bound for an m-dimensional sublattice for any $m < n$. The m-th Rankin invariant of the lattice Λ is the minimal ratio of the volume of an m-dimensional sublattice of Λ and of $\mathrm{vol}(\Lambda)^{m/n}$:

$$\sqrt{\gamma_{n,m}} := \min_{\substack{L_m \text{ is sublattice of } \Lambda, \\ \dim(L_m)=m}} \left(\frac{\mathrm{vol}(L_m)}{\mathrm{vol}(\Lambda)^{m/n}} \right).$$

Rankin's constant is the maximum of the Rankin's invariant over all n-dimensional lattices: $\sqrt{\gamma_{n,m}} := \max_\Lambda \gamma_{n,m}(\Lambda)$. When $m = 1$, then $\sqrt{\gamma_{n,1}} = \frac{\max_\Lambda \lambda_1(\Lambda)}{\mathrm{vol}(\Lambda)^{1/n}}$, and Rankin's constant becomes Hermite's constant.

As in [BGJ14], in order to measure how reduced is a basis $\mathbf{B}$ of a lattice Λ, we use the notion of Rankin factor that is closely related to Rankin's constant. An m-th Rankin factor of a basis $\mathbf{B}$ measures how far is the volume of the sublattice, spanned by the first m vectors of the basis from $\mathrm{vol}(\Lambda)^{m/n}$.

Definition 3.1. Let $\mathbf{B}$ be a basis of an n-dimensional lattice Λ and let $m \leqslant n$ be some positive integer number. The *Rankin factor* of $\mathbf{B}$ with index m is given by the following ratio:

$$\sqrt{\gamma_{n,m}(\mathbf{B})} := \frac{\mathrm{vol}(\mathscr{L}(\mathbf{b}_1, \ldots, \mathbf{b}_m))}{\mathrm{vol}(\Lambda)^{m/n}} = \frac{\mathrm{vol}(\Lambda)^{\frac{n-m}{n}}}{\mathrm{vol}(\pi_{m+1}(\Lambda))} = \frac{\mathrm{vol}(\Lambda)^{\frac{n-m}{n}}}{\|\mathbf{b}^*_{m+1}\| \cdot \cdots \cdot \|\mathbf{b}^*_n\|}. \tag{3.5}$$

We say that a basis $\mathbf{B}$ is quasi-orthonormal,if all its Rankin factors are bounded.

Definition 3.2. A basis $\mathbf{B}$ of an n-dimensional lattice Λ is called *quasi-orthonormal* if $\sqrt{\gamma_{n,m}(\mathbf{B})} \leqslant \mathrm{poly}(n)$ for all $m \in \{1, \ldots, n\}$.

Any basis, such that the length of its Gram-Schmidt vectors are almost equal to each other, would be quasi-orthonormal. However, the notion of reduction given by Definition 3.2 is very strong and even unachievable for most of the lattices. For example, an LLL-reduced basis achieve Rankin factors equal to $2^{O(n^2)}$.

The following lemma describes the complexity of Schnorr-Euchner's algorithm with a quasi-orthonormal basis as an input.

Lemma 3.2. Assume that the Gaussian Heuristic holds. Then, given a quasi-orthonormal basis $\mathbf{B}$ of an n-dimensional lattice Λ and a target radius $R = \beta\sqrt{\frac{n}{2\pi e}}\,\mathrm{vol}(\Lambda)^{1/n}$, where $\beta > 1$ is some constant, the Schnorr-Euchner's algorithm enumerates all lattice vectors of norm at most R using $e^{\frac{n\beta^2}{2e}} \cdot \mathrm{poly}(n)$ operations if $\beta < \sqrt{e}$, and using β^n operations if $\beta \geqslant \sqrt{e}$. Number of enumerated lattice vectors is equal to β^n.

Proof. Consider the complexity of Schnorr-Euchner's algorithm described by (3.4). If we multiply the numerator and the denominator of each summand in (3.4) by $\mathrm{vol}(\Lambda)^{i/n}$, where i is the number of the summand, we can rewrite it using Rankin factors in the following way:

$$N = \sum_{i=1}^{n} \frac{\mathrm{vol}(\Lambda)^{i/n}}{\mathrm{vol}(\pi_{n+1-i}(\Lambda))} \cdot \frac{\mathrm{vol}(B^i(R))}{\mathrm{vol}(\Lambda)^{i/n}} = \sum_{i=1}^{n} \sqrt{\gamma_{n,n-i}(\mathbf{B})} \cdot \frac{\mathrm{vol}(B^i(R))}{\mathrm{vol}(\Lambda)^{i/n}}.$$

Then, for a quasi-orthonormal basis we obtain:

$$N \leqslant \max_i \sqrt{\gamma_{n,n-i}(\mathbf{B})} \cdot \sum_{i=1}^{n} \frac{\mathrm{vol}(B^i(R))}{\mathrm{vol}(\Lambda)^{i/n}} = \mathrm{poly}(n) \cdot \sum_{i=1}^{n} \frac{\mathrm{vol}(B^i(R))}{\mathrm{vol}(\Lambda)^{i/n}}$$

Consider the sum

$$S_n = \sum_{i=1}^{n} \frac{\mathrm{vol}(B^i(R))}{\mathrm{vol}(\Lambda)^{i/n}}, \tag{3.6}$$

assuming that the target radius R is equal to $\beta \cdot \sqrt{\frac{n}{2\pi e}}\,\mathrm{vol}(\Lambda)^{1/n}$, where $\beta > 1$ is some constant. Using the asymptotic approximation for the volumes of the balls (see (2.4)), we obtain:

$$S_n \approx \sum_{i=1}^{n} \left(\sqrt{\frac{n}{i}} \cdot \beta\right)^i. \tag{3.7}$$

The expression $(\beta \cdot \sqrt{n/i})^i$ is maximized when $i = n\beta^2/e$. If $\beta > \sqrt{e}$, then the biggest summand of S_n is the last one. It is equal to $\frac{\mathrm{vol}(B_n(R))}{\mathrm{vol}(\Lambda)} = \beta^n$, which, under the

Gaussian heuristic, coincides with the number of lattice points inside the ball of radius R. if $R > \sqrt{\frac{n}{2\pi}} \cdot \mathrm{vol}(\Lambda)^{1/n}$, then we obtain:

$$N \leqslant n \cdot \max_i \gamma_{n,n-i}(\mathbf{B}) \cdot \max_j \left(\sqrt{\frac{n}{j}} \cdot \beta\right)^j \leqslant \mathrm{poly}(n) \cdot \beta^n.$$

Otherwise, if $\beta < \sqrt{e}$, the biggest summand is not the last one, and we obtain $N \leqslant \mathrm{poly}(n) \cdot e^{n\beta^2/2e}$. □

By Lemma 3.2, when the radius of the ball for enumeration is bigger than $\sqrt{\frac{n}{2\pi}}\,\mathrm{vol}(\Lambda)^{1/n}$, Schnorr-Euchner's algorithm, given a quasi-orthonormal basis of Λ as an input, can enumerate all the points inside the ball in time $\widetilde{O}(|B^n(R) \cap \Lambda|)$, which is optimal up to subexponential factors. However, for smaller values of the radius, the time complexity of the algorithm is determined by the number of points inside a sublattice of a smaller dimension.

3.4 The nearest plane algorithm.

The nearest plane algorithm was proposed in 1986 by Babai in [Bab86] as an algorithm for the approximate Closest Vector Problem. The quality of the approximation of the target point with a lattice vector by the nearest plane algorithm depends on how reduced an input basis is, i.e., it depends on the length of the Gram-Schmidt vectors that corresponds to the input basis. Originally, Babai proposed to use the nearest plane algorithm together with an LLL-reduced basis, which allows to solve $2^{\Theta(n)}$-CVP.

The nearest plane algorithm is based on the following property of the Gram-Schmidt orthogonalization: if we write a basis $\mathbf{B}$ in the coordinate system that corresponds to its normalized Gram-Schmidt basis $\mathbf{B}^*$, we obtain an upper triangular matrix with the norms of the corresponding Gram-Schmidt vectors on the diagonal. Then, for any target vector $\mathbf{t}$, we can find such an integer linear combination of vectors of $\mathbf{B}$ that the i-th coordinate of $\mathbf{t}$ in within $\pm\frac{\|\mathbf{b}_i^*\|}{2}$ from the i-th coordinate of the linear combination. The approach is summarized by Algorithm 3.4, the complexity of the nearest plane algorithm in presented by Lemma 3.3.

Algorithm 3.4: Nearest plane algorithm

input : a basis $\mathbf{B} \in \mathbb{R}^{m \times n}$ of a lattice Λ, Gram-Schmidt orthogonalization $\mathbf{B}^*$, $\mathbf{t} \in \mathbb{R}^m$

output: $\mathbf{v} \in \Lambda$ such that $\frac{\|\mathbf{b}_i^*\|}{2} \leqslant \frac{(\mathbf{v}-\mathbf{t})^t \mathbf{b}_i^*}{\|\mathbf{b}_i^*\|} \leqslant \frac{\|\mathbf{b}_i^*\|}{2}$ for all i.

```
nearestPlane(B, B*, t):
    t' ← t
    for i ∈ {n, ..., 1} do
        t' ← t − ⌊ (t^t b_i*) / ||b_i*||^2 ⌉ · b_i
    return t − t'
```

Lemma 3.3. Let $\mathbf{B}$ be a basis of a lattice $\Lambda \subset \mathbb{R}^m$ of rank n and let $\mathbf{B}^*$ be the Gram-Schmidt orthogonalization of $\mathbf{B}$. Then, Algorithm 3.4, given as input $\mathbf{B}$, $\mathbf{B}^*$, and any $\mathbf{t} \in \mathbb{R}^m$, finds a lattice vector $\mathbf{v} \in \Lambda$, such that the projection of $\mathbf{y} = \mathbf{t} - \mathbf{v}$ on the i-th Gram-Schmidt vector $\mathbf{b}_i$ is bounded by $\frac{1}{2}\|\mathbf{b}_i^*\|$ for all $i \in \{1, \ldots, n\}$, in polynomial in n time.

For the proof of Lemma 3.3, see [Bab86]. Essentially, for any target vector $\mathbf{t} \in \mathbb{R}^n$, the Nearest Plane algorithm finds a lattice vector $\mathbf{v}$, such that the difference $\mathbf{t} - \mathbf{v}$ lies inside the parallelepiped spanned by the vectors of the Gram-Schmidt orthogonalization $\mathbf{B}^*$:

$$(\mathbf{t} - \mathbf{v}) \in \mathscr{P}(\mathbf{B}^*) := \Big\{ \mathbf{B}^*\mathbf{x} \mid x_i \in \big[-\frac{1}{2}; \frac{1}{2} \big] \Big\}.$$

We use this property of the nearest plane algorithm's output further in the thesis in order to localize the target point in the algorithm for solving CVP(P) (see Section 6.4.1).

3.5 Sampling the discrete Gaussian distribution

In this section, we recall the algorithm that, given a lattice $\Lambda \subset \mathbb{R}^n$ and a point $\mathbf{c} \in \mathbb{R}^\times$ and not too small parameter s, samples lattice points from the distribution that looks like the n-dimensional Gaussian distribution with the expectation $\mathbf{c}$ and variance s^2.

First, let us formally define this distribution. For any $s > 0$ and $\mathbf{c} \in \mathbb{R}^n$, the Gaussian function $\rho_{s,\mathbf{c}} : \mathbb{R}^n \to \mathbb{R}$ is defined by:

$$\rho_{s,\mathbf{c}}(\mathbf{x}) := \exp\Big(-\frac{\pi \|\mathbf{x} - \mathbf{c}\|^2}{s^2} \Big),$$

for all $\mathbf{x} \in \mathbb{R}^n$. When the subscript $\mathbf{c}$ is omitted, we take $\mathbf{c}$ equal to $\mathbf{0}$. For any discrete set $A \subset \mathbb{R}^n$, we denote the sum $\sum_{\mathbf{x} \in A} \rho_{s,\mathbf{c}}(\mathbf{x})$ as $\rho_{s,\mathbf{c}}(A)$. Then, for any $s > 0$, $\mathbf{c} \in \mathbb{R}^n$, and any lattice $\Lambda \subset \mathbb{R}^n$, the discrete Gaussian distribution $D_{s,\mathbf{c},\Lambda}$ on Λ with the parameters s and $\mathbf{c}$ is defined as:

$$\forall \mathbf{x} \in \Lambda,\ \mathbb{P}_{D_{s,c,\Lambda}}\{\mathbf{x}\} = \frac{\rho_{s,\mathbf{c}}(\mathbf{x})}{\rho_{s,\mathbf{c}}(\Lambda)}.$$

In [MR07], Micciancio and Regev introduced a lattice parameter, connected to the discrete Gaussian distribution, called the smoothing parameter.

Definition 3.3. [MR07, Definition 3.1] For any lattice Λ and any $\varepsilon > 0$, the smoothing parameter $\eta_\varepsilon(\Lambda)$ is the smallest positive s such that $\rho_{1/s}(\Lambda^* \backslash \{\mathbf{0}\}) \leqslant \varepsilon$.

In [MR07], Micciancio and Regev also described various properties of the discrete Gaussian distribution. In this thesis, we are mostly interested in the following concentration result.

Lemma 3.4. [MR07, Lemma 4.4] For any n-dimensional lattice Λ, $\mathbf{c} \in \mathbb{R}^n$, and $\varepsilon \in (0; 1)$, if the parameter s of the discrete Gaussian distribution on Λ satisfies $s \geqslant \eta_\varepsilon(\Lambda)$, we have:

$$\mathbb{P}_{D_{s,\mathbf{c},\Lambda}}\{\|\mathbf{x} - \mathbf{c}\| > s\sqrt{n}\} \leqslant \frac{1+\varepsilon}{1-\varepsilon} \cdot 2^{-n}.$$

Thus, if we can sample the discrete Gaussian distribution with the parameter s centered at $\mathbf{0}$, we can get lattice vectors of bounded norm. The length of the sampled vectors depends only on the parameter s.

In [GPV08], Gentry, Peikert, and Vaikuntanathan showed that, for not too small values of the parameter s, the discrete Gaussian distribution can be sampled efficiently using a randomized version of the Nearest Plane algorithm. The randomization of the Nearest Plane algorithm was first proposed by Klein in [Kle00]. The Nearest Plane algorithm proceeds by deterministic rounding of the coefficients that corresponds to the choice of the closest hyperplane (see Line 4 of Algorithm 3.4). The randomized version at each iteration assigns the probabilities to the hyperplanes depending on their distance to the

considered point and then randomly chooses the hyperplane. See Section 4.2 in [GPV08] for the precise description of the randomized Nearest Plane algorithm.

In [GPV08], the authors analyze the distribution, produced by Klein's algorithm.

Theorem 3.3. [GPV08, Theorem 4.1] Klein's algorithm, given a basis $\mathbf{B}$ of an n-dimensional lattice Λ, a parameter $s = \|\mathbf{B}^*\| \cdot \omega(\sqrt{\log(n)})$, and a center $\mathbf{c} \in \mathbb{R}^n$, outputs a sample from a distribution that is statistically close to $D_{s,\mathbf{c},\Lambda}$.

Theorem 3.3 implies that, for any $\Lambda \subset \mathbb{R}^n$, lattice vectors of length $O(2^n) \cdot \lambda_1(\Lambda)$ can be sampled by Klein's algorithm that is given an LLL-reduced basis as an input. This allows to sample not too long lattice vectors in polynomial time. Klein's algorithm is widely used as a subroutine for initial sampling of lattice points in sieving algorithms (see, for example, [NV08, Section 4.2.1]).

3.6 Unbalanced lattice reduction

In this section, we recall the unbalanced reduction algorithm from [GINX16]. The algorithm takes as an input any basis $\mathbf{B}$ of an n-dimensional lattice Λ and $\sigma > 0$, and returns a basis $\mathbf{C}$ of Λ such that the Gram-Schmidt orthogonalization of $\mathbf{C}$ satisfies the following property: all Gram-Schmidt vectors, except the first one, are shorter than σ.

The algorithm is based on the following idea. Consider a two-dimensional lattice Λ, given by a basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2\}$. Let $\mathbf{B}^* = \{\mathbf{b}_1^* = \mathbf{b}_1, \mathbf{b}_2^*\}$ be the Gram-Schmidt orthogonalization of $\mathbf{B}$. Then, the volume of the lattice Λ is equal to the product of the norms of the two Gram-Schmidt vectors:

$$\mathrm{vol}(\Lambda) = \|\mathbf{b}_1\| \cdot \|\mathbf{b}_2^*\|.$$

We can make the first vector of the basis arbitrarily long by adding a suitable integer multiple of $\mathbf{b}_2$ to it: $\widehat{\mathbf{B}} := \{\mathbf{b}_1 + \alpha \cdot \mathbf{b}_2, \mathbf{b}_2\}$, where $\alpha \in \mathbb{Z}$, is also a basis of Λ. Since the norm of the first basis vector grows, the norm of the second Gram-Schmidt vector for the updated basis should decrease, because their product is an invariant as the volume of the lattice. Thus, we can make one of the two Gram-Schmidt vectors arbitrarily short at the cost of increasing the norm of the another vector.

In [GINX16], the authors extend the idea, described above, to any dimension. The unbalanced reduction, introduced in [GINX16], iteratively applies the two-dimensional procedure to two-dimensional projected sublattices of the input lattice. This way, all the Gram-Schmidt vectors, except one, can be made arbitrarily short.

In this section, we recall the unbalanced reduction algorithm and the analysis of its complexity. First, we describe the two-dimensional case, then we recall the algorithm for high dimensions. For simplicity, we consider a special case when the target bound σ is less than the norm of the Gram-Schmidt vectors that we aim to reduce. The proof for the general case is very similar (see [GINX16, Appendix C.2] for the proof with arbitrary $\sigma > 0$).

The unbalanced reduction in the dimension two for $\sigma < \|\mathbf{b}_2^*\|$ is summarized by Algorithm 3.5.

Lemma 3.5. Let $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2\}$ be a basis of a lattice $\Lambda \in \mathbb{R}^2$, let σ be a positive real number smaller than $\|\mathbf{b}_2^*\|$. Then, Algorithm 3.5, given as an input $\mathbf{B}$ and σ, returns a basis $\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2\}$ of Λ that satisfies:

1. $\|\mathbf{c}_1\| \leqslant \dfrac{\|\mathbf{b}_1\|\|\mathbf{b}_2\|}{\sigma} + \|\mathbf{b}_1\|$,
2. $\|\mathbf{c}_2^*\| \leqslant \sigma$,

in polynomial time of the size of the input.

Algorithm 3.5: Two-dimensional unbalanced reduction

input : a basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2\}$ of a lattice $\Lambda \subset \mathbb{R}^2$, $0 < \sigma < \|\mathbf{b}_2^*\|$.

output: a basis $\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2\}$ of Λ such that $\|\mathbf{c}_2^*\| \leqslant \sigma$, $\|\mathbf{c}_1\| \leqslant \frac{\|\mathbf{b}_1\|\|\mathbf{b}_2\|}{\sigma} + \|\mathbf{b}_1\|$.

1 `UnbalancedReductionDimTwo(`$\mathbf{B}$, σ`):`

2 **if** $(\mathbf{b}_1^t \mathbf{b}_2 < 0)$ **then**

3 $\mathbf{b}_2 \leftarrow -\mathbf{b}_2$

4 $\mathbf{B}^* \leftarrow$ GRAM-SCHMIDT$(\mathbf{B})$

5 $\mathbf{c}_2 \leftarrow \mathbf{b}_1$

6 $\alpha \leftarrow -\dfrac{\mathbf{b}_1^t \mathbf{b}_2}{\|\mathbf{b}_1\|^2} + \dfrac{\|\mathbf{b}_2^*\|}{\|\mathbf{b}_1\|} \cdot \sqrt{\dfrac{\|\mathbf{b}_1\|^2}{\sigma^2} - 1}$

7 $\mathbf{c}_1 \leftarrow \mathbf{b}_2 + \lceil \alpha \rceil \cdot \mathbf{b}_1$

8 $\mathbf{C} \leftarrow \{\mathbf{c}_1, \mathbf{c}_2\}$

9 **return** $\mathbf{C}$

Proof. The basis $\mathbf{B}$, written in the coordinate system that corresponds to its Gram-Schmidt orthogonalization $\mathbf{B}^*$, is given by the following matrix:

$$\mathbf{B} = \begin{pmatrix} \|\mathbf{b}_1^*\| & \frac{\mathbf{b}_1^t \mathbf{b}_2}{\|\mathbf{b}_1\|} \\ 0 & \|\mathbf{b}_2^*\| \end{pmatrix}. \tag{3.8}$$

The basis $\mathbf{C}$, returned by Algorithm 3.5, in the same coordinate system, is given by

$$\mathbf{C} = \begin{pmatrix} \left(\frac{\mathbf{b}_1^t \mathbf{b}_2}{\|\mathbf{b}_1\|} + \lceil \alpha \rceil \cdot \|\mathbf{b}_1\|\right) & \|\mathbf{b}_1\| \\ \|\mathbf{b}_2^*\| & 0 \end{pmatrix}, \tag{3.9}$$

where, by Line 4 of Algorithm 3.5,

$$\alpha = -\frac{\mathbf{b}_1^t \mathbf{b}_2}{\|\mathbf{b}_1\|^2} + \frac{\|\mathbf{b}_2^*\|}{\|\mathbf{b}_1\|} \cdot \sqrt{\frac{\|\mathbf{b}_1\|^2}{\sigma^2} - 1}. \tag{3.10}$$

The first vector of the Gram-Schmidt orthogonalization of $\mathbf{C}$ coincides with $\mathbf{c}_1$. Using (3.9), for the norm of $\|\mathbf{c}_1\|$, we get:

$$\|\mathbf{c}_1\| = \sqrt{\|\mathbf{b}_2^*\|^2 + \left(\frac{\mathbf{b}_1^t \mathbf{b}_2}{\|\mathbf{b}_1\|} + \lceil \alpha \rceil \cdot \|\mathbf{b}_1\|\right)^2}. \tag{3.11}$$

First, let us estimate $\|\mathbf{c}_1\|$ from the below. This can be done by replacing $\lceil \alpha \rceil$ by α in (3.11), because $\|\mathbf{b}_1\| \geqslant 0$ and $\mathbf{b}_1^t \mathbf{b}_2 / \|\mathbf{b}_1\| \geqslant 0$ (this is ensured by Lines 2-3 in Algorithm 3.5):

$$\|\mathbf{c}_1\| \geqslant \sqrt{\|\mathbf{b}_2^*\|^2 + \left(\frac{\mathbf{b}_1^t \mathbf{b}_2}{\|\mathbf{b}_1\|} + \alpha \cdot \|\mathbf{b}_1\|\right)^2} = \frac{\|\mathbf{b}_1\|\|\mathbf{b}_2^*\|}{\sigma}. \tag{3.12}$$

Then, using (3.12) together with the fact that $\mathrm{vol}(\Lambda) = \|\mathbf{c}_1\| \cdot \|\mathbf{c}_2^*\| = \|\mathbf{b}_1\| \cdot \|\mathbf{b}_2^*\|$, we see that $\|\mathbf{c}_2^*\| \leqslant \sigma$.

Thus, we ensured that the second condition of the lemma is satisfied. In order to check the first condition, we need an upper bound on $\|\mathbf{c}_1\|$. To do so, we replace $\lceil \alpha \rceil$ by $\alpha + 1$ in (3.11):

$$\|\mathbf{c}_1\| \leqslant \sqrt{\|\mathbf{b}_2^*\|^2 + \left(\frac{\mathbf{b}_1^t \mathbf{b}_2}{\|\mathbf{b}_1\|} + (\alpha + 1) \cdot \|\mathbf{b}_1\|\right)^2} = \frac{\|\mathbf{b}_1\|\|\mathbf{b}_2^*\|}{\sigma} + \|\mathbf{b}_1\|. \tag{3.13}$$

(3.13) is obtained using the inequality $\sqrt{a^2 + b^2} \leqslant |a| + |b|$.

The size of the coefficient α is polynomial in the size of the input (see Equation (3.10)). Thus, the complexity of the algorithm is polynomial in the size of the input. □

The unbalanced reduction in dimensions bigger than two works by iteratively applying Algorithm 3.5 to two-dimensional projected sublattices of the input lattice. The unbalanced reduction for a dimension $n > 2$ is summarized in Algorithm 3.6.

Algorithm 3.6: Unbalanced reduction

input : a basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ of a lattice $\Lambda \subset \mathbb{R}^n$, $0 < \sigma < \min_i \|\mathbf{b}_i^*\|$.
output: a basis $\mathbf{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_n\}$ of Λ such that for all $i \in \{2, \dots, n\}$, $\|\mathbf{c}_i^*\| \leqslant \sigma$, $\|\mathbf{c}_1\| \leqslant \sigma n \cdot \frac{\mathrm{vol}(\Lambda)}{\sigma^n}$.

1 `UnbalancedReduction(`$\mathbf{B}$, σ`):`
2 $\quad \mathbf{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_n\} \leftarrow \mathbf{B}$
3 $\quad$ **for** $i \in \{n-1, \dots, 1\}$ **do**
4 $\quad\quad (\mathbf{c}_i, \mathbf{c}_{i+1}) \leftarrow$ UnbalancedReductionDimTwo$(\{\mathbf{c}_i, \mathbf{c}_{i+1}\}, \sigma)$
5 $\quad$ **return** $\mathbf{C}$

The complexity and the quality of the output of the unbalanced reduction with a parameter $\sigma < \min_i \|\mathbf{b}_i^*\|$ is analyzed in [BGJ14]. In Theorem 3.4, we recall this result.

Theorem 3.4. [BGJ14, Theorem 3.2] Let $\mathbf{B}$ be an LLL-reduced basis of an n-dimensional integer lattice Λ. Let $\sigma \leqslant \min_i \|\mathbf{b}_i^*\|$ be a target bound. The unbalanced lattice reduction outputs in a polynomial time a basis $\mathbf{C}$ of the lattice Λ such that:

$$\|\mathbf{c}_i^*\| \leqslant \sigma \text{ for all } i \in \{2, \dots n\}, \tag{3.14}$$

$$\|\mathbf{c}_1\| \leqslant \sigma n \cdot \frac{\mathrm{vol}(\Lambda)}{\sigma^n}, \tag{3.15}$$

$$\frac{\sigma^{n+1-i}}{\mathrm{vol}(\pi_i(\Lambda))} \leqslant n + 1 - i \text{ for all } i \in \{2, \dots, n\}. \tag{3.16}$$

Proof. For all $i \in \{n-1, \dots, 1\}$, we use suffixes "old" and "new" to denote the values of variables at the beginning and at the end of the i-th iteration, respectively. That is, the suffix "old" denotes the values of variables just before applying the two-dimensional unbalanced reduction to $(\mathbf{c}_i, \mathbf{c}_{i+1})$ (see Line 4 of Algorithm 3.6) and the suffix "new" denotes the values just after applying Algorithm 3.5 to $(\mathbf{c}_i, \mathbf{c}_{i+1})$. Also, we denote as x_i the value $\|\mathbf{c}_i^{*\mathrm{new}}\|$ at the i-th iteration. Note that, at the iteration i, x_{i+1} coincides with $\|\mathbf{c}_{i+1}^{*\mathrm{old}}\|$. Also, x_n is the value of $\|\mathbf{c}_n^*\|$ at first iteration (when the index i is equal to $n-1$), therefore,

$$x_n = \|\mathbf{b}_n^*\| = \sigma a_n. \tag{3.17}$$

For all $i \in \{1, \dots, n\}$, let $a_i = \|\mathbf{b}_i^*\|/\sigma$. We show that, at each iteration $i \in \{n-1, \dots, 1\}$, the following invariant holds:

$$a_i x_{i+1} \leqslant x_i \leqslant a_i x_{i+1} + \sigma a_i. \tag{3.18}$$

Using suffixes "old" and "new", we can rewrite (3.18) as the following inequality for the norms of the Gram-Schmidt vectors $\mathbf{c}_i^*$ and $\mathbf{c}_{i+1}^*$ just before and just after the i-th iteration:

$$\frac{\|\mathbf{c}_i^{*\mathrm{old}}\|\|\mathbf{c}_{i+1}^{*\mathrm{old}}\|}{\sigma} \leqslant \|\mathbf{c}_i^{*\mathrm{new}}\| \leqslant \frac{\|\mathbf{c}_i^{*\mathrm{old}}\|\|\mathbf{c}_{i+1}^{*\mathrm{old}}\|}{\sigma} + \|\mathbf{c}_i^{*\mathrm{old}}\|. \tag{3.19}$$

As $(\mathbf{c}_i^{\mathrm{new}}, \mathbf{c}_{i+1}^{\mathrm{new}})$ are obtained by applying Algorithm 3.5 to $(\mathbf{c}_i^{\mathrm{old}}, \mathbf{c}_{i+1}^{\mathrm{old}})$, the inequalities given by (3.19) hold by Lemma 3.5. Also,by Lemma 3.5, for all $i \in \{2, \dots, n\}$, $\|\mathbf{c}_i^{*\mathrm{new}}\| \leqslant \sigma$, which is equivalent to (3.14).

Now, using (3.18) and (3.17), we obtain a bound on the norm of the first vector of the basis $\mathbf{C}$, returned by Algorithm 3.6:

$$\|\mathbf{c}_1\| = \|\mathbf{c}_1^{*\text{new}}\| = x_1 \leqslant \sigma \cdot \sum_{k=1}^{n} \prod_{i=1}^{k} a_i \leqslant n\sigma \prod_{i=1}^{n} \frac{\|\mathbf{b}_i^*\|}{\sigma} = n\sigma \frac{\text{vol}(\Lambda)}{\sigma^n}. \tag{3.20}$$

Similarly, for all $i \in \{1, \ldots, n\}$, we get:

$$x_i \leqslant (n+1-i)\sigma \cdot \frac{\text{vol}(\pi_i(\Lambda))}{\sigma^{n+1-i}}, \tag{3.21}$$

which is equivalent to (3.16). □

Theorem 3.4 implies that the unbalanced reduction allows to obtain quasi-orthonormal bases for the projected sublattices of the input lattice.

Corollary 3.1. Let $\mathbf{C}$ be a basis of a lattice Λ that satisfies the conditions from Theorem 3.4. Then, $\pi_2(\mathbf{C})$ is a quasi-orthogonal basis of the lattice $\pi_2(\mathbf{\Lambda})$.

Proof. The Rankin factor with index $j \leqslant n-1$ of basis $\pi_2(\mathbf{C})$ is given by

$$\gamma_{n-1,j} = \frac{\text{vol}(\pi_2(\Lambda)^{\frac{n-1-j}{n-1}})}{\text{vol}(\pi_{j+2}(\Lambda))}$$

In order to estimate the Rankin factors of $\pi_2(\mathbf{C})$, we need an upper and lower bounds for $\text{vol}(\pi_k(\Lambda))$ for $k \geqslant 2$. Using (3.14) we obtain the following upper bound: $\text{vol}(\pi_k(\Lambda)) \leqslant \sigma^{n-k+1}$. The lower bound can be deduced from (3.14) and (3.15):

$$\text{vol}(\pi_k(\Lambda)) = \prod_{i=k}^{n} \|\mathbf{c}_i^*\| = \frac{\text{vol}(\Lambda)}{\|\mathbf{c}_1\| \cdot \prod_{i=2}^{k-1} \|\mathbf{c}_i^*\|} \geqslant \frac{\sigma^{n-k+1}}{n}.$$

Then, we obtain $\gamma_{n-1,j}(\pi_2(\mathbf{C})) \leqslant n$ for all $j \leqslant n-1$. □

3.7 Covering d-dimensional surfaces

In this section, we consider the problems of the following kind. We are given an d-dimensional surface, such that we can define a uniform distribution on it. Then, we place N points on the surface uniformly at random. With each placed point we associate its vicinity on the surface of some particular shape, and we say that each point that belongs to that vicinity is covered. The goal is to estimate how many points we need in order to cover all the surface with a high probability, depending on the size and shape of the covered vicinity, associated with each point.

More precisely, we consider the three following problems: covering a sphere with random hemispheres, covering a sphere with spherical caps, and covering a hypercylinder with half-cylinders of smaller height. For covering a sphere with random hemispheres we consider the probability of covering the whole surface, for the other two problems, in order to simplify the computations, we consider an easier problem of covering the vast majority of the surface instead of the whole surface.

3.7.1 Covering a sphere with hemispheres

Consider covering a sphere with hemispheres. That is, we randomly drop points $x_1, \ldots, x_N$ on the surface of the sphere and we say that one dropped point $\mathbf{x}_i \in S^{d-1}$ covers all the sphere points that lie at the angular distance $\pi/2$ from it. The goal is to find how many hemispheres are enough to cover each point of the sphere at least once with high probability.

The event that the sphere is covered by N hemispheres, is the complement of the event that all N centers of the hemispheres have turned up on some hemisphere. In geometric probability, there is a result due to Wendel that gives a probability of that complement event.

Theorem 3.5. (Wendel's theorem [Wen62]) Let $\mathbf{x}_1, \ldots \mathbf{x}_N \in \mathbb{R}^d$ be N vectors that are independently drawn from the uniform distribution over the surface of the unit sphere S^{d-1}. Then, the probability that they all lie on some hemisphere is

$$p_{d,N} = \frac{1}{2^{N-1}} \cdot \sum_{k=0}^{d-1} \binom{N-1}{k}.$$

The following straightforward corollary from Wendell's theorem states that the linear in the dimension amount of points is enough to cover the sphere with probability close to one.

Corollary 3.2. Let $C = \{c_1, \ldots, c_{4d+1}\} \subset S^{d-1}$ be a set of $4d+1$ points drawn independently from the uniform distribution over the unit sphere S^{d-1}. Let $\mathrm{Hem}(\mathbf{c})$ be a hemisphere formed by all sphere points at the angular distance to $\mathbf{c}$ smaller than $\pi/2$.

Then, the probability that the sphere is covered by $\mathrm{Hem}(c_1), \ldots \mathrm{Hem}(c_{4d+1})$ is at least $1 - \sqrt{d} \cdot 2^{-7d/4}$.

Proof. The event that the sphere is covered, is the complement of the event that there is a hemisphere that contains all the points from C:

$$\begin{gathered} p_{\mathrm{cov}} := \Pr[\text{all points of } S^{d-1} \text{ are covered}] = \Pr[\forall \mathbf{x} \in S^{d-1} : \mathrm{Hem}(\mathbf{x}) \cap C \neq \varnothing] = \\ 1 - \Pr[\exists \mathbf{y} \in S^{d-1} : \mathrm{Hem}(\mathbf{y}) \cap C = \varnothing] = 1 - \Pr[\exists \mathbf{z} \in S^{d-1} : C \subset \mathrm{Hem}(\mathbf{z})]. \end{gathered} \tag{3.22}$$

Then, using Theorem 3.5, we obtain:

$$\begin{gathered} p_{\mathrm{cov}} = 1 - p_{d,4d+1} = \\ 1 - \frac{1}{2^{4d}} \cdot \sum_{k=0}^{d-1} \binom{4d-1}{k} \geqslant 1 - 2^{-4d} \cdot d \cdot \binom{4d}{d}. \end{gathered} \tag{3.23}$$

We use Stirling's approximation to estimate the binomial coefficient:

$$\binom{4d}{d} \sim \sqrt{\frac{2}{3\pi d}} \cdot \left(\frac{256}{27}\right)^d.$$

Then, for the probability of the sphere being covered, we obtain:

$$p_{\mathrm{cov}} \geqslant 1 - \sqrt{d} \cdot 2^{-1.754d}.$$

□

3.7.2 Covering a sphere with spherical caps

In this section, we consider the generalization of the previous problem: now, instead of covering a sphere with hemispheres, i.e., with spherical caps of angular radius $\frac{\pi}{2}$, we consider covering a sphere with spherical caps of an arbitrary radius.

The problem of covering a hypersphere with spherical caps often arises in the analysis of the complexity of lattice sieving algorithms. A typical sieving algorithm generates a list of long lattice vectors and looks for pairs of vectors, such that their difference is slightly shorter than the length of the vectors from the initial list. It is usually assumed that the vectors from the list behave like uniformly distributed on the sphere. Therefore, in order to ensure that the desired pairs of vectors can be found in the list, the size of the list should be big enough to cover the surface of the sphere with high probability.

The results on covering a sphere with spherical caps can be found in the literature on sieving algorithms. See, for example, [NV08, MV10], for covering a sphere with spherical caps of angular radius close to $\frac{\pi}{3}$, and see [BGJ14] for the volume of the intersection of the hyperballs depending on the distance between their centers. Here, for completeness, we provide a similar result on covering a sphere with random spherical caps of arbitrary angular radius $\alpha \in \left(0; \frac{\pi}{2}\right)$.

Let $C_d(\mathbf{c}, \alpha)$ denote a spherical cap centered at $\mathbf{c} \in S^{d-1}$ of angular radius α:

$$C_d(\mathbf{c}, \alpha) := \{\mathbf{x} \in S^{d-1} \mid \mathbf{x}^t\mathbf{c} \geqslant \cos(\alpha)\}.$$

In the previous section, we estimated how many random hemispheres are needed to cover the full surface of a sphere with high probability. Now, when considering a spherical cap of radius $\alpha \in (0; \pi/2)$, in order to avoid cumbersome computations, we consider an easier problem. Instead of estimating the probability of covering the full surface, we estimate the probability to cover the proportion of the surface that is close to one.

In order to do so, we need essentially two ingredients: an estimate of the area of the sphere surface covered by one spherical cap and Markov's inequality.

Theorem 3.6 (Markov's inequality)**.** Let X be a non-negative random variable and let $a > 0$. Then,

$$\mathbb{P}\{X > a\} \leqslant \frac{\mathbb{E}X}{a}.$$

In Lemma 3.6, we recall the area of an n-dimensional spherical cap of angular radius α.

Lemma 3.6. For any $\alpha \in (0; \frac{\pi}{2})$ and any $\mathbf{c} \in S^{d-1}$, we have

$$\frac{A(C_d(\mathbf{c}, \alpha))}{A(S^{d-1})} = \Theta\Big(\frac{1}{d} \cdot \sin(\alpha)^{d-1}\Big).$$

Proof. The surface area of the cap is given by

$$A(C_d(\mathbf{c}, \alpha)) = \frac{1}{2}A(S^{d-1})I_{\sin(\alpha)^2}\big(\frac{d-1}{2}, \frac{1}{2}\big),$$

where I denotes the incomplete beta function (see [Li11]). Then, the ratio between the surface of the spherical cap and the surface of the sphere is

$$\frac{A(C_d(\mathbf{c}, \alpha))}{A(S^{d-1})} = \frac{1}{2}\int_0^{\sin(\alpha)^2} \frac{t^{\frac{d-1}{2}}}{\sqrt{1-t}}dt.$$

The integral can be bounded from the below and the above:

$$\frac{\sin(\alpha)^{d-1}}{(d-1)} \leqslant \frac{1}{2} \int\limits_0^{\sin(\alpha)^2} \frac{t^{\frac{d-1}{2}}}{\sqrt{1-t}} dt \leqslant \frac{\sin(\alpha)^{d-1}}{(d-1)\cos(\alpha)}. \tag{3.24}$$

□

The following theorem states that $O(d^2)\sin(\alpha)^d$ random spherical caps of angular radius α are enough to cover all the surface of the n-dimensional sphere S^{n-1}, except an exponentially small part, with overwhelming probability.

Theorem 3.7. Let $\alpha \in (0; \pi/2)$. Let N be an integer number greater than $\dfrac{2\ln(2)d^2}{\sin(\alpha)^{d-1}}$. Let $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_N\}$ be a set of points, sampled independently from the uniform distribution on S^{d-1}. Denote as $S_{\text{cov}}(\mathbf{C}, \alpha)$ the surface of the sphere, covered by the spherical caps of angular radius α centered at the points from $\mathbf{C}$. Then,

$$\mathbb{P}\Big\{\frac{A(S_{\text{cov}}(\mathbf{C}, \alpha))}{A(S^{d-1})} \geqslant 1 - 2^{-d}\Big\} \geqslant 1 - 2^{-d}. \tag{3.25}$$

Proof. Let X be the proportion of the surface of the sphere S^{d-1} that is not covered by the points from $\mathbf{C}$, i.e.,

$$X = \frac{A(S^{d-1} \setminus S_{\text{cov}}(\mathbf{C}, \alpha))}{A(S^{d-1})}.$$

X is a random variable with the set of outcomes equal to $[0; 1]$. For any $\mathbf{x} \in S^{n-1}$, we define as $\xi_N(\mathbf{x})$ the indicator of the even that the point $\mathbf{x}$ is uncovered that is,

$$\xi_N(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \text{ is uncovered;} \\ 0, & \text{otherwise.} \end{cases}$$

The probability that the point $\mathbf{x}$ is uncovered, is equal to the probability that all the points from $\mathbf{C}$ are outside of the spherical cap $C_d(\mathbf{x}, \alpha)$ centered at $\mathbf{x}$. Then, using Lemma 3.6, for any $\mathbf{x} \in S^{d-1}$, we obtain

$$\mathbb{P}\{\xi_N(\mathbf{x}) = 1\} \leqslant \Big(1 - \frac{\sin(\alpha)^{n-1}}{n-1}\Big)^N.$$

We can rewrite X using ξ_N: $X = \int\limits_{S^{d-1}} \xi_N(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$. Therefore, the expectation of X is

$$\mathbb{E}\{X\} = \mathbb{E} \int\limits_{S^{d-1}} \xi_N(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \leqslant \int\limits_{S^{d-1}} \Big(1 - \frac{\sin(\alpha)^{d-1}}{d-1}\Big)^N p(\mathbf{x}) d\mathbf{x} = \Big(1 - \frac{\sin(\alpha)^{d-1}}{d-1}\Big)^N.$$

Then, for $N = \frac{2\ln(2)d(d-1)}{\sin(\alpha)^{d-1}}$, $\lim\limits_{n\to\infty} \frac{\mathbb{E}X}{2^{-2d}} = 1$. Using the obtained estimate on the expectation of X together with Markov's inequality, we get an upper bound for X:

$$\mathbb{P}\{X > 2^{-d}\} \leqslant \frac{\mathbb{E}X}{2^{-d}} = 2^{-d}.$$

□

In the proof of Theorem 3.7, we used Markov's inequality to estimate the probability to cover a sphere. A more precise bound can be obtained using the second moment of the distribution together with Chebyshev's inequality. However, computing the second moment involves estimation of the area of the intersection of two spherical caps, which is rather complicated (see [LK14] for the surface area of the intersection of two n-spherical caps).

3.7.3 Cover a hypercylinder with random half-cylinders of smaller height

In this thesis, we describe a sieving algorithm that generates a list of lattice vectors that lie inside a long and narrow hypercylinder and then, as other sieving algorithms, looks for a pairs of vectors that give a short difference. In order to analyze the complexity of such an algorithm, we need a result, similar to the ones described in two previous sections, but for a hypercylinder instead of a sphere. In this, section, we consider the problem of covering a hypercylinder with random half-cylinders of smaller height.

For any $\varepsilon \in (0;1)$ and for any $\mathbf{c} \in C^d_{\mathbf{u}}(h,R)$, we denote as $hC^d_{\mathbf{u},h,R}(\mathbf{c},\varepsilon)$ a half-cylinder of height εh, centered at the point $\mathbf{c}$, i.e., the following subset of the hypercylinder $C^d_{\mathbf{u}}(h,R)$:

$$hC^d_{\mathbf{u},h,R}(\mathbf{c},\varepsilon) = \Big\{\mathbf{x} \in C^d_{\mathbf{u}}(h,R) \,|\, \mathbf{x}^t\mathbf{u} \in \big(-\frac{\varepsilon h}{2}, \frac{\varepsilon h}{2}\big),\ \|\pi_{\mathbf{u}}(\mathbf{x}-\mathbf{c})\| \leqslant R\sqrt{2}.\Big\}$$

We can get a figure of the same shape and volume as $hC^d_{\mathbf{u},h,R}(\mathbf{c},\varepsilon)$, if we cut ε fraction of the hypercylinder across the axis, and then cut the obtained hypercylinder into two equal parts along the axis. We denote $hC^d_{\mathbf{u}}(\mathbf{c},\varepsilon) := hC^d_{\mathbf{u},h=1,R=1}(\varepsilon,\mathbf{c},C^d_{\mathbf{u}})$.

Lemma 3.7. Let $\mathbf{u} \in \mathbb{R}^d$ be any d-dimensional unit vector. Let $\varepsilon \in (0;1)$ and let $\mu > 0$. Let $N = \frac{4}{\varepsilon}\cdot\mu$. Let $\mathbf{C} = \{\mathbf{c}_1,\ldots,\mathbf{c}_N\} \subset C^d_{\mathbf{u}}$ be a set of points, sampled independently from the uniform distribution on the hypercylinder $C^d_{\mathbf{u}}$. Then, for any $\delta \in (0;1)$, the probability that at least $(1-\delta)$ fraction of the surface of $C^d_{\mathbf{u}}$ is covered by the half-cylinders of height ε, centered at the points from $\mathbf{C}$, is at least $1 - \frac{e^{-\mu}}{\delta}$.

Proof. The structure of the proof is similar to the proof of Theorem 3.7 about covering a sphere with spherical caps. Again, we denote the fraction of the surface, uncovered by N random points, as a random variable X. First, we compute the expectation of X and then we use Markov's inequality to estimate the probability that a certain amount of the surface is uncovered.

In order to compute the expectation of X, we need the probability that a fixed point $\mathbf{x} \in C^d_{\mathbf{u}}$ is uncovered. Denote at $\xi_N(\mathbf{x})$ the indicator of $\mathbf{x}$ being uncovered. The probability that $\xi_N(\mathbf{x}) = 1$, is equal to the probability that all N sampled points are outside the half-cylinder $hC(\mathbf{x},\varepsilon)$, centered at $\mathbf{x}$:

$$\mathbb{P}\{\mathbf{x} \text{ is uncovered}\} = \mathbb{P}\{\mathbf{C} \subset C^d_{\mathbf{u}} \setminus hC(\varepsilon,\mathbf{x})\}.$$

The fraction of the hypercylinder, covered by one half-cylinder of height ε, ranges from $\frac{\varepsilon}{4}$ for half-cylinders with a center on the boarder of the cylinder, to $\frac{\varepsilon}{2}$ for half-cylinders, centered closer to the middle of the hypercylinder. Then, for the expectation of X we obtain:

$$\mathbb{E}X = \int\limits_{C^d_{\mathbf{u}}} \xi_N(\mathbf{x})d\mathbf{x} \leqslant \int\limits_{C^d_{\mathbf{u}}} (\max_{\mathbf{x}} \xi_N(\mathbf{x}))d\mathbf{x} = \big(1-\frac{\varepsilon}{4}\big)^N.$$

Then, for $N \geqslant \frac{4\mu}{\varepsilon}$, when ε is close to zero, we get $\mathbb{E}X \leqslant e^{-\mu}$. Then, using Markov's inequality, we obtain the desired bound on the probability of $(1-\delta)$ fraction of the hypercylinder being uncovered. □

In the proof of Lemma 3.7, as in the proof of the analogous result from the previous section, we use only Markov's inequality. Again, a more precise bound on the probability can be obtained by considering the higher moments of the distribution. On the other side, since the proof uses only the bound on the expectation of the uncovered fraction of the surface, it can be adapted to non-uniform distributions of sampled points that behave like uniform. Informally, the desired property of the probability distribution that allows to obtain a result similar to the one described in Lemma 3.7, is that for half-cylinders of

not very small height, the probability to sample a point inside a half-cylinder is not much smaller than its area.

Definition 3.4. Let $\varepsilon \in (0;1)$, $h, R > 0$. Let D be a probability distribution with the support $C^d_{\mathbf{u}}(h,R)$. We say that the distribution D is (ε,α)-*quasi-uniform* on $C^d_{\mathbf{u}}(h,R)$ if it satisfies the following three conditions:

1. there exists a constant $\alpha \in (0;1)$, such that for any $t \in (-\frac{h}{2};\frac{h}{2})$, the probability that the projection of a point, sampled from D, on the cylinder axis, falls into an interval of length $h\varepsilon$, centered at t, is bounded from the above:
$$\mathbb{P}_D\left\{\mathbf{x}^t\mathbf{u} \in \left(t-\frac{h\varepsilon}{2}; t+\frac{h\varepsilon}{2}\right)\right\} \geqslant \varepsilon\cdot\alpha;$$
2. the rescaled projection of the point, sampled from the distribution D, on the subspace orthogonal to the cylinder axis is uniformly distributed on S^{n-2}, i.e., if $\mathbf{x}$ is sampled from D, then $\frac{1}{\|\pi_{\mathbf{u}}(\mathbf{x})\|}\cdot\pi_{\mathbf{u}}(\mathbf{x})$ is a random variable from the uniform distribution on S^{n-2};
3. the projections of the vector, sampled from D, on span($\mathbf{u}$) and on span($\mathbf{u}$)$^\perp$ are independent.

In Lemma 3.8, we generalize Lemma 3.7 for the defined species of distributions.

Lemma 3.8. Let $\mathbf{u} \in \mathbb{R}^d$ be a d-dimensional unit vector. Let $\varepsilon, \alpha \in (0;1)$, $\mu > 0$. Let D be a (ε,α)-quasi-uniform probability distribution on $C^d_{\mathbf{u}}$.

Let $N \geqslant \dfrac{2\mu}{\varepsilon\alpha}$. Let $\mathbf{C} = \{\mathbf{c}_1,\ldots,\mathbf{c}_N\} \subset C^d_{\mathbf{u}}$ be a set of N points, sampled independently from the distribution D. Then, for any $\delta \in (0;1)$, the probability that at least $(1-\delta)$ fraction of the surface of $C^d_{\mathbf{u}}$ is covered by the half-cylinders of height ε, centered at the points from $\mathbf{C}$, is at least
$$\mathbb{P}\{(1-\delta) \text{ fraction of cylinder is covered by } N \text{ random points}\} \geqslant 1-\frac{e^{-\mu}}{\delta}.$$

Proof. The structure of the proof is very close to the proof of Lemma 3.7 and we keep the same notation as in the proof Lemma 3.7. As before, X denotes a fraction of the surface of the cylinder, uncovered by N random points and $\xi_N(\mathbf{x})$ is an indicator of the event that a point $\mathbf{x} \in C^d_{\mathbf{u}}$ is uncovered, i.e., none of N sampled points is inside $hC(\varepsilon,\mathbf{x})$. By the definition of the distribution D, the probability that a randomly sampled point is not inside $hC^d_{\mathbf{u}}(\varepsilon,\mathbf{x})$ for any $\mathbf{x} \in C^d_{\mathbf{u}}$ is at least $\frac{\varepsilon\cdot\alpha}{2}$. Than, for any $\mathbf{x} \in C^d_{\mathbf{u}}$ we get the following bound:
$$\mathbb{P}\{\mathbf{x} \text{ is uncovered by } N \text{ points}\} = \mathbb{P}\{\xi_N(\mathbf{x}) = 1\} \leqslant (1-\varepsilon\cdot\alpha)^N.$$
When $\varepsilon \to 0$, we get $(1-\varepsilon\cdot\alpha)^N \to e^{-\mu}$.

Consider the expectation of X:
$$\mathbb{E}X = \int\limits_{C^d_{\mathbf{u}}} \xi_N(\mathbf{x})d\mathbf{x} \leqslant \int\limits_{C^d_{\mathbf{u}}} (\max_{\mathbf{x}\in U}\xi_N(\mathbf{x}))d\mathbf{x} \leqslant e^{-\mu}.$$

Then, using Markov's inequality for the random variable X, we get the bound on the probability that $(1-\delta)$ fraction of the cylinder is covered by N random points sampled from D. □

Chapter 4

Cylindrical sieving framework

In this chapter, we describe the framework of cylindrical sieving. First, we show that for integer lattices whose volume is a prime number, we can easily sample lattice points that lie inside a long but narrow hypercylinder. Then, we describe how these lattice points can be paired and sieved, in order to get a new set of lattice points that lie inside a much shorter but somewhat wider hypercylinder. By iteratively repeating the sieving step, we obtain a simple algorithm that solves SVP for lattices with a prime volume in time $\widetilde{O}(2^{n/2})$. Then, we show that this simple algorithm can be improved by controlling the growth of the width of the cylinder at each iteration. It allows to obtain time and memory trade-off and achieve the time complexity $\widetilde{O}(2^{0.3774n})$.

We also show that, after some preprocessing, the cylindrical sieving can be applied to any lattice. However, in the general case, the cost of the initial generation of lattice vectors inside a cylinder is higher than for a lattice with a prime volume. The cost of the sieving part remains the same as in the prime volume case. By choosing the parameters so that the overall cost of the algorithm is minimized, we obtain an algorithm that solves SVP for an arbitrary integer lattice with time complexity $\widetilde{O}(2^{0.3816n})$.

4.1 Generation of lattice vectors inside a cylinder

The first step of any sieving algorithm is the generation of an exponentially big list of lattice vectors with bounded length. In cylindrical sieving, we start with a list of lattice vectors that lie inside a long and narrow hypercylinder.

In this section, we show how the initial sampling for the cylindrical sieving can be performed. First, we show that in the case of a lattice with a prime volume, there is a straightforward way to do it thanks to the structure of the Hermite Normal form of the lattice. Then, we describe how to generate lattice vectors inside a cylinder for any input lattice using the structural lattice reduction from [GINX16].

4.1.1 Prime volume lattice case

Let n be a positive integer. Consider an integer lattice $\Lambda \in \mathbb{Z}^n$ whose volume is prime. We denote the volume of Λ as p and we assume that $\log(p) = \text{poly}(n)$. As p is huge, after computing the Hermite Normal Form of any basis of Λ most probably we get the following shape (see Section 2.3):

$$\mathbf{B} = \begin{pmatrix} p & a_1 & \dots & a_{n-1} \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & 1 \end{pmatrix}, \tag{4.1}$$

where $a_1, \dots, a_{n-1}$ are some numbers from $\mathbb{Z}_p$.

We can also obtain other shapes presented by (2.5). They can be seen as a concatenation of an identity matrix of dimension $0 < m < n$ and of the matrix of the shape described by (4.1) of dimension $m - n$. In that case, since the HNF contains unit vectors, by computing HNF we have already found the shortest vector of the matrix, as a unit vector is the shortest possible vector for any sublattice of $\mathbb{Z}^n$. If we are interested in finding other non-trivial short vectors of the lattice, we need to consider the sublattice of the same shape as given by (4.1), but of smaller dimension. Further in this work, we consider only the case when there is no unit vectors in the HNF of a lattice, as the hardest one.

Denote an $(n-1)$-dimensional vector with coordinates $a_1, \ldots, a_{n-1}$ as $\mathbf{a}$. For any $\mathbf{x} = (x_1, \ldots, x_{n-1})^t \in \mathbb{Z}^{n-1}$, a vector $\mathbf{y}$, given by

$$\mathbf{y} = \begin{pmatrix} \mathbf{a}^t\mathbf{x} \mod p \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}, \tag{4.2}$$

belongs to lattice Λ. Using this property, we can obtain a lattice vector that lies inside a hypercylinder $p \times S^{n-2}(R)$ by taking an $(n-1)$-dimensional integer vector of length smaller than R and then computing the corresponding lattice vector as described by (4.2).

In order to obtain a long list of lattice vectors inside a bounded hypercylinder, we enumerate all integer vectors that lie inside an $(n-1)$-dimensional ball centered at the origin and then compute the corresponding lattice vectors. If we take the radius of the ball proportional to $\sqrt{n-1}$, the number of lattice points that we obtain is exponential in n. Thanks to the structure of the integer lattice, this enumeration can be performed very efficiently. Algorithm 4.1 illustrates the approach.

Algorithm 4.1: Enumerate lattice points inside a hypercylinder.

input : description of a lattice Λ: a prime number p, $\mathbf{a} = (a_1, \ldots, a_{n-1})^t$;
radius of a hypercylinder $R = \sqrt{\alpha n}$.
output: $\nu(\alpha)^n$ points from $\Lambda \cap p \times S^{n-2}(R)$.

```
S ← ∅
L_{n-1} ← Z^{n-1} ∩ B_{n-1}(R)                    ▷ Algorithm 3.3
for x = (x_1, ..., x_{n-1})^t ∈ L_{n-1} do
    v_1 = a^t x  mod p
    v = (v_1, x_1, ..., x_n)^t
    S ← S ∪ {v}
return S
```

Lemma 4.1. Let p be a prime number and α be some positive constant. Let Λ be an integer n-dimensional lattice with $\mathrm{vol}(\Lambda) = p$. Then, Algorithm 4.1, given as input a description of the lattice Λ, enumerates all the points from the set $p \times S^{n-2}(\sqrt{\alpha(n-1)}) \cap \Lambda$ in time $O(n \cdot N)$, where $N = \nu(\alpha)^n$ is the number of the lattice points inside the hypercylinder.

Proof. The time complexity of the algorithm is the sum of the time required to create list L_{n-1} and of the time needed to compute the scalar product of $\mathbf{a}$ with each vector from the list. By Lemma 3.1, the time complexity of constructing L_{n-1} is $O(n|L_{n-1}|) = O(n \cdot N)$. The time complexity of computing the scalar product of $\mathbf{a}$ with each vector from the list is $O(n \cdot N)$, as $\mathbf{a}$ and vectors from L_{n-1} are $(n-1)$-dimensional. Then, the overall time complexity is also $O(n \cdot N)$. The memory complexity is the complexity of storing the list S, which is equal to $O(n \cdot |S|) = O(n \cdot |L_{n-1}|) = O(nN)$. As the radius of the hypercylinder is proportional to $\sqrt{n-1}$, by Theorem 3.2, number of enumerated points N is exponential in n and the constant in the exponent depends only on $\alpha = \frac{R^2}{n}$. □

4.1.2 Generate vectors inside a hypercylinder for any lattice

Let Λ be an n-dimensional lattice, let $h = 2^{O(n^2)} \cdot \mathrm{vol}(\Lambda)^{1/n}$, $R = 2^{-O(n)} \cdot \mathrm{vol}(\Lambda)^{1/n}$. In this section, we describe an algorithm that, given a basis of an n-dimensional lattice Λ, efficiently enumerates all lattice vectors that lie inside an n-dimensional hypercylinder of height h and radius R.

The algorithm is based on the following idea. Let $\mathbf{v} \in \Lambda$ be a lattice vector shorter than h. Assume that we can efficiently enumerate all the point in the lattice Λ such that their projection on $\mathrm{span}(\mathbf{v})^\perp$ lies inside inside the ball of radius R, i.e., we can efficiently compute the set

$$S' = \left\{ \mathbf{x} \in \Lambda \;\middle|\; \left\| \mathbf{x} - \frac{\mathbf{x}^t\mathbf{v}}{\mathbf{v}^t\mathbf{v}}\mathbf{v} \right\| \leqslant R \right\}.$$

Now, we get a set S' of lattice vectors that lie inside a hypercylinder of bounded radius R, but the height of the cylinder might be arbitrarily big, i.e., the projection of the vectors from $\mathbf{S}'$ on $\mathrm{span}(\mathbf{v})$ is unbounded.

For any lattice vector $\mathbf{x}$, we can compute the corresponding lattice vector $\mathbf{x}'$ such that the length of its projection on $\mathrm{span}(\mathbf{v})$ is bounded by $\|\mathbf{v}\|/2$ using size-reduction. The projection of $\mathbf{x}'$ on $\mathrm{span}(\mathbf{v})^\perp$ remains unchanged. Then, if we size-reduce the set S' with $\mathbf{v}$, we get the desired set S of all the lattice vectors inside the hypercylinder:

$$S = \left\{ \mathbf{x} - \left\lceil \frac{\mathbf{x}^t\mathbf{v}}{\mathbf{v}^t\mathbf{v}} \right\rfloor \cdot \mathbf{v} \;\middle|\; \mathbf{x} \in S' \right\}.$$

Therefore, in order to construct the algorithm, we need to find a vector $\mathbf{v} \in \Lambda$ of the norm smaller than h, such that we can easily enumerate all the vectors, shorter than R, in the lattice, obtained by the projection of Λ on $\mathrm{span}(\mathbf{v})^\perp$. This can be done by the unbalanced reduction (see Section 3.6). Recall that the unbalanced reduction produces a lattice basis $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_n\}$ such that the first vector $\mathbf{c}_1$ is long, but bounded, and, in addition, the last $(n-1)$ vectors form a short quasi-orthonormal basis for the projected sublattice $\pi_2(\mathbf{C})$.

The following corollary from Theorem 3.4 shows that, by applying the unbalanced reduction to an LLL-reduced basis of the lattice, we get the basis $\mathbf{C}$ with the desired parameters.

Corollary 4.1. Let $\delta \in (1/4; 1)$, let $q = \left(\delta - \frac{1}{4}\right)^{-1}$. Let $\mathbf{B}$ be a δ-LLL-reduced basis of an n-dimensional lattice Λ. Then, the unbalanced reduction algorithm (see Algorithm 3.6), given the basis $\mathbf{B}$ as an input, in polynomial time returns a basis $\mathbf{C}$ of the lattice Λ that satisfies the following properties:

$$\|\mathbf{c}_i^*\| \leqslant q^{-\frac{n-1}{2}} \cdot \mathrm{vol}(\Lambda)^{1/n} \text{ for all } i \in \{2, \ldots n\}, \tag{4.3}$$

$$\|\mathbf{c}_1\| \leqslant q^{\frac{(n-1)^2}{2}} n \cdot \mathrm{vol}(\Lambda)^{1/n}, \tag{4.4}$$

$$\text{basis } \pi_2(\mathbf{C}) \text{ of the projected sublattice } \pi_2(\Lambda) \text{ is quasi-orthonormal.} \tag{4.5}$$

Proof. For any $i, j \in \{1, \ldots, n\}$ denote the ratio $\frac{\mathbf{b}_i^t\mathbf{b}_j^*}{\mathbf{b}_j^*\mathbf{b}_j^*}$ as $\mu_{i,j}$. By the definition of δ-LLL-reduced basis, the Gram-Schmidt orthogonalization $\mathbf{B}^*$ of the basis $\mathbf{B}$ satisfies:

1. for all $i \in \{2, \ldots, n\}$, $j < i$, $|\mu_{i,j}| \leqslant \frac{1}{2}$;
2. for all $i \in \{1, \ldots, n-1\}$, $\delta\|\mathbf{b}_i^*\|^2 \leqslant \|\mu_{i+1,i}\mathbf{b}_i^* + \mathbf{b}_{i+1}^*\|^2$.

Combining this two inequalities together, we get:

$$\|\mathbf{b}_{i+1}^*\|^2 \geqslant (\delta - \frac{1}{4}) \cdot \|\mathbf{b}_i^*\|^2 = \frac{1}{q} \cdot \|\mathbf{b}_i\|^2, \tag{4.6}$$

for all $i \in \{1, \ldots, n-1\}$. Then, by iterating the inequality given by (4.6) i times, for the length of the first vector of the basis $\mathbf{B}$ we obtain:

$$\|\mathbf{b}_1\| = \|\mathbf{b}_1^*\| \leqslant q^{\frac{i-1}{2}} \cdot \|\mathbf{b}_i^*\| \leqslant q^{\frac{n-1}{2}} \min_i \|\mathbf{b}_i^*\|. \tag{4.7}$$

We assume that the norm of the first vector of the basis $\mathbf{B}$ is bigger than $\mathrm{vol}(\Lambda)^{1/n}$. We can safely assume that because, if this assumption doesn't hold, it means that the LLL algorithm returns a lattice vector shorter than the Minkowski's bound of Λ, which is very unlikely. With this assumption on the norm of $\mathbf{b}_1$, using (4.7), we get the following lower bound on the shortest Gram-Schmidt vector of $\mathbf{B}$:

$$\min_i \|\mathbf{b}_i^*\| \geqslant q^{-\frac{n-1}{2}} \mathrm{vol}(\Lambda)^{1/n}. \tag{4.8}$$

Then, since the shortest Gram-Schmidt vector of $\mathbf{B}$ is bounded from the below by (4.8), we can apply the unbalanced lattice reduction to $\mathbf{B}$ with the parameter sigma $\sigma = q^{-\frac{n-1}{2}} \cdot \mathrm{vol}(\Lambda)^{1/n}$.

By Theorem 3.4 and Corollary 3.1, the unbalanced reduction algorithm return a basis $\mathbf{C}$ of Λ that satisfies (4.3), (4.4), and (4.5). □

Figure 4.1 presents the shape of basis $\mathbf{C}$ after unbalanced reduction.

$$\begin{pmatrix} \|\mathbf{c}_1\| \leqslant \sigma \cdot 2^{O(n^2)} & \star & \ldots & \star \\ 0 & \|\mathbf{c}_2^*\| \leqslant \sigma & & \vdots \\ \vdots & & \ddots & \star \\ 0 & \ldots & 0 & \|\mathbf{c}_n^*\| \leqslant \sigma \end{pmatrix}$$

Figure 4.1 – Basis $\mathbf{C}$ of a lattice Λ after unbalanced reduction in the coordinate system that correspond to its normalized Gram-Schmidt basis $\mathbf{C}^*$. Here green rectangle denotes the quasi-orthonormal basis $\pi_2(\mathbf{C})$ of the projected sublattice $\pi_2(\Lambda)$.

Basis $\mathbf{C}$ can be used for enumerating lattice vectors inside a narrow hypercylinder of height $\|\mathbf{c}_1\|$ in the same way as the Hermite Normal Form is used in the case of a lattice with a prime volume p to enumerate vectors inside a narrow hypercylinder of height p. First, as $\mathbf{C}$ reveals a quasi-orthonormal basis of $\pi_2(\Lambda)$ (denoted by the green color in Figure 4.1), we are able to enumerate vectors of $\pi_2(\Lambda)$ inside a ball of a bounded radius efficiently using the Schnorr-Euchner's enumeration. Then, for each obtained vector we compute the corresponding vector in the lattice Λ. We can always ensure that the length of the projection of the resulting vectors on the direction of $\mathbf{c}_1$ is smaller than $\frac{\|\mathbf{c}_1\|}{2}$. The approach is summarized in Algorithm 4.2.

Algorithm 4.2, as well as Algorithm 4.1 for prime volume lattices, starts with enumerating short vectors in the $(n-1)$-dimensional sublattice of the input lattice. The difference between the prime volume case and the general case is that, in the prime volume case, the sublattice is always $\mathbb{Z}^{n-1}$, while in the general case the sublattice can be any $(n-1)$-dimensional lattice with a quasi-orthonormal basis.

Algorithm 4.2: Enumerate lattice points inside a hypercylinder.

input : basis $\mathbf{B}$ of an n-dimensional lattice Λ,
parameter of LLL reduction $q \in \left(\sqrt{4/3}; \sqrt{2}\right)$.

output: $e^{n/2}$ points from $\Lambda \cap h \times B_{n-1}(R)$, where $h = q^{\frac{(n-1)^2}{2}} \cdot n \operatorname{vol}(\Lambda)^{1/n}$,
$R = q^{-\frac{n-1}{2}} \cdot \sqrt{\frac{n-1}{2\pi}} \operatorname{vol}(\Lambda)^{1/n}$.

1 $\mathbf{B} \leftarrow \mathrm{LLL}(\mathbf{B}, q)$
2 $\sigma = q^{-\frac{n-1}{2}} \cdot \operatorname{vol}(\Lambda)^{1/n}$
3 $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_n\} \leftarrow$ UnbalancedReduction$(\mathbf{B}, \sigma)$ ▷ (see Theorem 3.4)
4 $\mathbf{C}_{n-1} = \{\mathbf{c}'_2, \ldots, \mathbf{c}'_n\} \leftarrow \pi_2(\mathbf{C})$
5 $S_{n-1} \leftarrow$ Enumerate$(\mathbf{C}_{n-1}, R)$ ▷ Schnorr-Euchner's enumeration
6 $S \leftarrow \varnothing$
7 **for** $\left(\mathbf{x} = \sum_{i=2}^{n} \alpha_i \cdot \mathbf{c}'_i \in S_{n-1}\right)$ **do**
8 $\quad \mathbf{v} \leftarrow \sum_{i=2}^{n} \alpha_i \cdot \mathbf{c}_i$
9 $\quad \mathbf{v} \leftarrow \mathbf{v} - \lfloor \frac{\mathbf{v}^t \mathbf{c}_1}{\mathbf{c}_1^t \mathbf{c}_1} \rceil \cdot \mathbf{c}_1$
10 $\quad S \leftarrow S \cup \{\mathbf{v}\}$
11 **return** S

Let Λ' be the $(n-1)$-dimensional sublattice of Λ with a quasi-orthonormal basis, recovered by Algorithm 4.2. In order to estimate the complexity of Algorithm 4.2, we need to count the number of points in the set $S^{n-1}(R) \cap \Lambda'$. To estimate the number of points in the set, we use the Gaussian Heuristic. That is the main difference from the analysis of Algorithm 4.1; in the prime volume case, in the similar situation, we used the estimates for an integer lattice from [MO90]. Now, as the lattice Λ' is not fixed, we assume that it behaves like a random lattice and use the Gaussian Heuristic.

Lemma 4.2. Let Λ be an n-dimensional integer lattice. Let $\beta > 1$. Let $\delta \in (1/4; 1)$ be the parameter of the LLL-reduction and let $q = (\delta - 1/4)^{-1}$. Then, Algorithm 4.2, given as an input a basis $\mathbf{B}$ of Λ and the parameters β and δ, outputs a list S of β^n lattice vectors such that there is a hypercylinder of height $h = q^{\frac{(n-1)^2}{2}} n \operatorname{vol}(\Lambda)^{1/n}$ and radius $R = \beta \cdot \sqrt{\frac{n-1}{2\pi e}} q^{-\frac{n-1}{2}} \operatorname{vol}(\Lambda)^{1/n}$, centered at the origin that contains all the vectors from S. The time complexity $\mathscr{T}$ of Algorithm 4.2 is given by:

$$\mathscr{T} = \begin{cases} \widetilde{O}(\beta^n), \text{ if } \beta > \sqrt{e}, \\ \widetilde{O}\Big(\exp\left(\frac{n\beta^2}{2e}\right)\Big), \text{ otherwise.} \end{cases} \tag{4.9}$$

Proof. First, let us consider the correctness of the algorithm, i.e., first, we check whether the vectors, returned by the algorithm, are inside the hypercylinder of the height h and the radius R. The algorithm starts by applying the unbalanced reduction to the input basis (see Line 3 of Algorithm 4.2). By Theorem 3.4 and Corollary 3.1, the unbalanced reduction returns the basis $\mathbf{C}$ of Λ such that

1. $\|\mathbf{c}_1\| \leqslant h/2$;
2. for all $i \in \{2, \ldots, n\}$, $\|\mathbf{c}_i^*\| \leqslant \sigma := R \cdot \dfrac{1}{\beta} \sqrt{\dfrac{2\pi}{n-1}}$;
3. the vectors $\mathbf{c}_2, \ldots, \mathbf{c}_n$ form a quasi-orthonormal basis of $\pi_2(\Lambda)$.

Denote the projected sublattice $\pi_2(\Lambda)$ as Λ' and denote the basis, formed by $\mathbf{c}_2, \ldots, \mathbf{c}_n$, as $\mathbf{C}'$. Using the quasi-orthonormal basis $\mathbf{C}'$, the algorithm enumerates point of the projected sublattice Λ' inside the ball of radius R, i.e., all the points from the set $S_{n-1} = \Lambda' \cap B^{n-1}(R)$ (see Line 5 of Algorithm 4.2). Then, the algorithm applies the size reduction with the vector $\mathbf{c}_1$ to all the points from S_{n-1} (see Lines 7-10 of Algorithm 4.2). Therefore, the projections of the vectors, returned by the algorithm, on $\mathrm{span}(\mathbf{c}_1)$ is within the range $(-h/2; h/2)$. The projections on $\mathrm{span}(\mathbf{c}_1)^{\perp}$ is bounded by R (ensured by Line 5 of the algorithm).

Now, consider the complexity of the algorithm. It is mostly defined by the complexity of the enumeration of the points in Λ', as the unbalanced reduction part takes only polynomial time. In order to estimate the complexity of the enumeration part, we estimate the number of lattice points inside the ball $B^{n-1}(R)$ using the Gaussian Heuristic. To do so, we need to estimate the volume of the projected sublattice. By Theorem 3.4, we get:

$$\frac{\sigma^{n-1}}{n} \leqslant \mathrm{vol}(\pi_2(\Lambda)) \leqslant \sigma^{n-1},$$

i.e., $\mathrm{vol}(\Lambda') \approx \sigma^{n-1}$. Then, using the Gaussian Heuristic, for the number of vectors of the lattice Λ' inside the ball of radius R, we obtain:

$$\frac{\mathrm{vol}(B_{n-1}(R))}{\mathrm{vol}(\pi_2(\Lambda))} \approx \beta^n.$$

Thus, by Lemma 3.2, the complexity of the enumeration (and of the whole algorithm, if we ignore polynomial factors) is given by (4.9). □

4.2 Sort-and-subtract algorithm for SVP

In this section, we describe a simple heuristic sieving algorithm for solving SVP that is based on the following idea. Assume that we are given a set of lattice vectors that lie inside a bounded hypercylinder of tiny radius R and of huge but bounded height h. As we have seen in the previous section, we can generate such a set of lattice vectors efficiently for any lattice. Consider the coordinate system, such that the first axis is parallel to the cylinder's axis. Sort the set of lattice vectors by the value of the first coordinate in that coordinate system. Since all the vectors lie inside the hypercylinder of height h, the first coordinates of the vectors from the set is bounded. Denote the size of the set as N. If we take the differences of vectors that are neighbours in the sorted set, we expect that for most of the differences the value of the first coordinate is smaller than $\frac{h}{N}$. Thus, if the size of the set is huge, we get a new set of vectors that lie inside the cylinder with a much lower height. At the same time, the other coordinates of the vectors may increase. If we assume that the vectors that consist of the last $(n-1)$ coordinates of the vectors from the initial set, behave like independent and uniformly distributed on the sphere, then we may expect that their differences are about $\sqrt{2}$ factor longer. Therefore, after taking the differences of the neighbours in the sorted set, the last coordinates grow by the constant factor, while the first coordinate decrease very much if the size of the initial list is big.

This way, by simply sorting vectors and subtracting neighbours, we get a new list of lattice vectors that are much shorter than before. By iteratively repeating this process, we can obtain lattice vectors of length close to the Minkowski's bound.

This section is organizes as follows. First, we describe show how the idea presented above can be made rigorous and turned into an algorithm that performs one sieving step. Then, we combine the initial sampling with the iterative cylindrical sieving in order to obtain an algorithm for finding short lattice vectors in $2^{n/2+o(n)}$ time and space.

4.2.1 One step of cylindrical sieving

Assume that we are given a list L of lattice vectors, such that all the vectors from the list lie inside a hypercylinder $h \times B_{n-1}(R)$, such that h is huge and R is tiny. The goal is to transform the list L into another list L' of lattice vectors, such that all vectors from the new list lie inside a hypercylinder with much smaller height at the cost of allowing its radius slightly grow:

$$h \times B_{n-1}(R) \to \frac{h}{N} \times B_{n-1}(R\sqrt{2}).$$

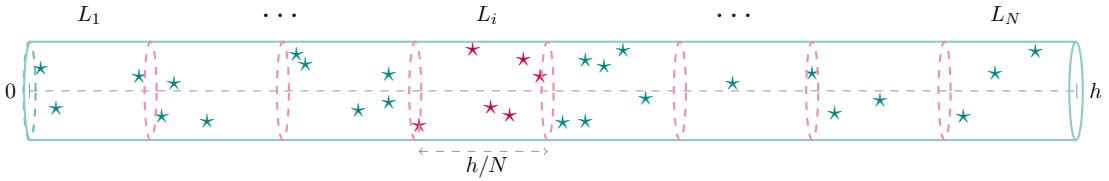
This is achieved by the sieving process, which chooses pairs of vectors from the input list in such a way that their differences satisfy the conditions above.

For convenience, we consider the coordinate system with the first axis parallel to the axis of the cylinder, so that vectors from the list have big first coordinate while their other coordinates are small. Further we use the following notation: for a vector $\mathbf{v} \in \mathbb{R}^n$, we denote its first coordinate as v_1 and the vector formed by the last $(n-1)$ coordinates of $\mathbf{v}$ as $\mathbf{u}_v$. For any vector $\mathbf{v}$ from the input list $\mathbf{L}$ we obtain:

$$\mathbf{v} = \begin{pmatrix} v_1 \in [0; h) \\ \mathbf{u}_v \in B_{n-1}(R). \end{pmatrix} \in \Lambda.$$

At first consider the reduction of the big first coordinate. All first coordinates of the vectors from list L lie inside the line segment of length h. We cut the line segment into N equal parts and correspondingly separate list L into N sublists $L_1, \ldots, L_N$: the i-th sublist contains vectors whose first coordinates are in the i-th part of the segment:

$$L = \bigcup_{i=1}^{N} L_i, \quad \forall i: \quad L_i = \{\mathbf{v} \in L \,|\, v_1 \in [i-1; i) \cdot h/N\}.$$



Consider the difference of two vectors from the same sublist L_i. For any two vectors $\mathbf{x}, \mathbf{y} \in L_i$, the difference is a lattice vector again and the absolute value of its first coordinate is bounded: $|x_1 - y_1| < h/N$.

The second condition that the difference $\mathbf{x} - \mathbf{y}$ should satisfy to be accepted to the new list L' is that the vector $\mathbf{u}_{x-y} = \mathbf{u}_x - \mathbf{u}_y$, formed by the last $(n-1)$ coordinates of the difference, should be short enough: $\|\mathbf{u}_{x-y}\|$ have to be smaller than $R\sqrt{2}$. It might not be true for any pair of vectors from L_i: in the worst case, when the angle between two vectors is almost π and both vectors have maximal length R, their difference is close to $2R$, so we have to sieve out differences with too long $\mathbf{u}_{x-y}$ part.

In order to achieve this goal, the algorithm performs a sieving step similar to the Nguyen-Vidick sieve inside each sublist L_i. For each L_i it selects a subset of centers $C_i \subset L_i$ and for each $\mathbf{v} \in L_i \setminus C_i$ it finds a suitable pair $\mathbf{c} \in C_i$ such that $\|\mathbf{u}_v - \mathbf{u}_c\| \leqslant R\sqrt{2}$.

The approach is summarized in Algorithm 4.3.

Each vector from the input list has two possibilities: either the algorithm finds an acceptable pair for it among centers and their difference passes to the new list L', or the vector becomes a center itself. Centers then are thrown away, so the size of the list decreases by the number of centers chosen by the algorithm:

$$|L'| = |L| - \sum_{i=1}^{N} |C_i|.$$

Algorithm 4.3: One sieving step

input : decrease rate N, list $L \subset \Lambda \cap h \times B_{n-1}(R)$, $|L| > 4nN$.
output: list $L' \subset \Lambda \cap \frac{h}{N} \times B_{n-1}(\sqrt{2}R)$, $|L'| > |L| - 4nN$.

1 $L' \leftarrow \varnothing$
2 $L_1 \leftarrow \varnothing, \ldots, L_N \leftarrow \varnothing$
3 $C_1 \leftarrow \varnothing, \ldots, C_N \leftarrow \varnothing$
4 **for** $\mathbf{v} \in L$ **do**
5 $\quad j \leftarrow \lceil v_1 \cdot N/h \rceil$
6 $\quad L_j \leftarrow L_j \cup \{\mathbf{v}\}$
7 **for** $(i = 1 \ldots N)$ **do**
8 $\quad$ **for** $(\mathbf{v} \in L_i)$ **do**
9 $\quad\quad$ **if** $(\exists \mathbf{c} \in C$ such that $\|\mathbf{v} - \mathbf{c}\| \leqslant R\sqrt{2})$ **then**
10 $\quad\quad\quad L' \leftarrow L' \cup \{\mathbf{v} - \mathbf{c}\}$
11 $\quad\quad C_i \leftarrow C_i \cup \{\mathbf{v}\}$
12 **return** L'

The essential part in analyzing the complexity of the algorithm is estimation of the required amount of centers. For that we make the following heuristic assumption:

Assumption 4.1. We assume that at any stage of Algorithm 4.3 vectors $\{\mathbf{u}_v\}_{\mathbf{v} \in L}$, after being normalized, behave as they are mutually independent and uniformly distributed on the unit sphere S^{n-2}.

Under this assumption, number of the points lost in different sublists are independent random variables, so it is enough to consider the loss of points inside one fixed sublist L_i, i.e., the number of centers needed for one sublist. Inside one sublist L_i, a vector $\mathbf{x}$ becomes a center if there is no existing center such that the vector formed by its last $(n-1)$ coordinates is at the angular distance less than or equal to $\frac{\pi}{2}$ from $\mathbf{u}_\mathbf{x}$. Once the centers from C_i have fully covered the sphere, all new incoming vectors in L_i will get the corresponding center from C_i and we won't loose vectors anymore. Therefore, estimating the number of centers lost inside one sublist turns into a problem of covering a sphere by random hemispheres with big enough probability.

The following lemma describes the complexity of Algorithm 4.3.

Lemma 4.3. Let $\Lambda \subset \mathbb{Z}^n$ be an n-dimensional lattice. Let $h, R > 0$. Let $L \subset \Lambda$ be a list of lattice vectors of size at least $4nN$, such that all vectors from L satisfy the following conditions:

$$\forall\ \mathbf{v} \in L, \quad \mathbf{v} = \begin{pmatrix} v_1 \in \mathbb{Z} \\ \mathbf{u}_v \in \mathbb{Z}^{n-1} \end{pmatrix}, \quad 0 < v_1 \leqslant h, \quad \|\mathbf{u}_v\| \leqslant R,$$

Then, if Assumption 4.1 holds, Algorithm 4.3, given as input the list L, outputs, with the probability greater than $1 - N \cdot 2^{-7n/4+o(n)}$, a list $L' \subset \Lambda$ of size at least $|L| - 4nN$, such that all vectors from L' satisfy the conditions:

$$\forall\ \mathbf{w} \in L', \quad \mathbf{w} = \begin{pmatrix} w_1 \in \mathbb{Z} \\ \mathbf{u}_w \in \mathbb{Z}^{n-1} \end{pmatrix}, \quad 0 < w_1 \leqslant \frac{h}{N}, \quad \|\mathbf{u}_w\| \leqslant R \cdot \sqrt{2}, \tag{4.10}$$

in time $O(n^2|L|)$ using $O(|L|\log(h))$ memory.

Proof. Algorithm 4.3 outputs lattice vectors, because the only operation it performs on the input vectors is subtraction. The lines $4-6$ and 9 of the algorithm ensure that the output list L' satisfy the requirements given by (4.10).

So, it remains to estimate how much shorter L' is compared to L and evaluate the complexity of the algorithm.

First, estimate the size of L'. There are two possible sources of the loss of points: collisions (when algorithm creates several equal differences) and centers. Under Assumption 4.1, the number of points lost due to collisions is negligible, therefore, it is enough to estimate the number of centers $\sum_{i=1}^{N} |C_i|$.

Consider one sublist L_i. We show that the number of points lost while processing one sublist is bounded with high probability. Denote as $\mathrm{cov_n}(\varepsilon)$ the number such that $\mathrm{cov_n}(\varepsilon)$ hemispheres, centered at points drawn from the uniform distribution on S^{n-2}, cover the sphere with probability $1-\varepsilon$.

Then, there are two cases: when $|L_i| > \mathrm{cov_n}(\varepsilon)$ and when $|L_i| \leqslant \mathrm{cov_n}(\varepsilon)$. In the first case, with probability $1-\varepsilon$, the number of centers $|C_i|$ won't exceed $\mathrm{cov_n}(\varepsilon)$. In the second case, even if we loose all the points from $|L_i|$ we do not loose more than $\mathrm{cov_n}(\varepsilon)$ points.

Under Assumption 4.1, the number of points lost in different sublists are independent, therefore, with probability $p_\varepsilon \geqslant (1-\varepsilon)^N$ number of all lost points satisfies

$$\sum_{i=1}^{N} |C_i| \leqslant N \cdot \mathrm{cov_n}(\varepsilon).$$

By Corollary 3.2, for $\varepsilon = 2^{-7n/4+o(n)}$, the required number of points is $\mathrm{cov_n}(\varepsilon) = 4n$. Then, with probability

$$p \geqslant (1-\varepsilon)^N \geqslant 1 - N \cdot \varepsilon = 1 - N \cdot 2^{-7n/4+o(n)},$$

the number of lost points is less than $4nN$.

Finally, estimate the complexity of the algorithm. The memory complexity is the memory required to store list L. List L contains n-dimensional vectors such that their first coordinate is less then h and all the other coordinates are smaller than R. If we assume that $h >> R \cdot (n-1)$, then the space needed to store one vector is $O(\log(h))$. Then, the memory complexity is $O(|L| \log(h))$.

For the time complexity, there are two parts: separating list into N sublist (lines $4-6$) and performing sieving on every sublist (lines $8-16$).

The index of the corresponding subset for each $\mathbf{v} \in L$ can be computed in constant number of operations and,therefore, the complexity of that part is linear in $|L|$.

In order to estimate the complexity of the sieving part consider one sublist L_i. For each $\mathbf{v} \in L_i$, the algorithm searches for the appropriate pair in C_i. If we assume that the subtraction of two n-dimensional vectors and computation of the norm of an n-dimensional vector can be performed in $O(n)$ operations, then the time spent on one sublist is $O(n|L_i| \cdot |C_i|)$. The overall number of operations is:

$$\sum_{i=1}^{N} |L_i| \cdot |C_i| \leqslant \mathrm{cov_n}(\varepsilon) \cdot \sum_{i=1}^{N} |L_i| = 4n \cdot |L|.$$

Then, the time complexity of the algorithm is $O(n^2|L|)$. □

4.2.2 Complexity of the sort-and-subtract algorithm for solving SVP

In this section, we combine the generation of lattice vectors inside a narrow cylinder with iteratively repeated cylindrical sieving in order to obtain an heuristic algorithm for solving SVP. As the initial generation of lattice vectors for cylindrical sieving differs from the general case for lattices with a prime volume, we start with analysing the case of lattices with a prime volume separately. Then, we show that, with slight modifications, the complexity analysis for the prime volume case can be adapted to the general case.

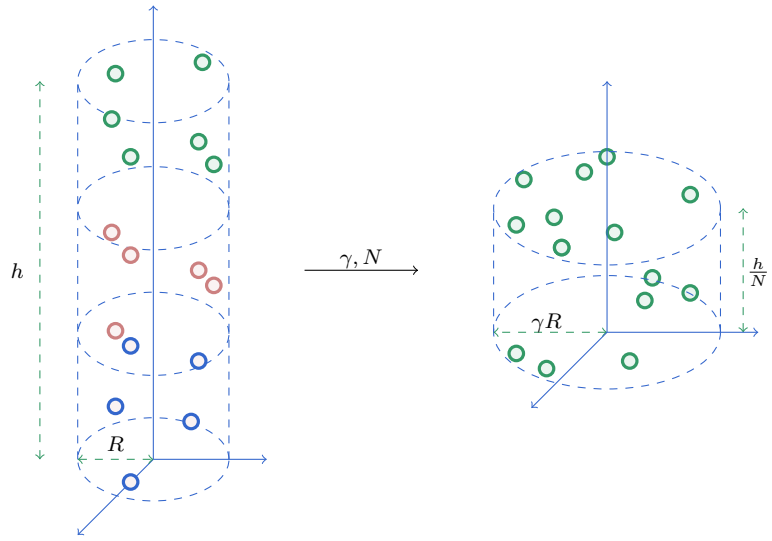


Figure 4.2 – One iteration of the cylindrical sieving. h denotes the upper bound on the first coordinate of the input vectors, R denotes the upper bound on the last $(n-1)$ coordinates. γ and N are the parameters of the cylindrical sieving algorithm. In Algorithm 4.3, $\gamma = \sqrt{2}$, $N = 2^{n/2}$.

Prime volume case. First, we describe the algorithm for the case of an n-dimensional lattice with a prime volume. The algorithm starts with generating exponentially many lattice vectors inside a narrow cylinder. For a lattice with a prime volume that can be done just by enumerating integer vectors inside a ball of a small radius (see Algorithm 4.1). Then, the algorithm iteratively applies the cylindrical sieving step to the list of lattice vectors until the short enough vector is found. The approach is summarized in Algorithm 4.4.

Algorithm 4.4: Sort-and-subtract algorithm

input : a basis $\mathbf{B}$ of a lattice Λ with a prime volume p.
output: $\mathbf{v} \in \Lambda$ such that $0 < \|\mathbf{v}\| \leqslant 0.41\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}$.

/* initial enumeration part */
$(p, \mathbf{a}) \leftarrow \mathrm{HNF}(\mathbf{B})$
$L_0 \leftarrow \textsc{enumerateCylinder}(p, \mathbf{a}, R_0 = \sqrt{0.084(n-1)})$ ▷ Algorithm 4.1
/* sieving part */
$k \leftarrow \left\lceil \frac{2}{n} \cdot \log(p) \right\rceil$
for $(i \in \{1, \ldots, k\})$ **do**
  $L_i \leftarrow \textsc{Sieve}(L_{i-1}, N)$ ▷ Algorithm 4.3
  if $(\exists \mathbf{v} \in L$ such that $0 < \|\mathbf{v}\| \leqslant 0.41\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n})$ **then**
    return $\mathbf{v}$

We analyze the complexity of Algorithm 4.4 in Theorem 4.1. It states that Algorithm 4.4 recovers a lattice vector of length close to the Minkowski's bound, using $\widetilde{O}(2^{n/2})$ time and memory.

Theorem 4.1. Let n be a positive integer number. Let $p = \mathrm{poly}(n)$ be prime and let $\lim_{n\to\infty} \frac{p}{n} = \infty$. Let Λ be an n-dimensional lattice with volume p. Let Assumption 4.1 hold. Then, Algorithm 4.4, given as input a basis of Λ, with probability at least $p = 1 - 2^{-5n/4+o(n)}$, outputs a lattice vector of length shorter than $0.41\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}$. The time and memory complexity of the algorithm are equal to $O(2^{n/2} \cdot \log(\mathrm{vol}(\Lambda))^2)$.

Proof. For all $i \in \{1, \ldots, k\}$, denote the parameters of the cylinder that contains the list

L_i as h_i and R_i: $L_i \subset h_i \times B_{n-1}(R_i)$. Then, by Lemma 4.3, the size of the list and the geometry of the vectors contained in the list change with one iteration in the following way:

$$|L_{i+1}| \geqslant |L_i| - 4n \cdot N, \quad R_{i+1} \leqslant R_i \cdot \sqrt{2}, \quad h_{i+1} \leqslant \frac{h_i}{N}. \tag{4.11}$$

Then, after k iterations:

$$|L_k| \geqslant |L_0| - 4nkN, \quad R_k \leqslant R_0 \cdot 2^{k/2}, \quad h_k \leqslant \frac{h_0}{N^k} = \frac{\det(\Lambda)}{N^k}. \tag{4.12}$$

The goal is to obtain a vector of length close to Minkowski's bound that is, we want to find a vector of length $\alpha \cdot \sqrt{\gamma_n}\,\mathrm{vol}(\Lambda)^{1/n}$ for some constant $\alpha > 1$. In order to obtain a lattice vector of the desired length, we need choose the parameters of the algorithm k and N, such that they satisfy the following two conditions:

$$R_k = R_0 \cdot 2^{k/2} \leqslant \sqrt{\frac{n-1}{n}} \cdot \alpha\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}, \tag{4.13}$$

$$h_k = \frac{\mathrm{vol}(\Lambda)}{N^k} \leqslant \alpha\,\mathrm{vol}(\Lambda)^{1/n}. \tag{4.14}$$

The first condition gives the upper bound for number of iterations k:

$$k \leqslant \frac{2}{n} \cdot \log(\mathrm{vol}(\Lambda)) + 2\log\left(\sqrt{n-1} \cdot \frac{\alpha}{R_0}\right). \tag{4.15}$$

If α and R_0 satisfy inequality the following inequality:

$$\frac{\alpha}{R_0} \geqslant \sqrt{\frac{2}{n-1}}, \tag{4.16}$$

then $2\log\left(\sqrt{n-1} \cdot \frac{\alpha}{R_0}\right) \geqslant 1$ and the number of iterations $k = \lceil \frac{2}{n}\log(\mathrm{vol}(\Lambda))\rceil$ satisfies the condition given by (4.15).

Now, using the chosen value of the number of iterations, we can choose the decrease rate N that satisfies the condition given by (4.14):

$$\log(N) \geqslant \frac{n}{2} - \frac{n\log(\alpha)}{2\log(\mathrm{vol}(\Lambda))}. \tag{4.17}$$

Since we assume that $\lim\limits_{n\to\infty} \frac{\log(\mathrm{vol}(\Lambda))}{n} = \infty$, the decrease rate $N = 2^{n/2}$ satisfies the condition given by (4.14).

Using the obtained values of the parameters k and N, we can evaluate the complexity of the algorithm. The memory complexity is the memory required to store the initial list L_0. After k iterations, the list L_k should be not empty, i.e., $|L_k| = |L_0| - 4nkN$ should be bigger than zero. Therefore, we require the size of the initial list to be equal to $(1+\mu) \cdot 4nkN$, where μ is some positive constant. Then, the initial list should contain $O(nkN)$ n-dimensional vectors. The vectors the first coordinate of the vectors from the initial list is smaller than $\mathrm{vol}(\Lambda)$ and the last $(n-1)$ coordinates are bounded by the constant R_0. Therefore, one vector from the initial list requires $O(\log(\mathrm{vol}(\Lambda)))$ of memory. Hence, the memory complexity of the algorithm is given by

$$\mathcal{S} = O(2^{n/2} \cdot \log(\mathrm{vol}(\Lambda))^2). \tag{4.18}$$

Now, consider the time complexity. By Lemma 4.3, i-th iteration of the algorithm requires $O(n^2 \cdot |L_{i-1}|)$ operations. Therefore, in order to estimate the time required for k

iterations, we need to compute the following sum:

$$\sum_{i=0}^{k-1} |L_i| = \sum_{i=0}^{k-1} (|L_0| - 4nN \cdot i) = k \cdot |L_0| - 4nN \cdot \sum_{i=0}^{k-1} i = k \cdot (|L_0| - 2nN(k-1)) = O(n \cdot k^2 N) \tag{4.19}$$

Then, the number of polynomial-time operations performed by the algorithm is given by:

$$\mathscr{T} = O\Big(2^{n/2} \cdot \log(\mathrm{vol}(\Lambda))^2\Big). \tag{4.20}$$

The probability of success of the algorithm is the product of probabilities of the successes of each iteration. Then, by Lemma 4.3,

$$p = (1 - N \cdot 2^{-7n/4})^k = 1 - 2^{-5n/4} \cdot O\Big(\frac{\mathrm{vol}(\Lambda)}{n}\Big).$$

The required size of the initial list for the algorithm is $2^{n/2+o(n)}$. The required size of the list defines the radius of the initial sampling R_0. Using Theorem 3.2, we obtain $R_0 \geqslant \sqrt{0.084(n-1)}$. Then, using (4.16), we get $\alpha \geqslant 0.41$. □

Sort-and-subtract algorithm for any lattice. In the general case, the sort-and-subtract algorithm remains almost the same as in the case of a lattice with a prime volume. Algorithm 4.5 represents the approach for any integer lattice. The sieving parts of both algorithms are completely similar. The main difference between Algorithm 4.4 and Algorithm 4.5 is the generation of the initial list of lattice vectors. In the case of a lattice with a non-prime volume, it takes more effort.

Algorithm 4.5: Sort-and-subtract algorithm

input : basis $\mathbf{B}$ of a lattice Λ

output: $\mathbf{v} \in \Lambda$ such that $0 < \|\mathbf{v}\| \leqslant \sqrt{\frac{2n}{\pi e}}\,\mathrm{vol}(\Lambda)^{1/n}$

```
/* initial enumeration part */
```

$L_0 \leftarrow$ EnumerateCylinder$(\mathbf{B}, q, \sqrt{2})$ ▷ Algorithm 4.2

```
/* sieving part */
```

$k \leftarrow \lceil (n-1)\log(q) \rceil$

for $(i \in \{1, \ldots, k\})$ **do**

  $L_i \leftarrow$ Sieve(L_{i-1}, N) ▷ Algorithm 4.3

  if $(\exists \mathbf{v} \in L$ such that $0 < \|\mathbf{v}\| \leqslant \sqrt{\frac{2n}{\pi e}}\,\mathrm{vol}(\Lambda)^{1/n})$ **then**

    return $\mathbf{v}$

The quality of the output and the complexity of Algorithm 4.5 are summarized in Theorem 4.2.

Theorem 4.2. Let Λ be an n-dimensional integer lattice. Let Assumption 4.1 hold. Then, Algorithm 4.5, given as input a basis of the lattice Λ, with probability at least $1 - 2^{-5n/4+o(n)}$, returns a non-zero lattice vector of length less then or equal to $\sqrt{\frac{2n}{\pi e}}\,\mathrm{vol}(\Lambda)^{1/n}$. The memory complexity of Algorithm 4.5 is $\widetilde{O}(2^{n/2})$, the time complexity is $\widetilde{O}(2^{0.531n})$. The time complexity of the cylindrical sieving part is equal to the time required to perform $O(n^4 \cdot 2^{n/2})$ polynomial-time operations.

Proof. The proof of Theorem 4.2 is very similar to the proof of Theorem 4.1. As before, after the enumeration part of the algorithm, we get a list L_0 of lattice vectors that lie

inside a bounded hypercylinder: $L_0 = \Lambda \cap h_0 \times B_{n-1}(R_0)$. But, in the general case, the parameters of the cylinder are different. By Lemma 4.2,

$$h_0 \leqslant q^{\frac{(n-1)^2}{2}} n \operatorname{vol}(\Lambda)^{1/n}, \quad R_0 \leqslant \beta \sqrt{\frac{n-1}{2\pi e}} q^{-\frac{n-1}{2}} \operatorname{vol}(\Lambda)^{1/n}, \tag{4.21}$$

where q is the parameter of the LLL reduction used by Algorithm 4.2 and β is some constant bigger than one that we choose later in the proof. The size of the list and the parameters of the hypercylinder change during the iterations of the algorithm in the same way as before (see (4.11)). The goal is to obtain a vector of length $\alpha\sqrt{n}\operatorname{vol}(\Lambda)^{1/n}$ after k iterations, with the smallest possible factor α. In order to find such a vector, we choose the number of iterations k and the decrease rate N such that they satisfy the conditions given by (4.13) and (4.14). Then, proceeding similarly as in the proof of Theorem 4.1, we get the following values for the number of iterations and on the decrease rate:

$$k = \lceil (n-1)\log(q) \rceil, \quad N = 2^{n/2}. \tag{4.22}$$

The choice of k also restricts the possible values of α; α should be bigger than $\frac{\beta}{\sqrt{\pi e}}$. Now, using the obtained values of k and N, we can estimate the complexity of the sieving part of the algorithm. The computations are similar to the proof of Theorem 4.1. The memory complexity of the algorithm is the memory, required to store a list of $O(n^2 \cdot 2^{n/2})$ n-dimensional vectors. The time complexity is the time required to perform $O(n^4 \cdot 2^{n/2})$ polynomial-time operations.

In order to estimate the complexity of the whole algorithm, it remains only to estimate the complexity of the enumeration part. For the sieving process to work, we need the initial list of lattice vectors of size $2^{n/2+o(n)}$. By Lemma 4.2, we need to take $\beta = \sqrt{2}$ to achieve that size of the list. Then, the complexity of the enumeration part is $\widetilde{O}(e^{\frac{n\beta^2}{2e}}) \approx \widetilde{O}(2^{0.531n})$. The length of the outputted vector divided by the Minkowski's bound is $\alpha \leqslant \sqrt{\frac{2}{\pi e}}$.

□

Table 4.1 summarizes the complexity of the sort-and-subtract algorithm. The memory complexity and the time complexity of the sieving part of the algorithm is $\widetilde{O}(2^{n/2})$ independently of the input lattice. However, the initial enumeration part is more costly in the general case than for lattices with a prime volume.

	prime volume	any lattice
memory	$2^{n/2}$	$2^{n/2}$
sieving time	$2^{n/2}$	$2^{n/2}$
initial enumeration time	$2^{n/2}$	$2^{0.513n}$
quality of output	0.41	0.484

Table 4.1 – The complexity of the sort-and-subtract algorithm for prime volume lattices (see Algorithm 4.4) and in the general case (see Algorithm 4.5). All complexities in the table are given without subexponential factors. The row "quality of the output" represents the lengths of the vectors, returned by the algorithms, divided by the Minkowski's bound of the input lattice.

4.3 Adding spherical sieving.

The cylindrical sieving proceeds by cutting the cylinder into N equal parts across its axis, in order to decrease the first coordinate of the lattice vectors, and then performing

the usual spherical sieving inside each part, in order to guarantee that the last $(n-1)$ coordinates doesn't grow too much. The cylindrical sieving algorithms from the previous section (see Algorithms 4.4 and 4.5), are based on the idea that the length of the difference between two random vectors on an n-dimensional sphere is, with high probability, about the factor $\sqrt{2}$ bigger than the radius of the sphere. For the analysis of the complexity of the algorithms it means that, in order to guarantee that the radius of the cylinder growth at most by the factor $\sqrt{2}$, we loose only polynomial amount of points inside each of N chunks of the cylinder during the sieving process.

In this section, we consider what happens, if we become more restrictive about the growth of the radius of the cylinder. Namely, we say that now the growth rate $\gamma \in (1; \sqrt{2}]$ of the radius of the cylinder is an additional parameter of the algorithm.

When $\gamma = \sqrt{2}$, we obtain the algorithms from the previous section. Introducing the new parameter γ adds an additional trade-off between the number of chunks of the cylinder and the time, spent inside one chunk. When $\gamma < \sqrt{2}$, the number of points needed inside one chunk becomes exponential in the dimension, but, since the growth rate of the radius of the cylinder is smaller, we can allow more iterations of the algorithm and, therefore, we may let the decrease rate N of the height of the cylinder be smaller.

The algorithms for $\gamma < \sqrt{2}$ remains essentially the same as described in Algorithms 4.3 to 4.5. We obtain the cylindrical sieving for arbitrary γ just by replacing $\sqrt{2}$ by the desired value of γ in the algorithms. In the same time, the analysis of the resulting complexity is slightly different.

In order to re-estimate the complexity of the cylindrical sieving for $\gamma \neq \sqrt{2}$, we first consider the complexity of one sieving step (corresponds to Algorithm 4.3), which is the same independently of the input lattice, and then, as before, we estimate separately the complexity of the whole algorithm for an arbitrary lattice and for a lattice with a prime volume.

4.3.1 One step of cylindrical sieving with $\gamma < \sqrt{2}$.

The algorithm that performs one step of the cylindrical sieving with the decrease rate $\gamma \neq \sqrt{2}$ can be obtained from Algorithm 4.3 by replacing one line in pseudo-code. The modified version with arbitrary γ is presented by Algorithm 4.6.

Algorithm 4.6: One iteration of sort-and-sieve algorithm.

input : decrease rate N, increase rate $\gamma \in (1; \sqrt{2})$, list $L \subset \Lambda \cap h \times B_{n-1}(R)$, $|L| > N \cdot \mathrm{cov}(n, \gamma) + 1$

output: list $L' \subset \Lambda \cap \frac{h}{N} \times B_{n-1}(R)$, $|L'| > |L| - N \cdot \mathrm{cov}(n, \gamma) - 1$

1 The algorithm coincides with Algorithm 4.3 everywhere except Line 9. At Line 9, checking inequality $\|\mathbf{u_x} - \mathbf{u_c}\| \leqslant R\sqrt{2}$ is replaced by checking $\|\mathbf{u_x} - \mathbf{u_c}\| \leqslant R\gamma$.

In this section, we analyse the complexity of Algorithm 4.6 as a function of the decrease rate γ.

As in Section 4.2.1, for the analysis of the complexity of one sieving step, we assume that vectors, composed by the last $(n-1)$ coordinates of the vectors from the input list, behave like uniformly distributed on the sphere. As before, the key point in the analysis of the complexity is the number of random points, needed to cover a sphere. When $\gamma = \sqrt{2}$, one point covers a half of the sphere. Now, one point covers a spherical cap of an angular radius less than $\frac{\pi}{2}$. We further use the following straightforward corollary from Theorem 3.7, in order to estimate number of points lost in the sieving process with $\gamma < \sqrt{2}$.

Corollary 4.2. Let n be a positive integer number and let $\gamma \in (1; \sqrt{2})$. Let $\alpha =$

$\arcsin\left(\sqrt{\gamma^2 - \frac{\gamma^4}{4}}\right)$ and let N be bigger than

$$\mathrm{cov}(n,\gamma) := 2\ln(2)n^2 \cdot \left(\gamma^2 - \frac{\gamma^4}{4}\right)^{-\frac{n-1}{2}}.$$

Let $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_N\}$ be a list of points N sampled independently from the uniform distribution on S^{n-1}. Then, the probability that the fraction of the sphere, covered by spherical caps of the angular radius α centered at the points from $\mathbf{C}$ is at least $1 - 2^{-d}$ is greater than or equal to $1 - 2^{-d}$.

The complexity of Algorithm 4.6 is described by Lemma 4.4.

Lemma 4.4. Let $\Lambda \subset \mathbb{Z}^n$ be an n-dimensional lattice. Let $h, R > 0$ and let $\gamma \in (1; \sqrt{2})$. Let $\mathrm{cov}(n,\gamma)$ be as defined in Corollary 4.2. Let $L \subset \Lambda$ be a list of lattice vectors, such that all vectors from L satisfy the following conditions:

$$\forall \mathbf{v} \in L, \quad \mathbf{v} = \begin{pmatrix} v_1 \in \mathbb{Z} \\ \mathbf{u}_v \in \mathbb{Z}^{n-1} \end{pmatrix}, \quad 0 < v_1 \leqslant h, \quad \|\mathbf{u}_v\| \leqslant R.$$

We assume that $|L| = O(N\mathrm{cov}(n,\gamma))$ and that N and γ are chosen in such a way that $|L| \leqslant 2^{n/2}$. Then, if Assumption 4.1 holds, Algorithm 4.6, given as input the list L, outputs, with the probability greater than $1-2^{-n}N$, a list $L' \subset \Lambda$ of size at least $|L| - N\mathrm{cov}(n,\gamma) - 1$, such that all vectors from L' satisfy the conditions:

$$\forall \mathbf{w} \in L', \quad \mathbf{w} = \begin{pmatrix} w_1 \in \mathbb{Z} \\ \mathbf{u}_w \in \mathbb{Z}^{n-1} \end{pmatrix}, \quad 0 < w_1 \leqslant \frac{h}{N}, \quad \|\mathbf{u}_w\| \leqslant R \cdot \gamma, \tag{4.23}$$

in time $\widetilde{O}(\mathrm{cov}(n,\gamma)|L|) = \widetilde{O}(N \cdot \mathrm{cov}(n,\gamma)^2)$ using $\widetilde{O}(|L|) = \widetilde{O}(N \cdot \mathrm{cov}(n,\gamma))$ memory.

Proof. The proof is similar to the proof of Lemma 4.3. The only part that is slightly different, is the estimation of the number of points lost during the sieving process. Hence, here we provide only that part of the proof.

As before, our goal is to estimate the number of points that the algorithm choose as centers,i.e., to estimate the following sum: $\sum_{i=1}^{N} |C_i|$. Let ε_f and ε_p be two positive numbers close to zero. Denote as $C_\gamma(\varepsilon_f, \varepsilon_p)$ a positive integer number such that the following expression is true:

$$\mathbb{P}\{C_\gamma(\varepsilon_f, \varepsilon_p) \text{ spherical caps of angular radius } \gamma \text{ cover } 1 - \varepsilon_f \text{ of } S^{n-1}\} \geqslant 1 - \varepsilon_p.$$

Then, consider the following two possibilities for the size of one sublist L_i. First, the size of the i-th sublist can be smaller than $C_\gamma(\varepsilon_f, \varepsilon_p)$. In that case, even if we loose the whole sublist, we do not loose more than $C_\gamma(\varepsilon_f, \varepsilon_p)$ points.

Another possibility is that $|L_i|$ is bigger than $C_\gamma(\varepsilon_f, \varepsilon_p)$. In that case, under Assumption 4.1, after the number of centers in L_i have reached $C_\gamma(\varepsilon_f, \varepsilon_p)$ points, new points become centers only with the probability at most ε_f. Therefore, the number of lost centers does not exceed $C_\gamma(\varepsilon_f, \varepsilon_p) + O(\varepsilon_f)(|L_i| - C_\gamma(\varepsilon_f, \varepsilon_p))$ with probability at least $1 - \varepsilon_p$. Then, for the overall number of centers we obtain:

$$\sum_{i=1}^{N} |C_i| \leqslant \sum_{i=1}^{N} C_\gamma(\varepsilon_f, \varepsilon_p) + O(\varepsilon_f)(|L_i| - C_\gamma(\varepsilon_f, \varepsilon_p)) \leqslant N \cdot C_\gamma(\varepsilon_f, \varepsilon_p) + O(\varepsilon_f|L|),$$

with probability at least $1 - \varepsilon_p$. If we choose $\varepsilon_f = \varepsilon_p = 2^{-n}$, then $C_\gamma(2^{-n}, 2^{-n})$ is equal to $\mathrm{cov}(n,\gamma)$; as we assume that $|L| < 2^{n/2}$, the number of lost points is bounded by $N \cdot \mathrm{cov}(n,\gamma) + 1$.

The probability of the success of the algorithm can be lower bounded by the probability that, for each sublist, $C_\gamma(\varepsilon_f, \varepsilon_g)$ points cover $1-\varepsilon_f$ of the sphere surface:

$$p \geqslant (1-\varepsilon_p)^N \geqslant 1 - \frac{N}{2^n}.$$

□

4.3.2 Sort-and-sieve algorithm for a lattice with a prime volume

In this section, we compute the complexity of cylindrical sieving with $\gamma < \sqrt{2}$ in case of a lattice with a prime volume.

By replacing Algorithm 4.3 by Algorithm 4.6, parameterized by $\gamma \in (1; \sqrt{2})$, in the sort-and-subtract algorithm (see Algorithm 4.4), we obtain a continuum of more efficient algorithms (see Algorithm 4.7).

Algorithm 4.7: Sort-and-sieve algorithm for lattices with prime volume

input : a basis $\mathbf{B}$ of a lattice Λ with a prime volume p, parameter $\gamma \in (1; \sqrt{2})$.
output: $\mathbf{v} \in \Lambda$ such that $0 < \|\mathbf{v}\| \leqslant \alpha(\gamma)\sqrt{\gamma_n}\,\mathrm{vol}(\Lambda)^{1/n}$.

```
/* initial enumeration part */
```
$(p, \mathbf{a}) \to \mathrm{HNF}(\mathbf{B})$
$L_0 \to \textsc{EnumerateCylinder}(p, \mathbf{a}, R_0(\gamma))$ ▷ Algorithm 4.1
```
/* sieving part */
```
$k \leftarrow \left\lceil \frac{\log(p)}{n\log(\gamma)} \right\rceil$
for $(i \in \{1, \ldots, k\})$ **do**
  $L_i \leftarrow \textsc{Sieve}(\gamma^n, \gamma, L_{i-1})$ ▷ Algorithm 4.6
  if $(\exists \mathbf{v} \in L$ such that $0 < \|\mathbf{v}\| \leqslant \alpha\sqrt{\gamma_n}\,\mathrm{vol}(\Lambda)^{1/n})$ **then**
    return $\mathbf{v}$

Theorem 4.3. Let $\gamma \in (1; \sqrt{2})$. Let n and p be as in Theorem 4.1. let Λ be an n-dimensional integer lattice with a prime volume $\mathrm{vol}(\Lambda) = p$. Let Assumption 4.1 hold. Let $R_0 > 0$ be such that $|B_{n-1}(R_0) \cap \mathbb{Z}^{n-1}| = \widetilde{O}\left(1 - \frac{\gamma^2}{4}\right)^{-n/2}$. Then, Algorithm 4.7, given as input a basis of the lattice Λ, with probability at least $1 - 2^{(\log(\gamma)-1)n+o(n)}$, outputs a non-zero lattice vector of length less then or equal to $\alpha(\gamma)\sqrt{\gamma_n}\,\mathrm{vol}(\Lambda)^{1/n}$, where $\alpha(\gamma) = \frac{\gamma}{\sqrt{\gamma_n}} R_0$. The memory complexity of the algorithm is $\widetilde{O}\left(\left(1 - \frac{\gamma^2}{4}\right)^{-n/2}\right)$, the time complexity is $\widetilde{O}\left(\left(\gamma - \frac{\gamma^3}{4}\right)^{-n}\right)$.

Proof. The structure of the proof is similar to the proof of Theorem 4.1. Essentially, the proof can be obtained by replacing $\sqrt{2}$ by the parameter γ in the proof of Theorem 4.1, and by replacing the number of vectors, lost at one iteration, according to Lemma 4.4.

By doing so for (4.11) that describes how the size of the list and the geometry of the vectors in the list update after one iteration, we obtain:

$$|L_{i+1}| \geqslant |L_i| - \mathrm{cov}(n, \gamma) \cdot N, \quad R_{i+1} \leqslant R_i \cdot \gamma, \quad h_{i+1} \leqslant \frac{h_i}{N}. \tag{4.24}$$

As before, the goal is to obtain a vector of length that is equal to some constant multiple of Minkowski's bound of the lattice. Therefore, we choose the number of iterations k and the decrease rate N such that, after k iterations, h_k satisfies the condition, given by (4.14), and R_k satisfies the condition, obtained from (4.13) by replacing $\sqrt{2}$ by γ:

$$R_k = R_0 \cdot \gamma^k \leqslant \sqrt{\frac{n-1}{n}} \cdot \alpha\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}, \tag{4.25}$$

Proceeding as in the proof of Theorem 4.1, we choose the following values of the parameters k and N:

$$k = \left\lceil \frac{\mathrm{vol}(\Lambda)}{n \log(\gamma)} \right\rceil, \quad N = \gamma^n, \tag{4.26}$$

and we also obtain the following bound for the ratio of the quality of the output α and the radius of the initial sampling R_0:

$$\frac{\alpha}{R_0} \geqslant \frac{\gamma}{\sqrt{n-1}}. \tag{4.27}$$

Using the obtained values of the parameters k and N, we get the time and memory complexities of the algorithm in the same way, as we obtain (4.18) and (4.20) in the proof of Theorem 4.1:

$$\mathcal{S}(\gamma) = \widetilde{O}(\mathrm{cov}(n,\gamma) \cdot N) = \widetilde{O}\Big(\big(1 - \frac{\gamma^2}{4}\big)^{-n/2}\Big), \tag{4.28}$$

$$\mathcal{T}(\gamma) = \widetilde{O}(\mathrm{cov}(n,\gamma)^2 \cdot N) = \widetilde{O}\Big(\big(\gamma - \frac{\gamma^3}{4}\big)^{-n}\Big). \tag{4.29}$$

In order to estimate the probability of success of the algorithm, we estimate the product of probabilities of success at each iteration. By Lemma 4.4, we obtain

$$p \geqslant 1 - 2^{-n} N \cdot \Theta(\mathrm{vol}(\Lambda)/n) = 1 - \gamma^n \cdot 2^{-n+o(n)}. \tag{4.30}$$

The radius of the initial sampling, and, therefore, the quality of the output of the algorithm, is defined by the required size of the initial list, which is given by (4.18). It can be numerically computed for a particular value of γ using Theorem 3.2. □

Denote as γ_{opt} the value of the parameter γ that optimizes the time complexity of Algorithm 4.7:

$$\gamma_{\mathrm{opt}} = \operatorname*{arg\,min}_{\gamma \in (1;\sqrt{2})} \Big(\gamma - \frac{\gamma^3}{4}\Big) = \frac{2}{\sqrt{3}}.$$

The complexity of Algorithm 4.7, given $\gamma = \gamma_{\mathrm{opt}}$, coincides with the complexity of the overlattice sieve algorithm [BGJ14]. The length of the lattice vector, recovered by Algorithm 4.7 with $\gamma = \gamma_{\mathrm{opt}}$, does not exceed the Minkowski's bound of the lattice.

Corollary 4.3. Let $\gamma_{\mathrm{opt}} = \frac{2}{\sqrt{3}}$ be the growth rate of Algorithm 4.7. Let n and p be as in Theorem 4.3. Then, if Assumption 4.1 holds, Algorithm 4.7, given as input an n-dimensional lattice Λ with the prime volume p, with probability at least $1 - 2^{-0.792n+o(n)}$, outputs a non-zero lattice vector of length at most $0.236\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}$. The time complexity of the algorithm is equal to $\widetilde{O}(2^{0.3774n})$, the memory complexity is $\widetilde{O}(2^{0.292n})$.

Proof. The complexity is obtained straightforwardly from Theorem 4.3 by substituting γ with γ_{opt}. In order to estimate the quality of the output of the algorithm, we need to choose the radius R_0 of the initial hypercylinder. We need to choose R_0 in such a way that an $(n-1)$-dimensional ball of radius R_0 contains $\mathcal{S}(\gamma_{\mathrm{opt}})$ integer points. Using Theorem 3.2, we obtain $R_0 = \sqrt{0.0418n}$. Therefore, by (4.27) requires that the quality of the output α should at least $\frac{2}{\sqrt{3}} \cdot \sqrt{0.0418}$. The probability of success of the algorithm follows from (4.30). □

If we optimize the memory complexity instead of the time, we get $\gamma_{\mathrm{opt}} \to 1$, which corresponds to the Nguyen-Vidick sieve algorithm [NV08].

Algorithm 4.8: Sort-and-sieve algorithm for lattices with prime volume

input : a basis $\mathbf{B}$ of a lattice $\Lambda \subset \mathbb{Z}^n$, parameter $\gamma \in (1; \sqrt{2})$.
output: $\mathbf{v} \in \Lambda$ such that $0 < \|\mathbf{v}\| \leqslant \beta\gamma\sqrt{\gamma_n}\,\mathrm{vol}(\Lambda)^{1/n}$.

/* initial enumeration part */
$L_0 \leftarrow$ EnumerateCylinder$(\mathbf{B}, q, \beta)$ ▷ Algorithm 4.2
/* sieving part */
$k \leftarrow \left\lceil \frac{n-1}{2} \cdot \frac{\log(q)}{\log(\gamma)} \right\rceil$
for $(i \in \{1, \ldots, k\})$ **do**
  $L_i \leftarrow$ Sieve$(\gamma^n, \gamma, L_{i-1})$ ▷ Algorithm 4.6
  if $(\exists \mathbf{v} \in L$ such that $0 < \|\mathbf{v}\| \leqslant \alpha\sqrt{\gamma_n}\,\mathrm{vol}(\Lambda)^{1/n})$ **then**
    return v

4.3.3 Sort-and-sieve algorithm for any lattice

In this section, we estimate the complexity of the cylindrical sieving with the parameter $\gamma \in (1; \sqrt{2})$ in the general case. That is, we replace the growth rate $\sqrt{2}$ in Algorithm 4.5 by the parameter γ, and we compute the complexity of the resulting algorithm as a function of γ. The parameters of the generalized algorithm are summarized in Algorithm 4.8.

The complexity of the sort-and-sieve algorithm for any lattice is described in Theorem 4.4.

Theorem 4.4. Let $\gamma \in (1; \sqrt{2})$. Let Λ be an n-dimensional integer lattice. Let Assumption 4.1 hold. Let $\beta > 0$ be such that $\beta^n = \widetilde{O}\left(1 - \frac{\gamma^2}{4}\right)^{-n/2}$.

Then, Algorithm 4.8, given as input a basis of the lattice Λ, with probability at least $1 - 2^{(\log(\gamma)-1)n+o(n)}$, outputs a non-zero lattice vector of length less then or equal to $\frac{\beta\gamma}{2\pi e}\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}$. The memory complexity of the algorithm is $\widetilde{O}\left(\left(1 - \frac{\gamma^2}{4}\right)^{-n/2}\right)$. The time complexity of the sieving part of the algorithm is $\widetilde{O}\left(\left(\gamma - \frac{\gamma^3}{4}\right)^{-n}\right)$. The time complexity of the initial enumeration part is given by $\widetilde{O}(e^{\frac{n\beta^2}{2e}})$, when $\beta \leqslant \sqrt{e}$, otherwise it is equal to $\widetilde{O}(\beta)$.

Proof. The proof is obtained by taking the proof of Theorem 4.2, replacing the growth rate $\sqrt{2}$ by the parameter γ, and replacing the estimate for the number of points lost by the algorithm after one iteration of sieving given by Lemma 4.3 with the estimate by Lemma 4.4. The probability of success of the algorithm can be estimated exactly as in Theorem 4.3. We do not rewrite all the steps again, instead, here we just provide the resulting parameters of the algorithm. The number of iterations k and the decrease rate N are given by:

$$k = \left\lceil \frac{n-1}{2} \cdot \frac{\log(q)}{\log(\gamma)} \right\rceil, \quad N = \gamma^n. \tag{4.31}$$

The ratio of the length of the vector, returned by Algorithm 4.8, and of the Minkowski's bound of the lattice is upper bounded by $\alpha = \frac{\beta\gamma}{2\pi e}$. □

Ignoring polynomial factors, the time complexity of Algorithm 4.8 is the maximum of the complexities of the initial enumeration part and of the sieving part:

$$\mathscr{T} = \widetilde{O}(\max(\mathscr{T}_{\text{sieving}}, \mathscr{T}_{\text{initial enumeration}})).$$

Denote as $\widehat{\gamma}_{\text{opt}}$ the value of the parameter γ that optimizes the time complexity of Algo-

rithm 4.7. It can be computed numerically.

$$\widehat{\gamma}_{\text{opt}} = \underset{\gamma\in(1;\sqrt{2})}{\arg\min}\left(\max\left(e^{\frac{n\beta(\gamma)^2}{2e}}, (\gamma - \frac{\gamma^3}{4})^{-n}\right)\right) \approx 1.10378. \tag{4.32}$$

The complexity of Algorithm 4.8 with the parameter $\gamma = \widehat{\gamma}_{\text{opt}}$, as well as the complexity with the value of the parameter $\gamma = \gamma_{\text{opt}}$ that optimized only the complexity of the sieving part, is given in the following corollary from Theorem 4.4.

Corollary 4.4. Let $\gamma_{\text{opt}} = \frac{2}{\sqrt{3}}$ and let $\widehat{\gamma}_{\text{opt}} \approx 1.10378$ be as in (4.32). Let Assumption 4.1 hold.

Then, Algorithm 4.8, given as input an n-dimensional integer lattice Λ and γ_{opt}, outputs a non-zero lattice vector of length at most $0.342\sqrt{n}\,\text{vol}(\Lambda)^{1/n}$. The time complexity of the initial enumeration is $\widetilde{O}(2^{0.398n})$; the time complexity of the sieving part of the algorithm is equal to $\widetilde{O}(2^{0.3774n})$. The memory complexity of the algorithm is $\widetilde{O}(2^{0.292n})$.

If the parameter γ of the algorithm is equal to $\widehat{\gamma}_{\text{opt}}$, then the time complexity of the initial enumeration part and of the sieving part are equal up to polynomial factor. The time complexity of the algorithm for $\gamma = \widehat{\gamma}_{\text{opt}}$ is $\widetilde{O}(2^{0.3816n})$, the memory complexity is $\widetilde{O}(2^{0.262n})$; the length of the lattice vector, returned by the algorithm is at most $0.32\sqrt{n}\,\text{vol}(\Lambda)^{1/n}$.

Table 4.2 summarizes the complexities of all the algorithms, presented in this chapter.

	prime volume		any lattice	
$\gamma = \sqrt{2}$	time	$2^{n/2}$	enum. time	$2^{0.531n}$
	memory	$2^{n/2}$	sieving time	$2^{n/2}$
	output	0.41	memory	$2^{n/2}$
			output	0.484
$\gamma = \frac{3}{\sqrt{2}}$	time	$2^{0.3774n}$	enum. time	$2^{0.396n}$
	memory	$2^{0.292n}$	sieving time	$2^{0.3774n}$
	output	0.236	memory	$2^{0.292n}$
			output	0.342
$\gamma \approx 1.10378$			time	$2^{0.3816n}$
			memory	$2^{0.262n}$
			output	0.32

Table 4.2 – Complexity of finding short lattice vectors using cylindrical sieving. The first row in the table represents the value of the growth rate used by an algorithm; $\gamma = \sqrt{2}$ corresponds to the sort-and-subtract algorithm, other values of γ correspond to the sort-and-sieve algorithm. The row, marked "output", represents the upper bound of the norm of the lattice vector, returned by the algorithm, divided by the Minkowski's bound of the lattice.

Chapter 5

Finding short vectors for lattices with a small prime volume

In the previous chapter, for the analysis of the complexity of algorithms, we do not make any assumptions on the size of the volume of the lattice. We only assume that the description of an input lattice can be stored using polynomial in the dimension amount of space, which implies that the logarithm of the volume of the lattice is also polynomial in the dimension.

In this chapter, we apply cylindrical sieving to a lattice with a small prime volume. That is, now we consider a lattice Λ with a prime volume $p = 2^{\Theta(n)}$.

In order to illustrate why cylindrical sieving may be efficient for lattices with a small volume, we start with a small example. Consider the sort-and-subtract algorithm applied to an n-dimensional lattice Λ with a prime volume, such that $p = \mathrm{vol}(\Lambda)$ is roughly equal to 2^n. Let the size of the initially generated list of lattice points L_0 be equal to $\mathrm{poly}(n) \cdot 2^{n/4}$. Initially, L_0 is contained inside a hypercylinder $p \times S^{n-2}(\sqrt{\alpha(n-1)})$, where $\alpha > 0$ is some constant. As we take a list of vectors of size $\mathrm{poly}(n) \cdot 2^{n/4}$, the decrease rate of the first coordinate of the vectors from the list is about $2^{n/4}$. Then, after 4 iterations, the first coordinate of the vectors is completely reduced:

$$2^n \to \frac{2^n}{2^{n/4}} = 2^{3n/4} \to 2^{n/2} \to 2^{n/4} \to O(1).$$

In the sort-and-subtract algorithm, the increase rate of the last $(n-1)$ coordinates of the vectors is equal to $\sqrt{2}$, then, after four iterations, the norm of the vectors, formed by the last $(n-1)$ coordinates, is increased only by a factor $\sqrt{2}^4 = 4$. The Minkowski's bound for a lattice with volume 2^n is $2\sqrt{\gamma_n} = O(\sqrt{n})$. Therefore, using cylindrical sieving, for a lattice with a prime volume $p \approx 2^n$, we can find a vector of length equal to a constant approximation of the Minkowski's bound in time $\widetilde{O}(2^{n/4})$, which is much faster than any existing algorithm that solves SVP in the general case.

Similarly, if we take the initial list of size $\widetilde{O}(2^{n/\log(n)})$, we obtain an algorithm that finds a polynomial approximation of the Minkowski's bound of the lattice in a subexponential time.

In this chapter, we consider the complexity of finding short vectors for lattices with a prime volume equal to $2^{\Theta(n)}$ using cylindrical sieving. It is organized as follows. First, we adapt the sort-and-sieve algorithm for lattices with a volume around 2^{cn} and consider its complexity as a function of c and of the length of the vector, returned by the algorithm. Then, we numerically estimate the complexity of the algorithm for various values of c and of the quality of the output, and compare the obtained algorithm with existing techniques for lattices with a small volume.

5.1 Complexity of cylindrical sieving for lattices with small prime volume

In this section, we consider the complexity of the sort-and-sieve algorithm applied to a lattice with a prime volume $p \approx 2^{cn}$. If the logarithm of the volume of the lattice is the constant multiple of the dimension, and if the size of the initial list of lattice vectors is exponential in n, then, in order to decrease the first coordinate of the vectors, we need only constant number of iterations. The last $(n-1)$ coordinates grow with each iteration by a constant factor $\gamma \in (1; \sqrt{2}]$. Then, since initially generated vectors have length proportional to $\sqrt{n}$, after the constant number of iterations we obtain a vector of length equal to Minkowski's bound of the lattice multiplied by some constant. How close is the length of the vector, returned by the algorithm, to the Minkowski's bound, depends on the radius of the initial hypercylinder and on the number of iterations. The number of iterations is defined by the exact size of the volume of the lattice and by the growth rate γ of the algorithm. Theorem 5.1 describes the choice of the parameters and the complexity of the resulting algorithm (see Algorithm 5.1) in a small volume case.

Algorithm 5.1: Sort-and-sieve algorithm for lattices with small prime volume

input : a basis $\mathbf{B}$ of a lattice Λ with a prime volume $p = 2^{cn}$, parameter $\gamma \in (1; \sqrt{2})$.
output: $\mathbf{v} \in \Lambda$ such that $0 < \|\mathbf{v}\| \leqslant \beta\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}$.

```
/* initial enumeration part */
```
$(p, \mathbf{a}) \to \mathrm{HNF}(\mathbf{B})$
$L_0 \to \textsc{enumerateCylinder}(p, \mathbf{a}, R_0(\gamma))$ ▷ Algorithm 4.1
```
/* sieving part */
```
$k \leftarrow \left\lceil \frac{c+\log(\beta/\sqrt{\alpha})}{\log(\gamma)} \right\rceil$
for $(i \in \{1, \ldots, k\})$ **do**
  $L_i \leftarrow \textsc{Sieve}(\gamma^{\frac{cn}{c+\log(\beta/\sqrt{\alpha})}}, \gamma, L_{i-1})$ ▷ Algorithm 4.6
  if $(\exists \mathbf{v} \in L$ such that $0 < \|\mathbf{v}\| \leqslant \beta\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n})$ **then**
    return $\mathbf{v}$

Theorem 5.1. Let n be a positive integer number and let p be a prime, such that $c = \frac{\log(p)}{n}$ is a positive constant. Let $\gamma \in (1; \sqrt{2})$, Let Λ be an n-dimensional lattice with $\mathrm{vol}(\Lambda) = p$. Denote $\frac{c}{c+\log(\beta/\sqrt{\alpha})}$ as μ, and denote $\left(\frac{\gamma^{\mu-2}}{1-\frac{\gamma^2}{4}}\right)^n$ as S. Let $\alpha > 0$ be such that the ball $B_{n-1}(\sqrt{\alpha(n-1)})$ contains $\widetilde{O}(S)$ integer points and let β be bigger than $\sqrt{\alpha}$. Let the decrease rate of Algorithm 5.1 be equal to $N = \gamma^\mu$.

Then, if Assumption 4.1 holds, Algorithm 5.1, given a basis of the lattice Λ, with probability at least $1 - \Theta(2^{-n/2})$, returns a non-zero lattice vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{v}\| \leqslant \beta\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}$. The time complexity of Algorithm 5.1 is $\mathscr{T} = \widetilde{O}\left(\frac{\gamma^{2(\mu-1)}}{1-\frac{\gamma^2}{4}}\right)^{n/2}$, the memory complexity is $\mathscr{S} = \widetilde{O}(S)$.

Proof. The proof is similar to the proof of Theorem 4.3. The goal of the algorithm is to obtain a vector of length $\beta\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}$ with β as small as possible. In order to achieve it, iterations of the algorithm k that satisfies the following condition (analogous to (4.14) and (4.25)) are satisfied:

$$\sqrt{\alpha(n-1)} \cdot \gamma^k \leqslant \beta\sqrt{n-1} \cdot 2^c; \tag{5.1}$$

and we choose the decrease rate of the height of the cylinder such that after k iterations the height is small enough:

$$\frac{2^{cn}}{N^k} \leqslant \beta \cdot 2^c. \tag{5.2}$$

The number of iterations $k = \frac{c+\log(\beta/\alpha)}{\log(\gamma)}$ satisfies the condition given by (5.1). Then, according to (5.2), we can choose $N = \gamma^\mu$, where μ is as described in the theorem. Using obtained values of k and N, we can derive the time and memory complexity of the algorithm in the same way as in the proof of Theorem 4.1.

By Lemma 4.4, the probability of success of one iteration of sieving is $(1 - N \cdot 2^{-n+o(n)})$. Then, the probability of success of the whole algorithm is

$$(1 - N \cdot 2^{-n+o(n)})^k \geqslant 1 - 2^{-n+\mu\log(\gamma)+o(n)}.$$

As μ is always less than 1 and $\gamma < \sqrt{2}$, the probability of success is at least $1 - 2^{-n/2+o(n)}$. □

Hence, in order to find optimal parameters for approximating the Minkowski's bound of a lattice with a volume $p \approx 2^{cn}$ up to the factor $\beta \geqslant 1$ by Algorithm 4.7, we need to solve the following optimization problem:

$$\begin{gathered} |\mathbb{Z}^{n-1} \cap B_{n-1}(\sqrt{\alpha(n-1)})| \geqslant \mathcal{S}(c,\alpha,\beta,\gamma), \\ \beta \geqslant \sqrt{\alpha}, \\ (\alpha,\gamma) = \underset{\gamma\in(1;\sqrt{2}),\alpha>0}{\arg\min} \mathcal{T}(c,\alpha,\beta,\gamma), \end{gathered} \tag{5.3}$$

where $\mathcal{T}$ and $\mathcal{S}$ are as described in Theorem 5.1.

5.2 Complexity of finding short lattice vectors for a lattice with a small prime volume

In this section, we estimate the complexity of finding short lattice vectors for lattices with a prime volume $p = 2^{cn}$ for various values of c.

First, consider a problem of finding a lattice vector of length shorter than Minkowski's bound for a lattice with a small prime volume. Up to our knowledge, there is no exact SVP algorithm that considers the case of lattices with a small volume separately. Therefore, for finding vectors shorter than Minkowski's bound, we compare the cylindrical sieving algorithm from the previous section with the fastest heuristic sieving algorithm for solving SVP [BDGL16] for an arbitrary lattice.

In order to estimate the time complexity of the cylindrical sieving for lattices with a small prime volume, we implement a python script that, given as input $c = \frac{\log(\mathrm{vol}(\Lambda))}{n}$ and the parameter $\beta = \left(\frac{\|\mathbf{v}\|}{\sqrt{n}\,\mathrm{vol}(\Lambda)}\right)^{1/n}$ that describes the length of the vector, returned by Algorithm 5.1, finds optimal parameters of Algorithm 5.1 (see (5.3)).

Our estimates show that for lattices with a volume smaller than $2^{1.71n}$, the cylindrical sieving finds vectors of length shorter than $\sqrt{\frac{1.744n}{2\pi e}} \cdot \mathrm{vol}(\Lambda)^{1/n}$ faster then the current fastest algorithm for SVP [BDGL16]. For example, for a lattice with a volume around 2^n,the estimated time complexity of Algorithm 5.1 is $\widetilde{O}(2^{0.229n})$, while the complexity of the algorithm [BDGL16] is $\widetilde{O}(2^{0.292n})$. The results are presented in Figure 5.1.

We also consider a problem of finding a vector of length that is at most a constant factor longer than the shortest vector of a lattice. In that case, we can compare the cylindrical sieving with Cheon's technique for solving approximate lattice problems that also exploits the small volume of the input lattice (see Section 3.2).

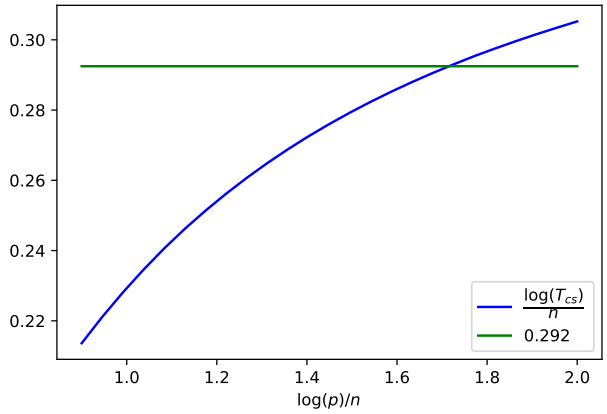


Figure 5.1 – Time complexity of finding a lattice vector shorter than Minkowski's bound for lattices a small prime volume. n denotes the dimension, p denotes the volume of a lattice, T_{cs} denotes the time complexity of finding a vector of length $\sqrt{\frac{1.744n}{2\pi e}}\,\mathrm{vol}(\Lambda)^{1/n}$ by cylindrical sieving (represented by blue line); $0.292n$ is the logarithm of the time complexity of the algorithm [BDGL16] (represented by green line).

We estimate the time complexities of both approaches for a lattice with $\mathrm{vol}(\Lambda) \approx 2^n$ for a range of approximation factors. The results are presented in Figure 5.2. When the required length of the vector returned by the algorithm is close to Minkowski's bound, the cylindrical sieving outperforms Cheon's technique. But, for bigger approximation factors, Cheon's technique is faster then the cylindrical sieving.

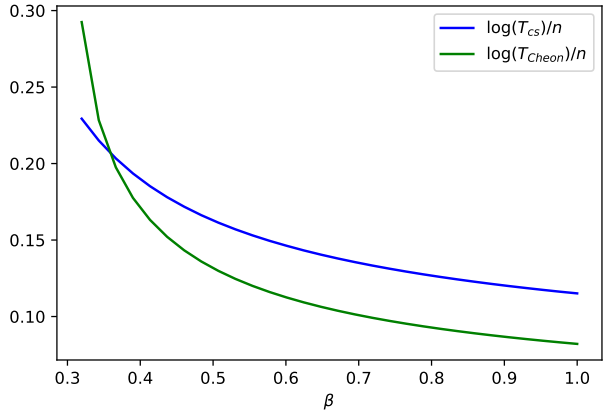


Figure 5.2 – Complexity of solving approximate SVP by cylindrical sieving and by Cheon's technique for a lattice with a prime volume $p \approx 2^n$. β denotes the ratio $\frac{\|\mathbf{v}\|}{\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}}$, where $\mathbf{v}$ is the vector returned by an algorithm. The blue line represents the logarithm of the time complexity of the cylindrical sieving, the green line represents the Cheon's technique.

Chapter 6

Solving the Closest Vector Problem with cylindrical sieving

In this chapter, we adapt the cylindrical sieving to solve the Closest Vector Problem.

In Chapter 4, we describe the cylindrical sieving algorithm for finding short lattice vectors (see Algorithms 4.4 and 4.5). We recall in short how the cylindrical sieving works for solving SVP. First, the algorithm generates a long list L of lattice vectors that lie inside a long and narrow hypercylinder. Then, the algorithm iteratively repeats the following procedure: it considers the differences of the vectors from the list and keeps for the next iteration only those that lie inside a much shorter but slightly wider hypercylinder. At the last iteration, we obtain lattice vectors of length smaller than Minkowski's bound of the lattice.

In this chapter, our goal is to solve the Closest Vector Problem. We are given a lattice Λ and a vector $\mathbf{t} \in \mathbb{R}^n$ and the goal is to construct an algorithm that finds a lattice point $\mathbf{v}$ such that the norm of the difference $\mathbf{v}-\mathbf{t}$ is short. Consider the special case of the problem, when $\mathbf{t}$ belongs to the initial hypercylinder of the cylindrical sieving. Then, we can add the target point $\mathbf{t}$ to the list L and run the cylindrical sieving process on the resulting list $L := L \cup \{\mathbf{t}\}$. Assume that at the first iteration the algorithm successfully find a pair $\mathbf{v}_1$ for $\mathbf{t}_1 := \mathbf{t}$ such that $\mathbf{t}_1 - \mathbf{v}_1$ belongs to the next hypercylinder. Then, $\mathbf{t}_2 := \mathbf{t}_1 - \mathbf{v}_1$ becomes a target point for the next iteration. Assume that at each iteration the algorithm finds a suitable pair for $\mathbf{t}_i$ and denote as $\mathbf{v}_i$ the pair, found at the iteration i. Then, the vector $\mathbf{t} - \sum_{i=1}^{k} \mathbf{v}_i$, where k is the number of iteration, belongs to the last hypercylinder of the sieving process. Thus, it is short. As all $\mathbf{v}_i$s are the lattice vectors, the sum $\sum_{i=1}^{k} \mathbf{v}_i$ is a solution of the CVP problem. The complexity of the resulting algorithm coincides with the complexity of solving SVP with the cylindrical sieving.

This approach has two restrictions. First, it can be applied only to the targets that lie inside a certain hypercylinder. Second, we do not have a guarantee that there is a suitable pair for the target among the vectors from the list. Consider these restrictions separately.

The first problem can be solved by reducing the target point $\mathbf{t} \in \mathbb{R}^n$ with the lattice basis that we used in Chapter 4 to sample the initial list of lattice vectors. For prime volume lattices, it is the basis given by the Hermite Normal Form of the lattice; for an arbitrary lattice we can obtain the basis using the unbalanced reduction.

The second restriction is harder to handle. In order to analyze the behavior of the list of lattice vectors together with the target point during the iterations of the sieving process, we need some assumptions on their distribution. We randomize the target point by adding a random lattice vector to it, and assume that at each iteration it behaves as if it is sampled from the uniform distribution on the corresponding hypercylinder.

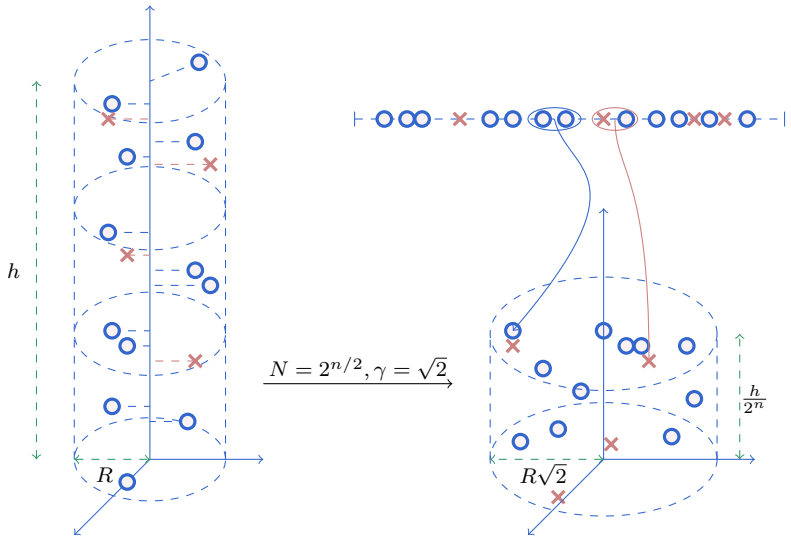


Figure 6.1 – One iteration of the cylindrical sieving for solving CVP. Blue points denote the lattice points, red points denote the target points for approximation. The line with blue and red points in the upper right part of the picture denotes the projection of the input points on the axis that corresponds to the first coordinate of the input vectors.

Also, to predict the number of points, needed to cover most of the surface of each hypercylinder, we need some assumptions on the behavior of the lattice points from the list. For solving CVP, we assume that the points from the list behave as sampled from a uniform-like distribution. Informally, we assume that the distribution has no holes, i.e., for not too small pieces of the hypercylinder, their probability is proportional to their volume. See Assumption 6.1 for the precise description of the assumptions made on the distribution.

The cylindrical sieving adapts nicely to solve the Closest Vector Problem with preprocessing. If we store the lists of the lattice points after one run of the cylindrical sieving, we can reuse them to solve new CVP instances for the given lattice. Then, to find a close vector for a new target we just iteratively reduce it with each list. If we spend $\widetilde{O}(2^{\frac{n}{2}})$ time on the preprocessing, one instance of the problem can be solved in the polynomial time in the dimension of the lattice.

This chapter is organized as follows. In Section 6.1, we analyze one iteration of the cylindrical sieving under new assumptions. In Section 6.2, we show how the initial generation of lattice vectors inside a hypercylinder can be combined with the iterative cylindrical sieving to produce the description of a lattice that can be used later for solving CVPP instances. As the initial generation of lattice vectors for prime volume lattices is easier, we describe the preprocessing for prime volume lattices and for the general case separately.

Then, in Section 6.3, we describe how to find close lattice vectors to the target points that lie inside the initial hypercylinder, given a description of the lattice produced by the preprocessing step.

In Section 6.4, we combine everything together and obtain an algorithm that solves the Closest Vector Problem in polynomial time in RAM computational model. Again, we describe two algorithms: one for a prime volume lattice, another for the general case.

Finally, in Section 6.5, we describe the implementation of the algorithm for prime volume lattices and provide experimental results to support the proposed heuristic assumptions.

6.1 One step of preprocessing

Let $N > 1$. In this section, we describe an algorithm that, given as input a list L of lattice vectors, uniformly distributed inside a hypercylinder $C^n_{\mathbf{u}}(h, R)$, produces two new lists C and L' of lattice vectors with the following properties:

— the list C is a subset of L, such that the overwhelming fraction of the hypercylinder $C^n_{\mathbf{u}}(h, R)$ is covered by the half-cylinders of height $\frac{h}{N}$, centered at points from C;

— all the vectors from the list L' lie inside the hypercylinder of much smaller height and slightly smaller radius $C^n_{\mathbf{u}}(\frac{h}{N}, R\sqrt{2})$.

The algorithm is similar to Algorithm 4.3 that performs one sieving step for solving SVP. It also creates an initially empty list of centers C, then goes through the input list L and for each point from L tries to find a pair in the list C. If the search for a pair is not successful, the considered vector from L becomes a new center and goes to the list of centers C. The difference is that now, in addition to obtaining a new list L' of shorter lattice vectors, we also need to produce a coverage C of the hypercylinder $C^n_{\mathbf{u}}(h, R)$. In order to do so, in the preprocessing for CVPP, we keep the centers, produced by the algorithm, instead of throwing them out as in Algorithm 4.3 for SVP.

The analysis of one step of the preprocessing for CVPP compared to one step of sieving for SVP is also different, mainly because now we use a different assumption on the distribution of lattice vectors that an algorithm obtains as an input. Vector $\mathbf{x}$ from a hypercylinder $C^n_{\mathbf{u}}(h, R)$ can be presented as a sum of two orthogonal parts: $(\mathbf{x}^t\mathbf{u})\mathbf{u}$ and $\pi_{\mathbf{u}}(\mathbf{x})$. When analysing SVP, we considered only $\pi_u(\mathbf{x})$, for the rest of the vector the knowledge that $(\mathbf{x}^t\mathbf{u})\mathbf{u}$ is bounded was enough to conclude that we get short enough vectors in the end. Now, as we need to produce a coverage of the hypercylinder, we also need to take into account the part of the vectors, parallel to the cylinder axis. We use a somewhat stronger, but natural assumption that vectors from the input list are uniformly distributed inside the corresponding hypercylinder.

This section is organized as follows. First, we consider a subroutine (see Algorithm 6.1) that reduces one lattice point with a list of centers, i.e., returns a suitable pair for it, or adds it to the list of centers. In Lemma 6.1, we analyze the complexity of the subroutine. Then, we present one iteration of the preprocessing for CVPP in Algorithm 6.2. In Lemma 6.2, we describe its complexity under Assumption 6.1 on the distribution of the input vectors.

Algorithm 6.1: Reduce one point with list of points

input : $\mathbf{x} \in C^n_{\mathbf{u}}(h, R)$, list $C \subset C^n_{\mathbf{u}}(h, R)$, decrease rate N.

output: $\mathbf{c} \in C$ such that $(\mathbf{v} - \mathbf{c}) \in C^n_{\mathbf{u}}\left(\frac{h}{N}, R\sqrt{2}\right)$ or None.

```
reducePointWithList(x, C, N):
    if C = ∅ then
        return None
    find the first j such that (x − C[j])^t u ⩽ h/N
    for i ∈ {j, ..., min(j + n, |C|)} do
        c ← C[i]
        if ((c − x)^t u > h/N) then
            break
        if (‖π_u(c) − π_u(x)‖ ⩽ R√2) then
            return c
        return None
```

Lemma 6.1. Let $h, R > 0$ and let $N > 1$. Let $\alpha \in \left(0; \frac{1}{2}\right)$. Let C be a list of points,

sampled independently from a $\left(\frac{1}{N}, \alpha\right)$-quasi-uniform distribution on the hypercylinder $C^n_{\mathbf{u}}(h, R)$ and then sorted by the value of their projections on span($\mathbf{u}$). Then, Algorithm 6.1, given as input the list C and a target point $\mathbf{x} \in C^n_{\mathbf{u}}(h, R)$, outputs a vector $\mathbf{c} \in C$, such that $\mathbf{x} - \mathbf{v} \in C^n_{\mathbf{u}}\left(\frac{h}{N}, R\sqrt{2}\right)$, or, outputs None if there is no such a vector in the list C. The expected time complexity of the algorithm is $O(\log(|C|))$.

If there is $\mu > 1$ such that the size of the list C is greater than $\frac{2\mu}{\alpha} \cdot N$, than, for all $\delta \in (0; 1)$, with probability at least $1 - \frac{e^{-\mu}}{\delta}$, for $(1 - \delta)$ fraction of all the possible input points, the algorithm returns a suitable pair from the list C.

Proof. First, estimate the time complexity of the algorithm. When the algorithm searches for a suitable pair in the list C, it looks only through those vectors, that have the projection on the span($\mathbf{u}$) within the distance $\frac{h}{N}$ from the projection on span($\mathbf{u}$) of the target vector $\mathbf{x}$. As the list C is sorted, the first occurrence of such a vector in the list can be found in time $O(\log |C|)$ by the binary search.

Then, if a vector $\mathbf{c} \in C$ with a suitable projection on span($\mathbf{u}$) is found, the algorithm checks whether the angle between the projections $\pi_u(\mathbf{c})$ and $\pi_u(\mathbf{x})$ is less than $\frac{\pi}{2}$. If so, the algorithm outputs $\mathbf{c}$, otherwise it checks the next vector. The algorithm proceeds in such a way until a suitable vector is found, or until the first vector with too big projection on span($\mathbf{u}$) appears.

In order to estimate the running time of the algorithm, we need to estimate the number of vectors, considered by the algorithm. First, if there is no vectors that have a suitable projection on span($\mathbf{u}$), the algorithm immediately returns None and does not spend any time considering vectors from the list. Assume that there are $M > 0$ vectors in $\mathbf{c}$ with a suitable value of the projection on span($\mathbf{u}$): $M = \left|\left\{\mathbf{c} \in C \;\middle|\; |(\mathbf{x} - \mathbf{c})^t \mathbf{u}| \leqslant \frac{h}{N}\right\}\right|$. Then the algorithm considers them subsequently until a vector that has also a suitable projection on span($\mathbf{u}$)$^\perp$ is found. As the projection of vectors from C on span($\mathbf{u}$)$^\perp$ are independent of their projections on span($\mathbf{u}$), the probability that the next vector has the projection on span($\mathbf{u}$)$^\perp$ at the angular distance at most $\frac{\pi}{2}$ from $\pi_{\mathbf{u}}(\mathbf{x})$ is $\frac{1}{2}$ for each new considered vector. Thus, the probability that after considering k vectors from C we have not found a suitable vector is 2^{-k}. Then the expected number of considered vectors can be estimated by:

$$\mathbb{E}\{\text{number of considered vectors}\} = \sum_{k=1}^{M} k \cdot 2^{-k} \leqslant \sum_{k=1}^{\infty} k \cdot 2^{-k} = 2.$$

Then, the expected running time is $O(\log(|C|) + 2) = O(\log(|C|))$.

If the list C has size greater than $\frac{2\mu}{\alpha} \cdot N$, the probability to find a suitable pair for an input point among the vectors from the C can be estimated using Lemma 3.8. □

Further, when we consider Algorithm 6.2, we always use the following assumption on the distribution of its input.

Assumption 6.1. Let $L \subset C^n_{\mathbf{u}}(h, R)$ be the list of lattice vectors that is given as input for Algorithm 6.2. Let $\alpha \in (0; 1)$. Denote as L_r a list of vectors, obtained by rescaling the vectors from L: $L_r := \{\text{Rescale}_{\mathbf{u},h}(\mathbf{x}) \mid \mathbf{x} \in L\}$, We assume that the vectors from the L_r behave as if they were sampled independently from a $(1/N, \alpha)$-quasi-uniform distribution on $C^d_{\mathbf{u}}$.

Lemma 6.2. Let Λ be an n-dimensional lattice. Let $1 < N < 2^n$ be the decrease rate of Algorithm 6.2. Let $L \subset \Lambda \cap C^n_{\mathbf{u}}(h, R)$ be a list of lattice vectors of size $\frac{4n}{\alpha} \cdot N < |L| < \text{poly}(n) \cdot N$. Let Assumption 6.1 hold. Then, Algorithm 6.2, given as input the list L, with probability greater than $1 - e^{-n}(|L| + 1)$, returns two lists of lattice vectors L and C with the following properties:

Algorithm 6.2: One step of preprocessing with cylindrical sieving for CVPP

input : list $L \subset \Lambda \cap C^n_{\mathbf{u}}(h, R)$, decrease rate N.

output: list $L' \subset \Lambda \cap C^n_{\mathbf{u}}\left(\frac{h}{N}, R\sqrt{2}\right)$, list $C \subset L$ such that $|C| = O(nN)$, vectors from C cover $1 - e^{-n}$ fraction of $C^n_{\mathbf{u}}(h, R)$.

```
sievingStepCVP(t, L, N):
    SORT(L)                          ▷ sort vectors by the value of the first coordinate
    C ← ∅; L' ← ∅
    for v ∈ L do
        c ← REDUCEPOINTWITHLIST(v, C, N)
        if (c ≠ None) then
            L' ← L' ∪ {v − c}
        else
            C ← C ∪ {v}
    return (L', C)
```

1. C is a subset of L of size $\frac{4n}{\alpha} \cdot N$, such that points from C cover at least $1 - e^{-n}$ fraction of the hypercylinder $C^n_{\mathbf{u}}(h, R)$;
2. L' is a list of lattice vectors that lie inside the hypercylinder $C^n_{\mathbf{u}}\left(\frac{h}{N}, R\sqrt{2}\right)$, size of the list L' is at least $|L| - \frac{4n}{\alpha} \cdot N$.

The memory complexity of the algorithm is the memory, required to store $O(|L|)$ n-dimensional vectors, the expected time complexity is the time, required to perform $O(n|L|)$ polynomial time operations.

Proof. First, estimate the number of lattice vectors lost during the execution of Algorithm 6.2, i.e., estimate the number of unpaired points that go to the list C. In order to do that, we need to estimate the number of random half-cylinders of height $\frac{h}{N}$, required to cover the cylinder $C^n_{\mathbf{u}}(h, R)$.

By Lemma 3.8, $\frac{4n}{\alpha} \cdot N$ half-cylinders of height $\frac{h}{N}$ with centers at points, sampled from $(1/N, \alpha)$-quasi-uniform distribution, are enough to cover the fraction $(1 - e^{-n})$ of the hypercylinder of height h with probability at least $(1 - e^{-n})$.

Consider the probability of lost of new centers after the size of the list C have reached $\frac{4n}{\alpha} \cdot N$ vectors. Denote as A the event that the fraction $(1 - e^{-n})$ of the hypercylinder is covered after the size of C have reached $\frac{4n}{\alpha} N$ vectors, then, by Lemma 3.8, we get

$$\mathbb{P}\{A\} \geqslant 1 - e^{-n}.$$

Denote as X_i the indicator of the event that the i-th new point, considered by the algorithm after the size of C have reached $\frac{4n}{\alpha} \cdot N$, becomes a center, i.e., the probability that it falls into an uncovered region. Then, for all X_i, we have the following estimate on the conditional probability:

$$(X_i \,|\, A) = \begin{cases} 1, \ p_1 \leqslant e^{-n}, \\ 0, \ p_0 \geqslant 1 - e^{-n}. \end{cases}$$

Denote as M the number of points considered after $|C|$ become greater than $\frac{4n}{\alpha} N$. Consider the random variable $\widehat{X} = \sum\limits_{i=1}^{M} X_i$, which counts the number of points lost in addition to $\frac{4n}{\alpha} N$ centers. By Markov's inequality, we obtain:

$$\mathbb{P}\{\widehat{X} \geqslant 1 \,|\, A\} \leqslant \mathbb{E}(\widehat{X} \,|\, A) = M \cdot \mathbb{E}(X_i \,|\, A) \leqslant M \cdot e^{-n}.$$

Then, as $\widehat{X}$ is a number of lost points and, therefore, it ranges over non-negative integers, we get $\mathbb{P}\{\widehat{X} = 0 \,|\, A\} \geqslant 1 - M \cdot e^{-n}$.

The success of the algorithm is the intersection of the following two events: of the event A that $(1 - e^{-n})$ of the hypercylinder is covered by $\frac{4n}{\alpha}N$ vectors from C, and of the event, that once the number of centers have reached $\frac{4n}{\alpha}N$, algorithm do not produce centers anymore, i.e., $\widehat{X} = 0$. Then, for the probability of success of the algorithm, we obtain:

$$\mathbb{P}\{A \cap (\widehat{X} = 0)\} = \mathbb{P}\{\widehat{X} = 0 \,|\, A\} \cdot \mathbb{P}\{A\} \geqslant (1 - M \cdot e^{-n})(1 - e^{-n}).$$

As $M = |L| - \frac{4n}{\alpha}N \leqslant |L|$, the probability of success is greater than $1 - e^{-n} \cdot (|L| + 1)$.

Then, estimate the time complexity. The time complexity of the algorithm is the sum of the time, required to sort the list L and of the time, needed to process each point from the list: to find a suitable pair or to mark it as a new center.

The time, required to sort the list L by the value of its first coordinate is $\widetilde{O}(|L| \log(|L|)) = \widetilde{O}(n|L|)$, as $|L| \leqslant 2^n$.

Under Assumption 6.1, the expected time needed to process one point is given by Lemma 6.1 and is equal to $O(\log(|C|)) = O(n)$. Then, the overall expected time complexity is $\mathbb{E}\mathcal{T} = |L| \cdot O(n)$. □

6.2 Preprocessing of lattice

In this section, we consider the preprocessing of the lattice,i.e. we describe the algorithm that, given a basis of a lattice Λ, produces the description of the lattice that allows to speed up solving instances of the CVP problem for the given lattice.

We show how the cylindrical sieving can be used to produce such a description of a lattice. The preprocessing algorithm is very similar to the algorithms for solving SVP from the previous chapter. Basically, the preprocessing algorithm runs the cylindrical sieving for solving SVP and stores all the lists of lattice vectors that were produced during the execution of the SVP algorithm. The produced lists of lattice vectors are contained in the hypercylinders with decreasing heights. Then, when the target vector appears, we iteratively reduce it with each list of lattice vectors. The geometry of the vectors from the lists allows to search for a close vectors very efficiently.

The main difference for the analysis of the preprocessing compared to the SVP algorithm, is that now in order to ensure that for most of the targets we can find a close vector using the produced description, for each list we need to guarantee that vectors from the list cover most of the corresponding hypercylinder.

As before, in the case of a lattice with a prime volume, we have slightly more efficient algorithm, so we consider prime volume lattices separately. We start with the preprocessing for a prime volume lattice, and then we describe the general case.

6.2.1 Preprocessing of lattice with prime volume

Lemma 6.3. Let Λ be an n-dimensional lattice with a prime volume. Let $R_0 = \sqrt{0.084(n-1)}$. Let Assumption 6.1 hold. Then, Algorithm 6.3, given as input a basis $\mathbf{B}$ of the lattice Λ, in time $\widetilde{O}(2^{n/2})$, with probability at least $1 - 2^{-0.94n+o(n)}$, outputs a set of $k = \left\lceil \frac{2}{n} \log(\mathrm{vol}(\Lambda)) \right\rceil$ lists of lattice vectors $C = \{C_1, \ldots, C_k\}$ such that for all $i \in \{1, \ldots, k\}$, the half-cylinders of height $\dfrac{\mathrm{vol}(\Lambda)}{2^{n(i+1)/2}}$, centered at the vectors from the list C_i, cover at least the fraction $(1 - e^{-n})$ of the hypercylinder $C^d_{\mathbf{u}}\left(\frac{\mathrm{vol}(\Lambda)}{2^{ni/2}}, R_0 \cdot 2^{k/2}\right)$.

The time complexity of the algorithm is equal to $\widetilde{O}(\log(\mathrm{vol}(\Lambda))^2 \cdot 2^{n/2})$, the memory complexity is $\widetilde{O}(\log(\mathrm{vol}(\Lambda)) \cdot 2^{n/2})$. The memory, required to store the outputted set C is $\widetilde{O}(\log(\mathrm{vol}(\Lambda) \cdot 2^{n/2}))$.

Algorithm 6.3: Preprocessing for CVP for lattices with prime volume

input : a basis $\mathbf{B}$ of an n-dimensional lattice Λ with a prime volume $\mathrm{vol}(\Lambda) = p$.
output: set $C = \{C_1, \ldots, C_k\}$ of lists of lattice points such that
$C_i \in \Lambda \cap C_{\mathbf{u}}^n\left(\frac{p}{2^{ni/2}}, R \cdot 2^{i/2}\right)$, where $\mathbf{u} = (1, 0, \ldots, 0)^t$,
$|C_i| = O(nN)$ for all $i \in \{1, \ldots, k\}$.

```
preprocessingPrimeVolume(B):
    /* initial generation of lattice vectors */
    (p, a) ← HNF(B)
    L_0 ← enumerateCylinder(p, a, R_0 = sqrt(0.084(n − 1)))        ▷ Algorithm 4.1
    C ← ∅
    k ← ⌈(2/n) log(p)⌉
    /* sieving part */
    for i ∈ {1, ..., k} do
        (L_{i+1}, C_i) ← sievingStepCVP(L_i, 2^{n/2})              ▷ Algorithm 6.2
        C ← C ∪ {C_i}
return C
```

Proof. By Lemma 4.1, the algorithm starts with generating a list L_0 of $2^{n/2+o(n)}$ lattice vectors that lie inside a hypercylinder of height $h_0 = p = \mathrm{vol}(\Lambda)$ and of radius $R_0 = \sqrt{0.084(n-1)}$. Then, Algorithm 6.3 iteratively applies the cylindrical sieving, as it is described in Algorithm 6.2, to the generated list of lattice vectors.

We suppose that Assumption 6.1 holds at each iteration of the preprocessing, i.e., there is a constant $\alpha \in (0; 1)$ such that at each iteration, after rescaling, vectors from the list L_i behave like independently sampled from the $(1/N, \alpha)$-quasi-uniform distribution. Then, we can use Lemma 6.2 to describe the evolution of the processed list of lattice vectors. Thus, with probability at least $(1 - e^{-n}(|L_i| + 1))$, after the i-th iteration, the parameters of the hypercylinder and the size of the list change in the following way:

$$h_{i+1} = \frac{h_i}{N}, \quad R_{i+1} = R_i\sqrt{2}, \quad |L_{i+1}| = |L_i| - \frac{4n}{\alpha}N. \tag{6.1}$$

Also, by Lemma 6.2, the size of the list $|C_i|$ is equal to $\frac{4n}{\alpha}N$ for all i.

Our goal is to obtain in the end a list of vectors such that their length is close to the Minkowski's bound of the lattice. In order to ensure that, we require that the hypercylinder that contains the list L_k has the parameters $h_k = \sqrt{c}\,\mathrm{vol}(\Lambda)^{1/n}$, $R_k = \sqrt{c(n-1)}\,\mathrm{vol}(\Lambda)^{1/n}$, where c is some positive constant. Then, proceeding similarly as in the proof of Theorem 4.1, we obtain the following values of the number of iterations k and of the decrease rate of the height of the hypercylinder N:

$$k = \left\lceil \frac{2}{n}\log(\mathrm{vol}(\Lambda)) \right\rceil, \quad N = 2^{n/2}. \tag{6.2}$$

Also, choosing that value of k implies that $c = 2\left(\frac{R_0}{\sqrt{n-1}}\right)^2$. As at each iteration $8nN$ vectors become centers, in order to get a non-empty list in the end, we need to take the list L_0 of size at least $k \cdot \frac{4n}{\alpha}N = O(\log(\mathrm{vol}(\Lambda)) \cdot 2^{n/2})$.

The time complexity of the algorithm is the sum of the time, needed to generate the initial list of lattice vectors L_0, and of the time, required to perform k iterations of the cylindrical sieving. The initial generation of vectors, by Lemma 4.1, takes $\widetilde{O}(|L_0|)$ time.

The sieving part of the time complexity is given by:

$$\mathcal{T}(\text{sieving part}) = \sum_{i=1}^{k} O(n|L_i|) = O\Big(n \cdot \sum_{i=1}^{k} |L_0| - \frac{4n}{\alpha} Ni\Big) = O(N \log(\mathrm{vol}(\Lambda))^2) = \widetilde{O}(2^{n/2}). \tag{6.3}$$

The probability that the algorithm outputs the set C of lists of lattice vectors with the desired properties is equal to the product of probabilities that at each iteration the cylindrical sieving is successful, i.e., the number of centers, lost at each iteration, does not exceed $\frac{4n}{\alpha}N$. Then, by Lemma 6.2, the overall probability of success is

$$\prod_{i=1}^{k}(1 - e^{-n}(|L_i| + 1)) \geqslant (1 - e^{-n}(|L_0| + 1))^k \geqslant 1 - k \cdot e^{-n} \cdot 2^{n/2 + o(n)} = 1 - 2^{0.94n + o(n)}.$$

□

6.2.2 Preprocessing of any lattice

Algorithm 6.4: Preprocessing for CVPP for any lattice

input : a basis $\mathbf{B}$ of an n-dimensional lattice Λ, parameter q of the LLL reduction

output: set $C = \{C_1, \dots, C_k\}$ of lists of lattice points such that

$C_i \in \Lambda \cap C_{\mathbf{u}}^n\Big(\frac{h}{2^{ni/2}}, R \cdot 2^{i/2}\Big)$, $|C_i| = O(nN)$ for all $i \in \{1, \dots, k\}$, where

$k = \lceil (n-1)\log(q) \rceil$, $h = q^{\frac{(n-1)^2}{2}} \cdot n\,\mathrm{vol}(\Lambda)^{1/n}$, $R = \sqrt{\frac{n-1}{\pi e}} q^{-\frac{n-1}{2}} \mathrm{vol}(\Lambda)^{1/n}$.

```
preprocessingAnyLattice(B, q):
    /* initial generation of lattice vectors */
    L_0 ← ENUMERATECYLINDER(B, q)                          ▷ Algorithm 4.2
    C ← ∅
    k ← ⌈(n − 1) log(q)⌉
    /* sieving part */
    for i ∈ {1, ..., k} do
        (L_{i+1}, C_i) ← SIEVINGSTEPCVP(L_i, 2^{n/2})      ▷ Algorithm 6.2
        C ← C ∪ {C_i}
return C
```

Lemma 6.4. Let Λ be an n-dimensional lattice, let $R_0 = \sqrt{\frac{n-1}{\pi e}} q^{-\frac{n-1}{2}} \mathrm{vol}(\Lambda)^{1/n}$, and $h_0 = q^{\frac{(n-1)^2}{2}} n\,\mathrm{vol}(\Lambda)^{1/n}$. Let Assumption 6.1 hold. Then, Algorithm 6.4, given as input a basis $\mathbf{B}$ of the lattice Λ, in time $\widetilde{O}(2^{n/2})$, with probability at least $1 - 2^{-0.94n + o(n)}$, outputs a set of $k = \lceil (n-1)\log(q) \rceil$ lists of lattice vectors $C = \{C_1, \dots, C_k\}$ such that for all $i \in \{1, \dots, k\}$, the half-cylinders of height $h_0/2^{n(i+1)/2}$, centered at the vectors from the list C_i, cover at least the fraction $(1 - e^{-n})$ of the hypercylinder $C_{\mathbf{u}}^d\Big(\frac{h_0}{2^{ni/2}}, R_0 \cdot 2^{k/2}\Big)$.

The time complexity of the algorithm is equal to $\widetilde{O}(2^{0.531n})$, the memory complexity is $\widetilde{O}(2^{n/2})$. The time, required to perform the sieving part is $\widetilde{O}(2^{n/2})$. The memory, required to store the outputted set C is the memory, required to store $O(n^2 \cdot 2^{n/2})$ n-dimensional vectors.

Proof. Algorithm 6.4, as Algorithm 6.3, has two parts: the initial generation of lattice vectors inside a hypercylinder and the iterative sieving part. The sieving part is essentially the same as the sieving part of Algorithm 6.3, but the initial generation of lattice vectors

is different: in both cases, we enumerate lattice vectors inside a hypercylinder, but we use different algorithms and we get the hypercylinders with different parameters in the end. In both algorithms, the goal is to obtain a set of lists of lattice vectors, such that the last list consists of vectors with a norm close to Minkowski's bound. Since the parameters of the initial hypercylinders that contain the list L_0 are not the same for Algorithms 6.3 and 6.4, the number of iteration of the sieving part also changes.

In Algorithm 6.4, for initial generation of lattice vectors, we use Algorithm 4.2, based on the unbalanced lattice reduction and Schnorr-Euchner's enumeration of an n-dimensional quasi-orthonormal sublattice. Then, by Lemma 4.2, after the initial generation part, we obtain a list L_0 of $\widetilde{O}(\beta^n)$ lattice points inside a hypercylinder of height h_0 and radius R_0:

$$h_0 := q^{\frac{(n-1)^2}{2}} n \operatorname{vol}(\Lambda)^{1/n}, \quad R_0 := \beta\sqrt{\frac{n}{2\pi e}} q^{-\frac{n-1}{2}} \operatorname{vol}(\Lambda)^{1/n}, \tag{6.4}$$

where β is the constant that we choose later, depending on how many lattice vectors we need for the algorithm to proceed, and $q > \sqrt{\frac{4}{3}}$ is the parameter of the lattice reduction used by Algorithm 4.2. As the desired length of the vectors from the lists L_k and C_k should be at most $c\sqrt{n}\operatorname{vol}(\Lambda)^{1/n}$, we require that the height h_k and the radius R_k of the k-th hypercylinder satisfy

$$h_k = \frac{h_0}{N^k} \leqslant c \operatorname{vol}(\Lambda)^{1/n}, \quad R_k = R_0 \cdot 2^{k/2} \leqslant c\sqrt{n-1} \operatorname{vol}(\Lambda)^{1/n}, \tag{6.5}$$

where N is the decrease rate of one iteration of the cylindrical sieving. Then, using (6.4) and (6.5), we obtain the following values of the decrease rate N and of the number of iterations k:

$$k = \lceil (n-1)\log(q) \rceil, \quad N = 2^{n/2}. \tag{6.6}$$

Also, taking $k = \lceil (n-1)\log(q) \rceil$ implies the following bound on the ratio of c and β: $\frac{c}{\beta} \geqslant \sqrt{\frac{1}{\pi e}}$ (see the proof of Algorithm 4.4 for similar computations).

Then, by Lemma 6.2, at each iteration, $\frac{4n}{\alpha}N$ vectors from the processed list become centers. Therefore, in order to have a non-empty list after $k = O(n)$ iterations, we need the initial list of $\frac{4n}{\alpha}N \cdot k = O(n^2 \cdot 2^{n/2})$ vectors. In order to obtain the initial list of lattice vectors of the required size, we need to take $\beta = \sqrt{2}$, which implies that the time complexity of the initial sampling is equal to

$$\mathscr{T}(\text{initial generation}) = \widetilde{O}(2^{0.531n})$$

and that the constant α should be at least $\sqrt{\frac{2}{\pi e}}$.

Consider the time complexity of the sieving part. By Lemma 6.2, the time complexity of the i-th iteration of sieving is equal to $O(n|L_i|)$ polynomial time operations. Then, the overall time complexity of $k = O(n)$ iterations of sieving is given by

$$\mathscr{T}(\text{sieving}) = \widetilde{O}\Big(\sum_{i=1}^{k}(|L_0| - \frac{4n}{\alpha}Ni)\Big) = \widetilde{O}(n^4 N).$$

The probability of success of the algorithm can be computed in the same way as in Lemma 6.3. □

6.3 Decoding for targets inside hypercylinder

Let $\mathbf{u}$ be an n-dimensional unit vector, let h and R be two positive real numbers such that $h >> R$. Let $C_{\mathbf{u}}^n(h, R)$ be the n-dimensional hypercylinder of height h and radius

R. Let Λ be an n-dimensional lattice. Assume that we are given k lists of lattice points $C_1, \dots, C_k \subset \Lambda$ such that the points from the i-th list cover most of the surface of the hypercylinder $C^n_{\mathbf{u}}(h/N^i, R \cdot 2^{i/2})$ with half-cylinders of length $\frac{h}{N^{i+1}}$.

In this section, we describe an algorithm that, given a target point $\mathbf{t} \in C_{\mathbf{u}}(h, R)$ from the first hypercylinder as input, find a lattice point $\mathbf{v} \in \Lambda$ such that the difference $\mathbf{v} - \mathbf{t}$ belongs to the last hypercylinder $C^n_{\mathbf{u}}(h/N^k, R \cdot 2^{k/2})$, using the lists $C_1, \dots, C_k$.

First, we show that an algorithm that solves the problem for $k = 1$ can be turned into an algorithm that solves the problem for any k. Assume that we can solve the problem when $k = 1$ for any reasonable pair of the parameters h and R. That is, there is an algorithm A that, given a target $\mathbf{t} \in C^n_{\mathbf{u}}(h, R)$ and a list $C_1 \subset C^n_{\mathbf{u}}(h, R) \cap \Lambda$, finds a lattice point $\mathbf{v}_1 \in \Lambda$ such that the difference $\mathbf{t} - \mathbf{v}$ belongs to the hypercylinder $C^n_{\mathbf{u}}(h/N, R\sqrt{2})$. Then, if we have a coverage $C_2 \subset \Lambda$ for the second hypercylinder $C^n_{\mathbf{u}}(n/N, R\sqrt{2})$, we can also solve a problem for $k = 2$: we apply the algorithm A to the input that consists of the list C_2 and the new target $\mathbf{t} - \mathbf{v}_1$. Let $\mathbf{v}_2 \in \Lambda$ be a pair, found by the algorithm for the target $\mathbf{t} - \mathbf{v}_1$. Then, if we have k lists $C_1, \dots, C_k$, we can perform k iterations:

$$\text{Input: } \mathbf{t} \in C^n_{\mathbf{u}}(h, R),\ C_1 \subset \Lambda \cap C^n_{\mathbf{u}}(h, R); \tag{6.7}$$

$$\text{Iteration 1: } \mathbf{v}_1 \leftarrow A(\mathbf{t}, C_1), \mathbf{v}_1 \in \Lambda,\ (\mathbf{t} - \mathbf{v}_1) \subset C^n_{\mathbf{u}}(h/N, R\sqrt{2}); \tag{6.8}$$

$$\text{Iteration 2: } \mathbf{v}_2 \leftarrow A(\mathbf{t} - \mathbf{v}_1, C_2),\ \mathbf{v}_2 \in \Lambda,\ (\mathbf{t} - \mathbf{v}_1 - \mathbf{v}_2) \subset C^n_{\mathbf{u}}(h/N^2, 2R); \tag{6.9}$$

$$\vdots \tag{6.10}$$

$$\text{Iteration } k: \mathbf{v}_k \leftarrow A\Big(\mathbf{t} - \sum_{i=1}^{k-1} \mathbf{v}_i, C_k\Big),\ \mathbf{v}_k \in \Lambda,\ \Big(\mathbf{t} - \sum_{i=1}^{k} \mathbf{v}_i\Big) \subset C^n_{\mathbf{u}}(h/N^k, R \cdot 2^{k/2}); \tag{6.11}$$

After k iterations, we get a vector $\mathbf{t}_k := \mathbf{t} - \sum_{i=1}^{k} \mathbf{v}_i$ that lies inside the hypercylinder $C^n_{\mathbf{u}}(h/N^k, R \cdot 2^{k/2})$. Subtracting the initial target $\mathbf{t}$ from $\mathbf{t}_k$, we get the desired lattice vector $\mathbf{v} = \sum_{i=1}^{k} \mathbf{v}_i \in \Lambda$.

Now, the goal is to construct an algorithm A that can solve the problem for $k = 1$. The algorithm A takes as input a point $\mathbf{t} \in C^n_{\mathbf{u}}(h, R)$ and a list of lattice points $C_1 \subset \Lambda \cap C^n_{\mathbf{u}}(h, R)$ such that points from the list C_1 cover most of the surface of $C^n_{\mathbf{u}}(h, R)$ with half-cylinders of length h/N. The algorithm should find a lattice point $\mathbf{v} \in \Lambda$ such that the difference $\mathbf{t} - \mathbf{v}$ satisfies the following two conditions:

1. the projection of the difference on the hypercylinder axis is bounded by h/N:

$$|(\mathbf{t} - \mathbf{v})^t \mathbf{u}| < \frac{1}{2} \cdot \frac{h}{N}; \tag{6.12}$$

2. the projection of the difference on the subspace, orthogonal to the cylinder axis is bounded by $R\sqrt{2}$:

$$\|\pi_{\mathbf{u}}(\mathbf{t} - \mathbf{v})\| \leqslant R\sqrt{2}. \tag{6.13}$$

Recall that in Section 6.1, while describing how one step of the preprocessing works, we used the subroutine that solves a very similar problem (see Algorithm 6.1). In Algorithm 6.1, we are given a target point $\mathbf{x} \in C^n_{\mathbf{u}}(h, R)$ and a list of points $C \subset C^n_{\mathbf{u}}(h, R)$ that are sampled independently from a certain uniform-like distribution and then sorted by the value of the projection of the vectors on the cylinder axis. The goal there was to find a vector $\mathbf{c} \in C$ such that the difference $\mathbf{x} - \mathbf{c}$ satisfies the same conditions as the difference $\mathbf{t} - \mathbf{v}$ (see (6.12) and (6.13)).

Algorithm 6.1 has two steps. First, it finds a position $j \in \{0, \dots, |C| - 1\}$ of the first vector in the sorted list C such that the difference $\mathbf{x} - C[i]$ satisfies the first condition given by (6.12). Then the algorithm checks the two conditions for all the vectors in the list C starting from the j-th vector, until one of the two events happens:

1. the algorithm founds a vector that satisfies both conditions,i.e., the desired pair for the target $\mathbf{x}$ is found;
2. the algorithm finds a first vector that does not satisfy the first condition, i.e., is no suitable pair for $\mathbf{x}$ in the list C.

Algorithm 6.1 ensures the same conditions as we need in the current settings, so we can use it to find a suitable pair for the target $\mathbf{t}$. Then, the decoding algorithm consists in iterative application of Algorithm 6.1 to a target. The decoding algorithm is summarized in Algorithm 6.5.

Algorithm 6.5: Decoding algorithm

input : preprocessing of an n-dimensional lattice Λ:
$C = \{C_1 \subset C^n_{\mathbf{u}}(h_1, R_1), \dots, C_k \subset C^n_{\mathbf{u}}(h_k, R_k)\} \subset \Lambda$, where $h_i = \frac{h}{N^{i-1}}$,
$R_i = R_1 \cdot 2^{\frac{i-1}{2}}$ for all i, a target vector $\mathbf{t} \in C^n_{\mathbf{u}}(h_1, R_1)$.

output: $\mathbf{v} \in \Lambda$ such that $\|\mathbf{v} - \mathbf{t}\| \leqslant \sqrt{h_k^2 + R_k^2}$

```
decode(C, t):
    v ← 0
    for i ∈ {1, ..., k} do
        v_i ← REDUCEPOINTWITHLIST(t, C, N)          ▷ Algorithm 6.1
        if (v_i ≠ None) then
            v ← v + v_i
            t ← t − v_i
        else
            return v
    return v
```

However, the analysis of the complexity and of the probability of success of Algorithm 6.1 is different in the new settings, because now an input list and a target point have different properties compared to the ones described in Section 6.1.

Recall in short the analysis of the complexity of Algorithm 6.1. The search for the position j can be done in $O(\log(|C|))$ time, as the list C is sorted by the length of the projection of the vectors on the cylinder axis. We also showed in Lemma 6.1 that, on average, the algorithm needs to check only a constant number of vectors until the desired pair for the target is found. We can get such an estimate because we assume that the projections of the vectors from the list C on the subspace orthogonal to $\mathbf{u}$, after rescaling, behave as if they are sampled from the uniform distribution on S^{n-2}.

Consider the difference between the current input and the input of Algorithm 6.1, described in Section 6.1. In Section 6.1, we did not require anything for the target point, except that the target point should belong to the hypercylinder. For the list C, we required that the vectors from the list should behave like sampled independently from a $(1/N, \alpha)$-quasi-uniform distribution, where $\alpha \in (0; 1/2)$ is some constant.

Now, we do not require any specific distribution for the list C_1. The only requirement for C_1 is that the points from C_1 should cover most of the surface of the cylinder. But since we do not have any requirements on the distribution of the vectors from C_1, we need to add some assumption on the target point $\mathbf{t}$. First, because if the point $\mathbf{t}$ falls into uncovered region, there is no suitable pair for it in the list C_1. Second, if we do not have

any assumption on the distribution of $\mathbf{t}$, it is hard to estimate how many points in the list C the algorithm needs to check until the pair is found.

Therefore, for the analysis of the decoding algorithm, we assume that the target point, given as input to Algorithm 6.1, behaves as sampled from the uniform distribution on the hypercylinder with the corresponding parameters (see Assumption 6.2).

Assumption 6.2. Let h, N, k, and $\mathbf{u}$ be as in Algorithm 6.5. We assume that for all $i \in \{1, \dots, k\}$, at i-th iteration of Algorithm 6.5, the rescaled target vector $\mathrm{Rescale}_{\mathbf{u},h_1/N^i}(\mathbf{t})$ behaves like it is sampled from the uniform distribution on $C^d_{\mathbf{u}}$.

The differences between the inputs for Algorithm 6.1, when it is used as a subroutine for the preprocessing and for the decoding, are summarized in Table 6.1.

	preprocessing	**decoding**
points from list C	cover $(1-\varepsilon)$ fraction of the hypercylinder $C^n_{\mathbf{u}}(h, R)$	sampled independently from $(1/N, \alpha)$-quasi-uniform distribution
target $\mathbf{t}$	any point in $C^n_{\mathbf{u}}(h, R) \cap \Lambda$	uniformly distributed in $C^n_{\mathbf{u}}(h, R)$

Table 6.1 – Comparison of the input properties of Algorithm 6.1 for the preprocessing and for the decoding.

The complexity and the probability of success of the decoding procedure under Assumption 6.2 are analyzed in Lemma 6.5.

Lemma 6.5. Let Λ be an n-dimensional lattice. Let k be a positive integer number and let $N > 1$. Let $\varepsilon \in (0; 1)$. Let $\mathbf{u} \in \mathbb{R}^n$ be a unit vector. Let $R, h > 0$ and, for all $i \in \{1, \dots, k\}$, let $R_i = R \cdot 2^{(i-1)/2}$, $h_i = h/N^{i-1}$. Let $C = \{C_1, \dots, C_k\}$ be a set of k lists of lattice vectors such that for all $i \in \{1, \dots, k\}$, the list C_i satisfies the following three properties:

1. the vectors in the list C_i are sorted by the value of their projection on $\mathrm{span}(\mathbf{u})$;
2. half-cylinders of height $\frac{h_i}{N}$, centered at the vectors from the list C_i, cover the $(1-\varepsilon)$-fraction of the hypercylinder $C_{\mathbf{u}}(h_i, R_i)$;
3. for any pair of vectors $\mathbf{x}, \mathbf{y} \in C_i$, at least one of the two conditions holds:
   (a) $|(\mathbf{x}-\mathbf{y})^t\mathbf{u}| > \frac{h_i}{N}$;
   (b) $\|\pi_{\mathbf{u}}(\mathbf{x}-\mathbf{y})\| > R_i\sqrt{2}$.

Let Assumption 6.2 hold. Then, Algorithm 6.5, given as input a target point $\mathbf{t} \in C_{\mathbf{u}}(h_1, R_1)$ and the set C, with probability at least $(1-\varepsilon)^k$, outputs a lattice vector $\mathbf{v} \in \Lambda$, such that $\|\mathbf{t}-\mathbf{v}\| \leqslant \sqrt{h_k^2 + R_k^2}$. The expected time complexity of the algorithm is

$$\sum_{i=1}^{k} O(\log(|C_i|)). \tag{6.14}$$

Proof. Under Assumption 6.2, since at each iteration $(1-\varepsilon)$-fraction of the corresponding hypercylinder is covered by points from C_i, the probability to find a suitable pair at one iteration is $1-\varepsilon$. Then, as we assume that iterations are independent, the probability to find a pair at each iteration is $(1-\varepsilon)^k$.

Denote the target point, given to the algorithm as input, as $\mathbf{t}_0$. Then, after k iterations, if at each iteration the pair was successfully found, we obtain

$$\mathbf{t} = \mathbf{t}_0 - \sum_{i=1}^{k} \mathbf{v}_i \in C^n_{\mathbf{u}}(h \cdot N^{-k}, R \cdot 2^{k/2})$$

and $\mathbf{v} = \sum_{i=1}^{k} \mathbf{v}_i$ (see Lines 6-7 of Algorithm 6.5). The algorithm outputs $\mathbf{v}$. It is a lattice vector, as all $\mathbf{v}_i$ are lattice vectors. The difference $\mathbf{t}_0 - \mathbf{v}$ belongs to the hypercylinder $C^n_{\mathbf{u}}(h \cdot N^{-k}, R \cdot 2^{k/2})$, so its norm is bounded by $\sqrt{(\frac{h}{N^k})^2 + R^2 \cdot 2^k}$.

The time complexity of the algorithm is the complexity of reducing a point with each of the k lists. Under Assumption 6.2, the time complexity of the i-th iteration is the time complexity of Algorithm 6.1, applied to a target point $\mathbf{t}$ from the uniform distribution on $C^n_{\mathbf{u}}(h_i, R_i)$ and to the set C_i. The goal of the algorithm is to find a vector $\mathbf{c} \in C_i$ such that the difference $\mathbf{t} - \mathbf{c}$ satisfies:

1. $|(\mathbf{t} - \mathbf{c})^t \mathbf{u}| \leqslant \frac{h_i}{N}$;
2. $\|\pi_{\mathbf{u}}(\mathbf{t} - \mathbf{c})\| \leqslant R_i\sqrt{2}$.

Algorithm 6.1 starts with finding the first vector in the sorted list C_i that satisfies the first condition. It is done using the binary search and takes $O(|C_i|)$ time.

If such a vector is found, Algorithm 6.1 checks whether the second condition is also satisfied. If so, then the desired vector is found. Otherwise, the algorithm moves to the next vector in the list and checks both conditions. Algorithm proceeds in such a way until the vector, that satisfies both conditions is found or until it meets the first vector that does not satisfy the first condition.

To estimate the time complexity, we need an upper bound for the expectation of the number of vectors, considered by the algorithm until the desired pair for $\mathbf{t}$ is found. Assume that there are M vectors in C_i that satisfy the first assumption. Denote their rescaled projections on $\mathrm{span}(\mathbf{u})^{\perp}$ as $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_M$. Under Assumption 6.2, $\mathbf{t}$ is uniformly distributed, then $\widehat{\mathbf{t}} = \pi_{\mathbf{u}}(\mathbf{t})/\|\pi_{\mathbf{u}}(\mathbf{t})\|$ is uniformly distributed on S^{n-2}. If $\widehat{\mathbf{t}}$ and $\mathbf{c}_1$ appear in the same hemisphere, the second condition for $\mathbf{c}_1$ is satisfied and the desired point is found. Then, the probability that the first considered point satisfies the second condition is at least $\frac{1}{2}$.

In order to estimate the expectation of the number of considered points, we need to estimate the probability that the desired pair is found after checking m points, i.e., the probability that the first $m-1$ points do not satisfy the second condition, while $\mathbf{c}_m$ satisfies. In other words, it is the probability that for all $i \in \{1, \ldots, m-1\}$, the vectors $\mathbf{c}_i$ and $\mathbf{t}$ lie in different hemispheres while $\mathbf{c}_m$ and $\mathbf{t}$ are in the same hemisphere.

For any $i, j \in \{1, \ldots, M\}$, $i \neq j$, the angle between $\mathbf{c}_i$ and $\mathbf{c}_j$ is at least $\pi/2$ (see Property 3 from the statement of the lemma). Then, $\mathbf{c}_1$ covers a half of the sphere S^{n-2}, $\mathbf{c}_2$ covers at least a half of sphere's surface non-covered by $\mathbf{c}_1$. More generally, any subset D of the set $\{\mathbf{c}_1, \ldots, \mathbf{c}_M\}$ of size k cover at least $(1 - 2^{-k})$-fraction of the sphere, and any $\mathbf{c}_i \notin D$ covers at least a half of the surface of the sphere, uncovered by D. Therefore, the probability that the desired pair is found after considering m points is at most 2^{-m}. Then, the expected number of points, considered by the algorithm is bounded:

$$\mathbb{E}\{\text{number of checked points}\} \leqslant \sum_{m=1}^{M} m \cdot 2^{-m} \leqslant \sum_{m=1}^{\infty} m \cdot 2^{-m} = 2.$$

The expected time complexity of k iterations of the algorithm is bounded by:

$$\sum_{i=1}^{k} (O(\log(|C_i|)) + 2) = \sum_{i=1}^{k} O(\log |C_i|).$$

□

6.4 Solving CVPP in polynomial time.

Solving CVPP has two parts: the preprocessing part and the decoding part. In Section 6.2, we described how a lattice can be preprocessed using the cylindrical sieving.

In Section 6.3, we presented the decoding algorithm (see Algorithm 6.5) that works for the target points that are uniformly distributed on a certain n-dimensional hypercylinder. The parameters of the hypercylinder, accepted by Algorithm 6.5 depend on the description of a lattice that is given to the decoding algorithm.

In this chapter, we put all the pieces together and obtain an algorithm, that heuristically solves the Closest Vector Problem with Preprocessing in polynomial time. Namely, we describe a modification of Algorithm 6.5 that can handle any target point from $\mathbb{R}^n$.

Let Λ be an n-dimensional lattice. Let $N = 2^{n/2}$, $k = \text{poly}(n)$, $h = O(\sqrt{n}) \cdot \text{vol}(\Lambda)^{1/n} \cdot 2^{nk/2}$, and $R = O(\sqrt{n}) \cdot \text{vol}(\Lambda)^{1/n} \cdot 2^{-k/2}$. Let $\mathbf{u} \in \mathbb{R}^n$ be a unit vector. Algorithm 6.5, by Lemma 6.5, takes as input a description C of the lattice Λ and a target point $\mathbf{x}$. The description $C = \{C_1, \ldots, C_k\}$ is a set of lists of lattice points. By Lemma 6.5, in order to form a correct input for Algorithm 6.5, the pair $(C, \mathbf{x})$ should satisfy:

1. for all $i \in \{1, \ldots, k\}$, the half-cylinders of height $\frac{h_i}{N}$, centered at the vectors from C_i, cover most of the surface of a hypercylinder $C^n_{\mathbf{u}}(h_i, R_i)$, where $h_i = \frac{h_i}{N^{i-1}}$, $R_i = R \cdot 2^{\frac{i-1}{2}}$;
2. the target point $\mathbf{t}$ should belong to the first hypercylinder, covered by the vectors from C:
$$\mathbf{t} \in C^n_{\mathbf{u}}(h, R); \tag{6.15}$$
3. the target point $\mathbf{t}$ should behave as sampled from the uniform distribution on $C^n_{\mathbf{u}}(h, R)$.

In order to construct a decoding algorithm for solving CVPP, we need to prepare an input for Algorithm 6.5 that satisfies the conditions, listed above. The first condition is on the description of the lattice. In Section 6.2, we showed that a description that satisfies this condition can be obtained using cylindrical sieving in time $2^{O(n)}$.

The goal of this section is to construct a polynomial-time algorithm, that transforms an arbitrary target point $\mathbf{t} \in \mathbb{R}^n$ into a target point $\mathbf{x}$ that satisfies conditions 4 and 5. That is, after the transformation, the vector $\mathbf{x}$ should behave as sampled from the uniform distribution on the hypercylinder with certain parameters.

The parameters of the hypercylinder that should contain $\mathbf{x}$ depend on the available description C of the given lattice. The lattice descriptions have different parameters in case of a prime volume lattice and in the general case, so the algorithms for preparing the target point are also slightly different. First, we describe the general idea which is similar in both cases, then, in Section 6.4.2, we describe solving CVPP for integer lattices with a prime volume, and, finally, in Section 6.4.3, we describe solving CVPP for an arbitrary lattice.

6.4.1 Transformation of target vector: general idea

In this section, we consider the following problem. We are given a target point $\mathbf{t} \in \mathbb{R}^n$ and a basis $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_n\}$ of a lattice Λ that satisfies:

1. the first vector of the basis is bounded: $\|\mathbf{c}_1\| \leqslant h$;
2. for all $i \in \{2, \ldots, n\}$, $\|\mathbf{c}^*_i\| \leqslant \dfrac{R}{\sqrt{n-1}}$.

Denote $\frac{1}{\|\mathbf{c}_1\|} \cdot \mathbf{c}_1$ as $\mathbf{u}$. The goal is to construct an algorithm that, given the vector $\mathbf{t}$ and the basis $\mathbf{C}$ as input, finds a vector $\mathbf{x}'$ such that

1. $\mathbf{x}'$ belongs to $C^n_{\mathbf{u}}(h, R)$;
2. $\mathbf{x}'$ behaves like it is sampled from the uniform distribution on $C^n_{\mathbf{u}}(h, R)$;
3. if we are able to approximate $\mathbf{x}'$ by a lattice vector, then we can use this approximation to find a close lattice vector to $\mathbf{t}$.

If the first two conditions are satisfied, we can find a lattice vector that is close to $\mathbf{x}'$ using Algorithm 6.5.

The first condition is easy to satisfy. Using the nearest plane algorithm, we can reduce the target vector $\mathbf{t}$ with the basis $\mathbf{C}$. Thus we get a lattice vector $\mathbf{v}_0 \in \Lambda$ such that the difference $\mathbf{t} - \mathbf{v}_0$ belongs to the hypercylinder $C^n_{\mathbf{u}}(h, R)$.

The third condition is also easy to satisfy, if the only operations, performed on the target, are adding or subtracting lattice vectors. Assume that $\mathbf{x}' = \mathbf{t} + \mathbf{y}$, where $\mathbf{y} \in \Lambda$ and we can approximate $\mathbf{x}'$ by a lattice point, i.e., we are able to find a vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{x}' - \mathbf{v}\| \leqslant r$ for some small $r > 0$. Then, $(\mathbf{v} - \mathbf{y}) \in \Lambda$ is a lattice vector that approximates the initial target $\mathbf{t}$: $\|\mathbf{t} - (\mathbf{v} - \mathbf{y})\| = \|\mathbf{x}' - \mathbf{v}\| \leqslant r$.

To satisfy the second condition, we need to add some randomness to the target point. Assume that we are able to use the basis $\mathbf{C}$ to sample a random lattice point $\mathbf{r}$ from the uniform-like distribution D on the hypercylinder $C^n_{\mathbf{u}}(h, R')$ of the same height h, but of a possibly bigger radius $R' \geqslant R$. Consider the sum of the random vector $\mathbf{r}$ and of the target, reduced with the basis $\mathbf{C}$:

$$\mathbf{x} := \mathbf{t} - \mathbf{v}_0 + \mathbf{r}.$$

The vector $\mathbf{x}$ might not belong to the hypercylinder $C^n_{\mathbf{u}}(h, R)$. The both projections on $\mathrm{span}(\mathbf{u})$ and on $\mathrm{span}(\mathbf{u})^\perp$ may be longer than needed. The length of the projection on $\mathrm{span}(\mathbf{u})$ can be made short enough to fit into the hypercylinder by size-reducing $\mathbf{x}$ with $\mathbf{c}_1$:

$$\mathbf{x} := \mathbf{x} - \lceil \frac{\mathbf{x}^t \mathbf{c}_1}{\mathbf{c}_1^t \mathbf{c}_1} \rfloor \cdot \mathbf{c}_1.$$

The vector $\mathbf{x}$ now has a short enough projection on $\mathrm{span}(\mathbf{u})$, but the projection of $\mathbf{x}$ on $\mathrm{span}(\mathbf{u})^\perp$ is bounded only by $R' + R/2$. Consider $\mathbf{x}'$, obtained by rescaling the projection of $\mathbf{x}$ on $\mathrm{span}(\mathbf{u})^\perp$:

$$\mathbf{x}' \leftarrow \mathbf{x} - \pi_{\mathbf{u}}(\mathbf{x}) \cdot \Big(1 - \frac{R}{R'}\Big).$$

This vector is a good candidate to satisfy the first two conditions: it is inside the desired hypercylinder and it has some randomness inside. Although the last transformation was not adding a lattice vector, if the sampling radius R' is not too big, finding a close lattice point to $\mathbf{x}$ allows to recover a close approximation for $\mathbf{t}$. We describe how it can be done in details in Lemma 6.6. The transformation of the target point is summarized by Algorithm 6.6.

Algorithm 6.6: Transform target point

input : a target vector $\mathbf{t} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^n$ such that $\|\mathbf{u}\| = 1$, $h, R > 0$, a basis $\mathbf{C}$ of Λ.
output: $\mathbf{w} \in \Lambda$ and $\mathbf{x}' \in C_{\mathbf{u}}(h, R)$ such that $\mathbf{t} - \mathbf{w} - \textsc{Decode}(\mathbf{x}')$ is short.

```
transformTarget(t, C):
    v_0 ← NEARESTPLANE(C, t)                          ▷ Algorithm 3.4
    r ← SAMPLED(C_u(h, R') ∩ Λ)
    x ← t − v_0 + r
    x ← x − ⌈(x^t c_1)/(c_1^t c_1)⌋ · c_1
    R' ← ‖π_u(x)‖
    x' ← x − π_u(x) · (1 − R/R')
    return (w := v_0 − r, x')
```

Lemma 6.6. Let $h, R, r > 0$. Let Λ be an n-dimensional lattice. Let $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_n\}$ be a basis of Λ such that $\|\mathbf{c}_1\| \leqslant h$ and, for all $i \in \{2, \ldots, n\}$, $\|\mathbf{c}_i^*\| \leqslant R/\sqrt{n-1}$.

Denote the vector $\frac{1}{\|\mathbf{c}_1\|} \cdot \mathbf{c}_1$ as $\mathbf{u}$. Let $R' \geqslant R$. Let $(\mathbf{w} \in \Lambda, \mathbf{x}' \in C^n_{\mathbf{u}}(h, R))$ be the pair of vectors, returned by Algorithm 6.6 on the input $\mathbf{t}$ and $\mathbf{C}$. Suppose that SAMPLED is an

algorithm that, given $\mathbf{C}$, samples the distribution D on the set $C^n_{\mathbf{u}}(h, R') \cap \Lambda$ such that the distribution of $\text{Rescale}_{\mathbf{u},h}(\mathbf{y})$ is close to the uniform on $C^n_{\mathbf{u}}$ for a vector $\mathbf{y}$ sampled from D.

Then, $\mathbf{x}'$ is uniformly distributed on $C^n_{\mathbf{u}}(h, R)$. If there is a lattice vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{x}' - \mathbf{v}\| \leqslant r$, then the distance from $\mathbf{t}$ to $(\mathbf{w} + \mathbf{v}) \in \Lambda$ is also bounded:

$$\|\mathbf{t} - (\mathbf{w} + \mathbf{v})\| \leqslant R' + r. \tag{6.16}$$

If the time complexity of SAMPLED is $\text{poly}(n)$, then the time complexity of Algorithm 6.6 is also $\text{poly}(n)$.

Proof. Consider the norm of the difference $\mathbf{t}-(\mathbf{w}+\mathbf{v})$. According to Line 8 of Algorithm 6.6, $\mathbf{w} = \mathbf{v}_0 - r$. Then,

$$\|\mathbf{t} - (\mathbf{w} + \mathbf{v})\| = \|\mathbf{t} - \mathbf{v}_0 + \mathbf{r} - \mathbf{v}\|. \tag{6.17}$$

By Line 4 of Algorithm 6.6, $\mathbf{t} - \mathbf{v}_0 + \mathbf{r}$ can be replaced by $\mathbf{x}$. Then, after adding and subtracting the vector $\mathbf{x}'$ and using the triangle inequality, we get:

$$\|\mathbf{t} - (\mathbf{w} + \mathbf{v})\| = \|\mathbf{x} - \mathbf{v}\| \leqslant \|\mathbf{x} - \mathbf{x}'\| + \|\mathbf{x}' - \mathbf{v}\| = \|\mathbf{x} - \mathbf{x}'\| + r. \tag{6.18}$$

Then, to prove that the inequality, given by (6.16), holds for the output of Algorithm 6.6, we need to show that $\|\mathbf{x} - \mathbf{x}'\| \leqslant R'$. By Line 7 of Algorithm 6.6, the norm of $\mathbf{x} - \mathbf{x}'$ is given by

$$\|\mathbf{x} - \mathbf{x}'\| = \left(1 - \frac{R}{R'}\right) \cdot \|\pi_{\mathbf{u}}(\mathbf{x})\|. \tag{6.19}$$

Hence, consider the norm of the projection of $\mathbf{x}$ on $\text{span}(\mathbf{u})^{\perp} = \text{span}(\mathbf{c}_1)^{\perp}$. By Line 4 of Algorithm 6.6, $\mathbf{x}$ is the sum of the two vectors: $(\mathbf{t} - \mathbf{v}_0)$ and $\mathbf{r}$. Thus we have

$$\|\pi_{\mathbf{u}}(\mathbf{x})\| = \|\pi_{\mathbf{u}}(\mathbf{t} - \mathbf{v}_0)\| + \|\pi_{\mathbf{u}}(\mathbf{r})\|. \tag{6.20}$$

By Line 3 of Algorithm 6.6, the vector $\mathbf{r}$ is sampled from the distribution with the support $C^n_{\mathbf{u}}(h, R') \cap \Lambda$, therefore,

$$\|\pi_{\mathbf{u}}(\mathbf{r})\| \leqslant R'. \tag{6.21}$$

The vector $\mathbf{v}_0 \in \Lambda$ is a lattice vector, obtained by reducing $\mathbf{t}$ with the basis $\mathbf{C}$ using the nearest plane algorithm. Then, using Lemma 3.3, we get

$$\|\pi_{\mathbf{u}}(\mathbf{t} - \mathbf{v}_0)\| \leqslant \sqrt{\sum_{i=2}^{n} \frac{\|\mathbf{c}_i\|^2}{4}} \leqslant \frac{R}{2}. \tag{6.22}$$

Combining (6.19), (6.20), (6.21), and (6.22), we obtain

$$\|\mathbf{x} - \mathbf{x}'\| \leqslant \left(1 - \frac{R}{R'}\right) \cdot \left(R' + \frac{R}{2}\right) \leqslant R'. \tag{6.23}$$

Thus, (6.23) and (6.18) imply that the inequality, given by (6.16), holds for the vectors $\mathbf{x}'$ and $\mathbf{w}$, returned by Algorithm 6.6.

By Lines 3-5 of Algorithm 6.6, after rescaling, the distribution of $\mathbf{x}$ should be close to the uniform on $C^n_{\mathbf{u}}$. As the vector $\mathbf{x}'$ is obtained from $\mathbf{x}$ by rescaling the projection of $\mathbf{x}$ on $\text{span}(\mathbf{u})^{\perp}$ (Line 7), the distribution of $\text{Rescale}_{\mathbf{u},h}(\mathbf{x}')$ also should be close to the uniform on $C^n_{\mathbf{u}}$.

Algorithm 6.6 consists is calling the nearest plane algorithm on the input $(\mathbf{C}, \mathbf{t})$, sampling the vector $\mathbf{r}$ from the distribution D, and performing several arithmetic operations on n-dimensional vectors (Lines 4-7). Since the nearest plane algorithm has the time complexity $\text{poly}(n)$ and since we assume that sampling from D also takes polynomial time, the time complexity of the whole algorithm is $\text{poly}(n)$. □

6.4.2 Solving CVPP for an integer lattice with a prime volume.

In this section, we describe a decoding algorithm that solves CVPP for prime volume integer lattices using the cylindrical sieving.

Let Λ be an n-dimensional integer lattice whose volume is a prime number p. The decoding algorithm takes as input a basis $\mathbf{B}$ of the lattice Λ, a description of Λ, produced by the preprocessing algorithm for prime volume lattices (see Algorithm 6.3), and a target vector $\mathbf{t} \in \mathbb{R}^n$, and returns a lattice vector $\mathbf{x} \in \Lambda$ such that $\|\mathbf{x} - \mathbf{t}\| = O(\sqrt{n})\,\mathrm{vol}(\Lambda)^{1/n}$.

Let $N = 2^{n/2}$, $R = O(\sqrt{n})$, $\mathbf{u} = (1, 0, \ldots, 0)^t \in \mathbb{R}^n$. Recall that the preprocessing algorithm produces the lists $C_1, \ldots, C_k$ of lattice vectors such that the vectors from the i-th list cover the hypercylinder of the height p/N^{i-1} and the radius $R \cdot 2^{(i-1)/2}$ with half-cylinders of height p/N^i.

The decoding algorithm has two parts. First, it transforms the input target vector $\mathbf{t}$ into a vector $\mathbf{x}'$ that belongs to the hypercylinder $\mathbf{C}^n_{\mathbf{u}}(h, R)$, i.e., the first hypercylinder covered by the lattice vectors, returned by the preprocessing. For the targets inside the first hypercylinder we have the decoding procedure, given by Algorithm 6.5. The second part is applying Algorithm 6.5 to $\mathbf{x}'$. The vector, returned by Algorithm 6.5, is then used to recover a lattice vector that is close to the initial target $\mathbf{t}$.

Consider the first part of the decoding: transforming the target $\mathbf{t}$ into the target $\mathbf{x}'$ that can be given as input to Algorithm 6.5. In the previous section, we have seen that this can be done by Algorithm 6.6, if we have an access to the oracle that samples a uniform-like distribution D on $\mathbf{C}^n_{\mathbf{u}}(h, R) \cap \Lambda$. More precisely, uniform-like means that, for $\mathbf{x}$ sampled from D, $\mathbf{y} = \mathrm{Rescale}_{\mathbf{u},h}(\mathbf{x})$ has the distribution that is close to the uniform on $C^n_{\mathbf{u}}$.

For a prime volume lattice, we can replace an oracle that samples the distribution D by the following simple algorithm. First, we randomly sample an $(n-1)$ dimensional integer vector $\mathbf{u}_x$ such that $\|\mathbf{u}_x\| \leqslant R$. Then, using the shape of the HNF of Λ, we compute a lattice vector $\mathbf{x}$ that corresponds to $\mathbf{u}_x$:

$$\mathrm{HNF}(\Lambda) = \begin{pmatrix} p & a_2 & \ldots & a_n \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & 1 \end{pmatrix}, \quad \mathbf{u}_x = \begin{pmatrix} x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} \sum_{i=2}^{n} a_i \cdot x_i \mod p \\ x_2 \\ \vdots \\ x_n \end{pmatrix}. \tag{6.24}$$

In order to sample a random vector inside the ball of the radius $R = O(\sqrt{n})$, we sample each coordinate independently from the Bernoulli distribution with the parameter α, i.e.,

$$\mathbb{P}\{x_i = 1\} = \alpha, \quad \mathbb{P}\{x_i = 0\} = 1 - \alpha,$$

for all $i \in \{2, \ldots, n\}$. Changing α allows to change the radius of sampling. The algorithm for replacing the oracle that samples D is summarized by Algorithm 6.7.

Algorithm 6.7: Sample a random lattice point in a hypercylinder

```
sampleD(B, α):
    {p, a_2, ..., a_n} ← HNF(B)
    x_1 ← 0
    for i ∈ {2, ..., n} do
        x_i ← Bernoulli(α)
        x_1 = x_1 + x_i · a_i  mod p
    x ← (x_1, x_2, ..., x_n)^t
    return x
```

Some properties of the distribution, produced by Algorithm 6.7, are described in Lemma 6.7.

Lemma 6.7. Let Λ be an n-dimensional lattice such that $\mathrm{vol}(\Lambda) = p$ is a prime number. Let $\mathbf{B}$ be a basis of Λ, let $\alpha \in (0; 1)$. Let $\mathbf{u} = (1, 0, \ldots, 0)^t \in \mathbb{R}^n$. Then, Algorithm 6.7, given as input $\mathbf{B}$ and α, returns a vector $\mathbf{r} = (r_1, \ldots, r_n)^t \in \Lambda$ such that

1. $|r_1| \leqslant \dfrac{p}{2}$;
2. $\|\pi_{\mathbf{u}}(\mathbf{r})\| \leqslant \sqrt{n}$;
3. for $\varepsilon \geqslant 0$, the following inequality holds:

$$\mathbb{P}\{\|\pi_{\mathbf{u}}(\mathbf{r})\| \leqslant \sqrt{(\alpha + \varepsilon)n}\} \geqslant 1 - e^{-2n\varepsilon^2}. \tag{6.25}$$

The time complexity of the algorithm is $\mathrm{poly}(n)$.

Proof. As Λ is a prime volume lattice, its HNF is given by (6.24). Therefore, the vector $\mathbf{x}$, returned by the algorithm is a lattice vector as an integer linear combination of the vectors that form the HNF of Λ.

The first property is ensured by Line 6 of Algorithm 6.7.

Consider the second and the third properties. The vector $\pi_{\mathbf{u}}(\mathbf{r})$ is given by $(0, r_2, \ldots, r_n)$. Then, $\|\pi_{\mathbf{u}}(\mathbf{r})\| = \sqrt{\sum_{i=2}^{n} r_i^2}$. As each r_i is sampled independently from the Bernoulli distribution with the support $\{0, 1\}$, we get $\|\pi_{\mathbf{u}}(\mathbf{r})\| = \sqrt{\sum_{i=2}^{n} r_i} \leqslant \sqrt{n}$. The inequality, given by (6.25), follows from Hoeffding's inequality for the sum of n independent Bernoulli random variables with the parameter α.

The time complexity of the algorithm is the complexity of computing the HNF for $n \times n$ matrix and generating n independent Bernoulli random variables. Thus, the time complexity of the algorithm is polynomial in n. □

We heuristically assume that the distribution, produced by Algorithm 6.7 is uniform-like. Then, we can use Algorithm 6.7 as an oracle for Algorithm 6.6. Thus, by combining Algorithms 6.5 to 6.7, we obtain the decoding algorithm that solves CVPP for prime volume integer lattices (see Algorithm 6.8).

Algorithm 6.8: Solving CVPP with cylindrical sieving for lattices with prime volume

input : a target vector $\mathbf{t} \in \mathbb{R}^n$, a basis $\mathbf{B}$ of Λ, the description C of Λ, produced by Algorithm 6.3.
output: a lattice vector $\mathbf{x} \in \Lambda$ such that $\|\mathbf{t} - \mathbf{x}\| = O(\sqrt{n})\,\mathrm{vol}(\Lambda)^{1/n}$

```
1 findCloseLatticeVector(t, B, C):
2     C ← HNF(B)
      /* Algorithm 6.6 with Algorithm 6.7 for sampling a random lattice vector */
3     (w, x') ← TRANSFORMTARGET(C, t)
4     v ← DECODE(C, x')                                   ▷ Algorithm 6.5
5     return w + v
```

The complexity and the probability of success of Algorithm 6.8 are described in Theorem 6.1.

Theorem 6.1. Let $\Lambda \subset \mathbb{Z}^n$ be an n-dimensional lattice whose volume is a prime number p. Let C be a description of the lattice Λ, produced by Algorithm 6.3. Let Assumption 6.2

hold. Then, Algorithm 6.8, given as input a basis $\mathbf{B}$ of the lattice Λ, the description C, and a target vector $\mathbf{t} \in \mathbb{R}^n$, finds a lattice vector $\mathbf{v} \in \Lambda$ such that

$$\|\mathbf{v} - \mathbf{t}\| \leqslant 0.41(1 + o(1))\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n} \tag{6.26}$$

in poly(n) time, with probability at least $1 - e^{-n+o(n)}$.

Proof. Let $R = \sqrt{0.084(n-1)}$, $N = 2^{n/2}$, and $k = \left\lceil \frac{2}{n}\log(p) \right\rceil$. First, Algorithm 6.8 computes the Hermite Normal Form of Λ. We denote the basis of Λ that corresponds to the HNF as $\mathbf{C}$. As $\mathrm{vol}(\Lambda) = p$ is a prime number, $\mathbf{C}$ is given by (6.24). Then, the Gram-Schmidt orthogonalization $\mathbf{C}^*$ satisfies:

1. $\|\mathbf{c}_1\| = p$;
2. for all $i \in \{2, \ldots, n\}$, $\|\mathbf{c}_i^*\| = 1$.

Then, Algorithm 6.8 applies Algorithm 6.6 to the input $(\mathbf{C}, \mathbf{t})$. Algorithm 6.6 uses Algorithm 6.7 as oracle for sampling a random lattice point $\mathbf{r}$. The support of the distribution, produced by Algorithm 6.7 is the set $\Lambda \cap C_{\mathbf{u}}^n(p, R')$, where $R' = \sqrt{n}$. Then, by Lemma 6.6, Algorithm 6.6 returns a pair of vectors $(\mathbf{w}, \mathbf{x}')$ such that

1. $\mathbf{w} \in \Lambda$;
2. $\mathbf{x}' \in \Lambda \cap C_{\mathbf{u}}^n(p, R)$
3. if there exists $\mathbf{v} \in \Lambda$ and $r > 0$ such that $\|\mathbf{v} - \mathbf{x}'\| \leqslant r$, then

$$\|\mathbf{t} - (\mathbf{v} + \mathbf{w})\| \leqslant r + R'. \tag{6.27}$$

Finally, the algorithm applies the decoding for targets inside cylinder (see Algorithm 6.5) to the vector $\mathbf{x}'$ and the description C of the lattice Λ. Here we suppose that Assumption 6.2 holds, i.e., the distribution of $\mathbf{x}'$ is required in Lemma 6.5. As the description C is produced by the preprocessing described in Algorithm 6.3 and $\mathbf{x}'$ is produced by Algorithm 6.6, the input $(C, \mathbf{x}')$ satisfies the requirements of Lemma 6.5. Then, by Lemma 6.5, with probability at least $1 - e^{-n+o(n)}$, Algorithm 6.5 returns a vector $\mathbf{v}$ such that

$$\|\mathbf{v} - \mathbf{x}'\| \leqslant \sqrt{\frac{p^2}{N^{2k}} + R^2 \cdot 2^k} \leqslant 0.41\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}. \tag{6.28}$$

Combining (6.27) and (6.28), we get

$$\|\mathbf{t} - (\mathbf{v} + \mathbf{w})\| \leqslant \sqrt{n} + 0.41\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n} = 0.41\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n} \cdot (1 + O(\mathrm{vol}(\Lambda)^{-1/n})). \tag{6.29}$$

Assuming that $\log(\mathrm{vol}(\Lambda)) = \omega(n)$, we obtain the bound given by (6.26).

Since the time complexity of both Algorithms 6.5 and 6.6 is poly(n), the time complexity of Algorithm 6.8 is also poly(n). □

6.4.3 Solving CVPP for an arbitrary lattice.

In this section, we consider solving CVPP for an arbitrary n-dimensional lattice Λ. As in the previous section, our goal is to modify the decoding algorithm (see Algorithm 6.5), so that it can be applied to any target $\mathbf{t} \in \mathbb{R}^n$.

Let $h >> R > 0$ be some fixed constants, $N = 2^{\frac{n}{2}}$, and k be a positive integer number. Let $\mathbf{u} \in \mathbb{R}^n$ be a unit vector.

Recall that an input for Algorithm 6.5 consists of the description $C = \{C_1, \ldots, C_k\}$ of the lattice Λ and of the target point $\mathbf{x}$, that satisfy certain conditions (see Conditions 1-3 listed at the beginning of Section 6.4). Informally, the target vector $\mathbf{x}$ should belong to the hypercylinder $C_{\mathbf{u}}^n(h, R)$ that is covered by the vectors from C_1 and, moreover, $\mathbf{x}$ should behave as sampled from the uniform distribution on this hypercylinder.

In Section 6.2, we describe an algorithm that can produce a description $C = \{C_1, \dots, C_k\}$ for an arbitrary lattice Λ such that the vectors from the first list C_1 cover the hypercylinder $C^n_{\mathbf{u}}(h, R)$ with the following parameters:

$$h = q^{\frac{(n-1)^2}{2}} \cdot n \cdot \mathrm{vol}(\Lambda)^{1/n}, \;\; R = q^{-\frac{n-1}{2}} \cdot \mathrm{vol}(\Lambda)^{1/n}, \tag{6.30}$$

where $q > \frac{4}{3}$ is some fixed constant, while the vectors from the last list C_k cover the hypercylinder with the parameters, given by

$$h_k = O(\mathrm{vol}(\Lambda)^{1/n}), \;\; R_k = O(\mathrm{vol}(\Lambda)^{1/n}) \cdot \sqrt{n-1}. \tag{6.31}$$

Therefore, if we use the output of Algorithm 6.4 as a description of Λ for Algorithm 6.5, we need a procedure that transforms an arbitrary target vector $\mathbf{t} \in \mathbb{R}^n$ into a target point $\mathbf{x}$ such that

$$\mathbf{x} \in C^n_{\mathbf{u}}(h, R); \tag{6.32}$$

$$\mathbf{x} \text{ behaves as sampled from the uniform distribution on } C^n_{\mathbf{u}}(h, R). \tag{6.33}$$

In Section 6.4.1, we describe an algorithm (see Algorithm 6.6) that transforms an arbitrary $\mathbf{t} \in \mathbb{R}^n$ into $\mathbf{x}$ that satisfies the conditions given by (6.32) and (6.33). To do so, Algorithm 6.6 requires a basis $\mathbf{C}$ of Λ such that $\|\mathbf{c}_1\| \leqslant h$ and, for all $i \in \{2, \dots, n\}$, $\|\mathbf{c}_i\| \leqslant \frac{R}{\sqrt{n-1}}$, and an access to an oracle that samples a distribution with the support $C^n_{\mathbf{u}}(h, R') \cap \Lambda$ that is, in some sense, close to the uniform on $C^n_{\mathbf{u}}(h, R')$, where $R' \geqslant R$ is close to R.

The desired basis $\mathbf{C}$ can be obtained by applying the unbalanced reduction to any LLL-reduced basis of Λ (see Section 3.6 for the description of the unbalanced reduction algorithm).

In order to get an oracle that samples a uniform-like distribution on $C^n_{\mathbf{u}}(h, R)$, we can sample a discrete Gaussian distribution on a certain projected sublattice of Λ. More precisely, computing the basis $\mathbf{C}$ using the unbalanced reduction reveals a projected sublattice $\pi_2(\Lambda)$ together with its basis $\pi_2(\mathbf{C})$ such that $\|\pi_2(\mathbf{C})^*\|$ is smaller than $R/\sqrt{n-1}$. Having such a basis allows us to sample a discrete Gaussian distribution on $\pi_2(\Lambda)$ with the small parameter $s = \frac{R}{\sqrt{n-1}} \cdot \omega(\sqrt{\log(n)})$ in polynomial time using Klein's algorithm (see Section 3.5 or [GPV08]). The lattice vectors, sampled in such a way, with high probability will have short projection on $\mathrm{span}(\mathbf{c}_1)^\perp$. Then, we can reduce the sampled vectors with $\mathbf{c}_1 \leqslant h$ in order to obtain vectors that have short enough projection on $\mathrm{span}(\mathbf{c}_1)$ to get inside the hypercylinder $C^n_{\mathbf{u}}(h, R')$.

Thus, using the unbalanced reduction of the lattice basis together with sampling the discrete Gaussian distribution on a projected sublattice, we can sample a random vector from the set $C^n_{\mathbf{u}}(h, R') \cap \Lambda$. The approach is summarized in Algorithm 6.9

The distribution, produced by Algorithm 6.9, is analyzed in Lemma 6.8.

Lemma 6.8. Let $q > 4/3$. Let $\mathbf{B}$ be a basis of an n-dimensional lattice Λ. Then, there exists a unit vector $\mathbf{u} \in \mathbb{R}^n$ such that Algorithm 6.9, given $\mathbf{B}$ and q as input, returns a vector $\mathbf{r} \in \Lambda$ that satisfies

1. the norm of the projection of $\mathbf{r}$ on $\mathrm{span}(\mathbf{u})$ is bounded by $\frac{1}{2} \cdot q^{(n-1)^2/2} \cdot n\,\mathrm{vol}(\Lambda)^{1/n}$;
2. $\|\pi_{\mathbf{u}}(\mathbf{r})\| \leqslant \sigma \cdot \sqrt{n}\log(n)$, where $\sigma = q^{-(n-1)/2}\,\mathrm{vol}(\Lambda)^{1/n}$.

The expected time complexity of the algorithm is $\mathrm{poly}(n)$. With probability at least $1 - O(2^{-n})$, the number of calls to the discrete Gaussian sampler is equal to 1.

Proof. By Corollary 3.1, the unbalanced reduction algorithm (see Algorithm 3.6), given as input a $(q^{-1}+1/4)$-LLL-reduced basis $\mathbf{B}$ and a parameter $\sigma = q^{-(n-1)/2}\,\mathrm{vol}(\Lambda)^{1/n}$, returns a basis $\mathbf{C}$ of Λ that satisfies

1. $\|\mathbf{c}_1\| \leqslant q^{(n-1)^2/2} \cdot n \cdot \mathrm{vol}(\Lambda)^{1/n}$;

Algorithm 6.9: Sample a random lattice point in a hypercylinder for an arbitrary lattice

```
1  sampleD(B, α):
2      B ← LLL(B, q)
3      σ ← q^{-(n-1)/2} vol(Λ)^{1/n}
4      C ← UnbalancedReduction(B, σ)                    ▷ Algorithm 3.6
5      s ← σ · log(n))
6      r_max = s · √n
7      r_norm = r_max + 1
8      while (r_norm > r_max) do
9          r ← sampleDiscreteGaussian(π_2(C), s)        ▷ sampling algorithm
           from [GPV08]
10         r_norm ← ‖π_u(r)‖
11     r ← r − ⌊(r^t c_1)/(c_1^t c_1)⌉ · c_1
12     return r
```

2. for all $i \in \{2, \dots, n\}$, $\|\mathbf{c}_i\| \leqslant \sigma$.

Let $\mathbf{u} = \frac{1}{\|\mathbf{c}_1\|} \cdot \mathbf{c}_1$. Then, the bound on the projection of $\mathbf{r}$ on $\mathrm{span}(\mathbf{u})$ from Lemma 6.8 is ensured by reducing the vector $\mathbf{r}$ with $\mathbf{c}_1$ in Line 8 of Algorithm 6.9.

Consider the projection of $\mathbf{r}$ on $\mathrm{span}(\mathbf{u})^\perp$. The projection $\pi_{\mathbf{u}}(\mathbf{r})$ is sampled from the centered discrete Gaussian distribution on the projected sublattice $\pi_2(\Lambda)$ with the parameter $s = \sigma \log(n)$. The norm of $\|\pi_2(\mathbf{C})^*\| = \max_{2 \leqslant i \leqslant n} \|\mathbf{c}_i^*\|$ is bounded by σ. Therefore, by Theorem 3.3, the distribution, statistically close to the described above, can be sampled in polynomial time by Klein's algorithm using the basis $\pi_2(\mathbf{C})$.

Since $\pi_{\mathbf{u}}(\mathbf{r})$ has the discrete Gaussian distribution with the parameter $\sigma \cdot \log(n)$, Lemma 3.4 implies the following probabilistic bound on the norm of $\pi_{\mathbf{u}}(\mathbf{r})$:

$$\mathbb{P}\{\|\pi_{\mathbf{u}}(\mathbf{r})\| \leqslant \sigma \cdot \sqrt{n} \log(n)\} \geqslant 1 - O(2^{-n}), \tag{6.34}$$

Thus, after one call to Klein's sampler we get the vector $\mathbf{r}$ with short enough projection on $\mathrm{span}(\mathbf{u})^\perp$ with the probability at least $1 - O(2^{-n})$.

As the LLL algorithm, the unbalanced reduction, and Klein's algorithm are polynomial-time algorithms, the expected time complexity of Algorithm 6.9 is also $\mathrm{poly}(n)$. □

Then, using Algorithm 6.9 as oracle for Algorithm 6.6, we can transform any target $\mathbf{t} \in \mathbb{R}^n$ into a valid input for Algorithm 6.5. Algorithm 6.10 describes finding close lattice vectors for an arbitrary target using the information, produced by the preprocessing (i.e. Algorithm 6.4).

The complexity of solving CVPP for an arbitrary integer lattice by Algorithm 6.10 is described by Theorem 6.2.

Theorem 6.2. Let Λ be an n-dimensional lattice.. Let C be a description of the lattice Λ, produced by Algorithm 6.4. Let Assumption 6.2 hold. Then, Algorithm 6.10, given as input a basis $\mathbf{B}$ of the lattice Λ, the description C, and a target vector $\mathbf{t} \in \mathbb{R}^n$, finds a lattice vector $\mathbf{v} \in \Lambda$ such that

$$\|\mathbf{v} - \mathbf{t}\| \leqslant 0.483 \cdot (1 + c^{-n+o(log(n))}) \sqrt{n} \, \mathrm{vol}(\Lambda)^{1/n}, \tag{6.35}$$

where c is some positive constant. The time complexity of Algorithm 6.10 is $\mathrm{poly}(n)$, the probability of success is at least $1 - e^{-n+o(n)}$.

Algorithm 6.10: Solving CVPP with cylindrical sieving for an arbitrary lattice

input : a target vector $\mathbf{t} \in \mathbb{R}^n$, a basis $\mathbf{B}$ of Λ, the description C of Λ, produced by Algorithm 6.4.

output: a lattice vector $\mathbf{x} \in \Lambda$ such that $\|\mathbf{t} - \mathbf{x}\| = O(\sqrt{n}) \operatorname{vol}(\Lambda)^{1/n}$

1 `findCloseLatticeVector(`$\mathbf{t}$, $\mathbf{B}$, C`):`

2 $\quad \mathbf{B} \leftarrow \mathrm{LLL}(\mathbf{B}, q)$

3 $\quad \sigma \leftarrow q^{-\frac{n-1}{2}} \operatorname{vol}(\Lambda)^{1/n}$

4 $\quad \mathbf{C} \leftarrow \textsc{UnbalancedReduction}(\mathbf{B}, \sigma)$ ▷ Algorithm 3.6

`/* Algorithm 6.6 with Klein's algorithm from [GPV08] for sampling a random lattice vector */`

5 $\quad (\mathbf{w}, \mathbf{x}') \leftarrow \textsc{transformTarget}(\mathbf{C}, \mathbf{t})$

6 $\quad \mathbf{v} \leftarrow \textsc{decode}(C, \mathbf{x}')$ ▷ Algorithm 6.5

7 $\quad$ **return** $\mathbf{w} + \mathbf{v}$

Proof. Let $\delta = \frac{1}{4} + \frac{1}{q}$ be the parameter of the LLL reduction of the basis $\mathbf{B}$ at the Line 2 of Algorithm 6.10. Let $\sigma = q^{-(n-1)/2} \cdot \operatorname{vol}(\Lambda)^{1/n}$, $R = \sqrt{\frac{n-1}{\pi e}}\sigma$, $N = 2^{n/2}$, and $h = q^{(n-1)^2/2} \cdot n \operatorname{vol}(\Lambda)^{1/n}$. Let $k = \lceil (n-1)\log(q) \rceil$.

Algorithm 6.10 starts with applying the unbalanced reduction with the parameter σ to the δ-LLL-reduced basis $\mathbf{B}$ of Λ. We denote the basis of Λ obtained after the unbalanced reduction as $\mathbf{C}$. By Corollary 3.1, the Gram-Schmidt orthogonalization $\mathbf{C}^*$ satisfies:

1. $\|\mathbf{c}_1\| \leqslant h$;
2. for all $i \in \{2, \ldots, n\}$, $\|\mathbf{c}_i^*\| \leqslant \sigma$.

Then, Algorithm 6.10 applies Algorithm 6.6 to the input $(\mathbf{C}, \mathbf{t})$. Algorithm 6.6 uses Klein's algorithm that samples the discrete Gaussian distribution $D_{\pi_2(\Lambda),s}$ as oracle for sampling a random lattice point $\mathbf{r}$, where $s = \sigma \cdot \log(n)$.

By Lemma 6.8, the norm of the projection $\mathbf{r}$ on $\operatorname{span}(\mathbf{c}_1)^\perp$ is bounded by $R' = \sigma \cdot \sqrt{n}\log(n)$.

Thus, the support of the distribution, produced by Algorithm 6.9 is the set $\Lambda \cap C_{\mathbf{u}}^n(p, R')$. Then, by Lemma 6.6, Algorithm 6.6 returns a pair of vectors $(\mathbf{w}, \mathbf{x}')$ such that

1. $\mathbf{w} \in \Lambda$;
2. $\mathbf{x}' \in \Lambda \cap C_{\mathbf{u}}^n(h, R)$
3. if there exists $\mathbf{v} \in \Lambda$ and $r > 0$ such that $\|\mathbf{v} - \mathbf{x}'\| \leqslant r$, then

$$\|\mathbf{t} - (\mathbf{v} + \mathbf{w})\| \leqslant r + R'. \tag{6.36}$$

Finally, the algorithm applies the decoding for targets inside cylinder (see Algorithm 6.5) to the vector $\mathbf{x}'$ and the description C of the lattice Λ. Here we suppose that Assumption 6.2 holds, i.e., the distribution of $\mathbf{x}'$ is required in Lemma 6.5. As the description C is produced by the preprocessing described in Algorithm 6.4 and $\mathbf{x}'$ is produced by Algorithm 6.6, the input $(C, \mathbf{x}')$ satisfies the requirements of Lemma 6.5. Then, by Lemma 6.5, with probability at least $1 - e^{-n+o(n)}$, Algorithm 6.5 returns a vector $\mathbf{v}$ such that

$$\|\mathbf{v} - \mathbf{x}'\| \leqslant \sqrt{\frac{p^2}{N^{2k}} + R^2 \cdot 2^k} \leqslant 0.483\sqrt{n} \operatorname{vol}(\Lambda)^{1/n}. \tag{6.37}$$

Combining (6.36) and (6.37), we get

$$\|\mathbf{t} - (\mathbf{v} + \mathbf{w})\| \leqslant \sqrt{n}\log(n) \cdot \sigma + 0.483\sqrt{n} \operatorname{vol}(\Lambda)^{1/n} \tag{6.38}$$

$$= \sqrt{n} \operatorname{vol}(\Lambda)^{1/n} \cdot (0.483 + \log(n) \cdot q^{-(n-1)/2}), \tag{6.39}$$

	prime volume	any lattice
preprocessing: time	$2^{n/2}$	$2^{0.531n}$
preprocessing: memory	$2^{n/2}$	$2^{n/2}$
decoding: time	$\mathrm{poly}(n)$	$\mathrm{poly}(n)$
size of lattice description	$\frac{\log(\mathrm{vol}(\Lambda))}{n} \cdot 2^{n/2}$	$n^2 \cdot 2^{n/2}$
distance	0.403	$\sqrt{\frac{2}{\pi e}} \approx 0.484$

Table 6.2 – Complexity of solving CVPP using cylindrical sieving. The time and memory complexities of the algorithms are given ignoring polynomial factors. The row "size of lattice description" represents the number of n-dimensional vectors in the description of an input lattice, produced by the preprocessing algorithms, ignoring constant factors. The row "distance" represents the upper bound on the ratio of the distance between the lattice and the vector, returned by the decoding algorithms and of the Minkowski's bound $\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}$.

which implies the bound given by (6.35).

Since the time complexity of both Algorithms 6.5 and 6.6 is $\mathrm{poly}(n)$, the time complexity of Algorithm 6.10 is also $\mathrm{poly}(n)$.

□

Table 6.2 summarizes the complexity of solving CVPP using the cylindrical sieving in case of a lattice with a prime volume and in the general case.

6.5 Experimental results

Since the theoretical analysis of our algorithm for solving CVPP is based on heuristic assumptions, it is important to provide experiments in order to see whether the proposed assumptions are feasible. In this section, we describe our implementation of Algorithm 6.8 for solving CVPP with the cylindrical sieving and the obtained experimental results.

6.5.1 Description of implementation.

We implement the algorithm that solves CVPP using the cylindrical sieving for lattices with a prime volume. Namely, we implement the preprocessing and decoding algorithms, given by Algorithms 6.3 and 6.8 correspondingly. Both parts are implemented in C++ using GMP library [GMP]. For the implementation, we used simplified versions of Algorithms 6.3 and 6.8. Essentially, the main differences are the initial sampling of the lattice vectors and the randomization of the target point. In the next two paragraphs, we describe all the differences in details, first, for the preprocessing algorithm, then, for the decoding.

Preprocessing. The preprocessing of a lattice with the cylindrical sieving has two parts: the initial generation part (see Lines 2-3 of Algorithm 6.3) and the sieving part (see Lines 6-8 of Algorithm 6.3). The implementation of the sieving part of the preprocessing coincides with the sieving part described in Algorithm 6.3. But, the implemented version of the initial generation slightly differs from the one described in Lines 2-3 of Algorithm 6.3. In Line 3 of Algorithm 6.3, we suggest to use Algorithm 4.1 to generate the lattice points inside the hypercylinder. Algorithm 4.1 enumerates all the integer vectors of dimension $(n-1)$ inside a ball $B^{n-1}(O(\sqrt{n}))$ and then computes the corresponding lattice vector for each of the enumerated integer vectors. In order to simplify the implementation, we replace enumerating the set $\mathbb{Z}^{n-1} \cap B^{n-1}(O(\sqrt{n}))$ by enumerating all the binary vectors

with $k < n$ ones. Also, due to memory constrains, in all our experiments, the initial list of lattice vectors is shorter than theoretically required. By Lemma 4.3, the upper bound on the number of vectors, lost at one iteration of the sieving process, is $O(n \cdot 2^{\frac{n}{2}})$. Thus, for k iterations of sieving, we need the initial list of size $O(n \cdot k \cdot 2^{\frac{n}{2}})$. For the experiments, we usually took the initial list of size about $2k \cdot 2^{\frac{n}{2}}$.

Decoding The implementation of the decoding is also slightly different from Algorithm 6.8. Algorithm 6.8 decode has two parts: transformation of the target point (see Line 3 of Algorithm 6.8) and decoding of the target inside the hypercylinder (see Line 4 of Algorithm 6.8). In the implementation, decoding of the target inside cylinder is exactly the same as described in Algorithms 6.5 and 6.8. But, the transformation of the target is simpler. In our experiments, we consider only target vectors from $\mathbb{Z}^n$, which allows to simplify Algorithm 6.6. If the target $\mathbf{t}$ is an integer vector, then, after reducing it with the HNF of a prime volume lattice (see (4.1) for the shape of the HNF), we get a vector $\mathbf{v} \in \Lambda$ such that the difference $\mathbf{v} - \mathbf{t}$ has only one non-zero coordinate:

$$\mathbf{v} - \mathbf{t} = (v_1, 0, \ldots, 0)^t \in \mathbb{Z}^n, \ |v_1| \leqslant \frac{\mathrm{vol}(\Lambda)}{2}.$$

Then, to randomize the target, we can just add to $\mathbf{v} - \mathbf{t}$ a random vector that lies inside the hypercylinder and reduce the first coordinate of the resulting vector modulo $\mathrm{vol}(\Lambda)$. The implementation version of the transformation of the target is described by Algorithm 6.11.

Algorithm 6.11: Transform target point (implementation version)

input : a target vector $\mathbf{t} \in \mathbb{Z}^n$, $\mathbf{B} = \mathrm{HNF}(\Lambda)$.
output: $\mathbf{x} \in C_{\mathbf{u}}(\mathrm{vol}(\Lambda), O(\sqrt{n}))$ such that $\mathbf{t} - \mathbf{x} \in \Lambda$, where $\mathbf{u} = (1, 0, \ldots, 0)^t \in \mathbb{Z}^n$.

1 `transformTarget(t, B):`
2 $\quad \mathbf{v} \leftarrow \textsc{NearestPlane}(\mathbf{B}, \mathbf{t})$ ▷ $\mathbf{v} - \mathbf{t} = (v_1, 0, \ldots, 0)^t$, where $v_1 \in \left(-\frac{\mathrm{vol}(\Lambda)}{2}; \frac{\mathrm{vol}(\Lambda)}{2}\right)$
3 $\quad$ **for** $i \in \{2, \ldots, n\}$ **do**
4 $\quad\quad r_i \leftarrow \textsc{Bernoulli}(\alpha)$ ▷ $a_2, \ldots, a_n$ are from the HNF of Λ, see (4.1)
5 $\quad r_1 \leftarrow \sum_{i=2}^{n} r_i \cdot a_i$
6 $\quad \mathbf{x} = ((r_1 + v_1) \mod \mathrm{vol}(\Lambda), r_2, \ldots, r_n)^t$
7 $\quad$ **return x**

Experiments. We tested our implementation on random lattices with prime volumes. To generate a random lattice, we use the algorithm described in Section 2.3. We consider lattices with the following parameters: dimension $n = 20$ and $\mathrm{vol}(\Lambda) \approx 2^{100}$, dimension $n = 30$ and $\mathrm{vol}(\Lambda) \approx 2^{60}$,dimension $n = 40$ and $\mathrm{vol}(\Lambda) \approx 2^{80}$. Here, when we write $\mathrm{vol}(\Lambda) \approx 2^m$ for some $m \in \mathbb{Z}$, it means that the volume of Λ is equal to the biggest prime number that is smaller than 2^m.

For each of the three lattices, we first run the preprocessing and then try to find close vectors for 1000 randomly generated targets, using the description of the lattice, produced by the preprocessing. The results of these experiments are described in Sections 6.5.2 to 6.5.4.

Also, for dimension $n = 30$ and $\mathrm{vol}(\Lambda) \approx 2^{60}$, we picked one target and randomized it by adding random lattice vectors to it (see Algorithm 6.11). Thus we obtained 10^5 different targets and then tried to find close lattice vectors using the cylindrical sieving for each of them. The results of this experiment is described in Section 6.5.5.

All the experiments were performed on a 1.8GHz computer with 4 GB of RAM.

6.5.2 Dimension $n = 30$, volume $\mathrm{vol}(\Lambda) \approx 2^{60}$.

Let Λ be a random lattice of dimension 30 with a volume equal to the biggest prime smaller than 2^{60}. The parameters of the CVPP algorithm (Algorithm 6.3), applied to the lattice Λ, namely, the decrease rate N and the number of iterations k, are given by:

$$N = 2^{n/2} \approx 2^{15}, \quad k = \left\lceil \frac{\log(\mathrm{vol}(\Lambda))}{n/2} \right\rceil = 4.$$

We want to obtain the initial list of size bigger than $2k \cdot N = 8 \cdot 2^{15}$. The size of the initial list L_0 is equal to the number of all the binary vectors of dimension $n-1 = 29$ with m ones. Therefore, taking $m = 6$ is sufficient to obtain the initial list of the desired size: $\binom{29}{6} \approx 14.5 \cdot 2^{15}$.

Preprocessing. The preprocessing of Λ consists in performing 4 iterations of the cylindrical sieving. Performing the preprocessing takes 8.4 seconds. Table 6.3 describes the properties of the list of lattice vectors L and of the list of centers C obtained at each of 4 iterations of the preprocessing.

iteration	d	R	$\lvert C\rvert$	$\lvert L\rvert$	unique values in L	coverage
1	2^{60}	2.45	$0.94 \cdot 2^{15}$	$14.5 \cdot 2^{15}$	100%	93.5%
2	2^{45}	3.46	$1.36 \cdot 2^{15}$	$13.6 \cdot 2^{15}$	47%	92%
3	2^{30}	4.9	$1.06 \cdot 2^{15}$	$12.2 \cdot 2^{15}$	25%	81%
4	2^{15}	6.92	$0.55 \cdot 2^{15}$	$11.3 \cdot 2^{15}$	6% (72%)	85%

Table 6.3 – *Preprocessing of a random lattice* Λ *of dimension* $n = 30$ *with prime volume* $\mathrm{vol}(\Lambda) \approx 2^{60}$.
The columns named d and R represent respectively the height and the radius of the hypercylinder that contains the list L. The columns named $|C|$ and $|L|$ represent the size of the list of centers C and the size of the list of lattice vectors L respectively.
Unique values in L. Let $u(L)$ be the number of unique values of the first coordinate of the vectors from L. The column named *unique values in* L represent the ratio $\frac{u(L)}{|L|}$. For the iteration 4, the number in brackets in column *unique values in* L is the ratio $\frac{u(L)}{2^{15}}$.
Coverage. The values in the column *coverage* are obtained in the following way. For each iteration, we divide the line segment $[0; d)$ into $N = 2^{15}$ non-intersecting segments. Then, we compute the number of covered segments, i.e., the segments that contain at least one center from the list C. The column *coverage* represents the percentage of the covered segments.

First coordinate. Figure 6.2 is a visualization of the first coordinate of the vectors from the lists L and C during the preprocessing. The pictures from Figure 6.2 are obtained in the following way. Let L be the list of lattice vectors from the i-th iteration. First, we take a random subset of L of size 1000. Then, for each vector $\mathbf{v}$ from the chosen subset we add a blue point on the i-th picture such that the horizontal coordinate of the blue point is equal to the first coordinate of $\mathbf{v}$ and the vertical coordinate is a real number, sampled uniformly at random from $[0; 1)$.

The centers from C are visualized in the similar way. We take a random subset of C size of size equal to $\left\lceil 1000 \cdot \frac{|C|}{|L|} \right\rceil$ and then for each point from the obtained subset we add an orange star on the picture in the same way as for the vectors from L.

While analysing the complexity of the preprocessing, we heuristically assume that the distribution of the points inside the hypercylinder is quasi-uniform (see Definition 3.4

and Assumption 6.1). Informally, for the first coordinate of the vectors, it means that there is no gaps in the distribution of the first coordinate, i.e., there is no sub-intervals with very low probability. The pictures from Figure 6.2 show that, indeed, at each iteration there is no visible gaps in the distribution of the first coordinate. The only observable deviation from uniformity can be noticed for the last two iterations. For the last two iterations, there are quite a lot of points with the first coordinate equal to zero. But, since there are still points all along the whole interval, it should not affect the behavior of the algorithm.

Number of the unique values of the first coordinate in the list L is given by the column "unique values in L" in Table 6.3. We see that just after the initial generation of the list all the values are unique. After 3 iterations of sieving we are left with 25% of unique values, which is still a lot: about $3 \cdot 2^{15}$ unique values of the first coordinate. After the last iteration of sieving there are only 6% of unique values. This percentage looks small, but actually it is normal, because at the 4-th iteration the first coordinate ranges from 0 to 2^{15}, thus, there can not be more than 2^{15} different values of it, while the size of $|L|$ is $11.3 \cdot 2^{15}$.

Last $(n-1)$ coordinates. At each iteration of the cylindrical sieving, the algorithm considers the differences of the lattice vectors from L and rejects those that do not fit at least one of the two requirements:

- the first coordinate of the difference should be small enough;
- the length of the vector, formed by the last $(n-1)$ coordinates of the difference should be short enough: if the list L belongs to the hypercylinder of radius R, for the next iteration we keep only the differences such that the length of the vector, formed by their last $(n-1)$ coordinates, is smaller than $R\sqrt{2}$.

For each of the 4 iterations, we have computed the number of differences such that their first coordinate is small enough, but the vector formed by the last $(n-1)$ coordinates, is too long. The results are summarized in Table 6.4.

iteration	number of differences	rejected	ratio
1	444361	0	0
2	458843	59182	13%
3	364870	34110	9%
4	346935	21991	6%

Table 6.4 – Number of pairs, rejected due to the length of the vector formed by the last $(n-1)$ coordinates.
Column *number of differences* denotes the number of differences such that their first coordinate fits the requirements, column *rejected* denotes the number of differences that were rejected because of the length of the vector formed by the last $(n-1)$ coordinates of the difference.

For the analysis of the algorithm, we assume that the vectors, formed by the last $(n-1)$ coordinates, behave as uniformly distributed on a sphere (see Assumption 6.1). Thus, we expect that about one half of the differences that have small enough first coordinate, should be rejected because the last $(n-1)$ coordinates. Table 6.4 shows that the rejections caused by the last $(n-1)$ coordinates appear rarer than we theoretically expect.

Decoding. We tried to solve 1000 CVP instances using the description of the lattice obtained after the preprocessing. Each target is a random n-dimensional vector with coordinates, sampled independently from the uniform distribution on integer numbers from 0 to some huge threshold T. The results are presented in Table 6.5. More than 95% of

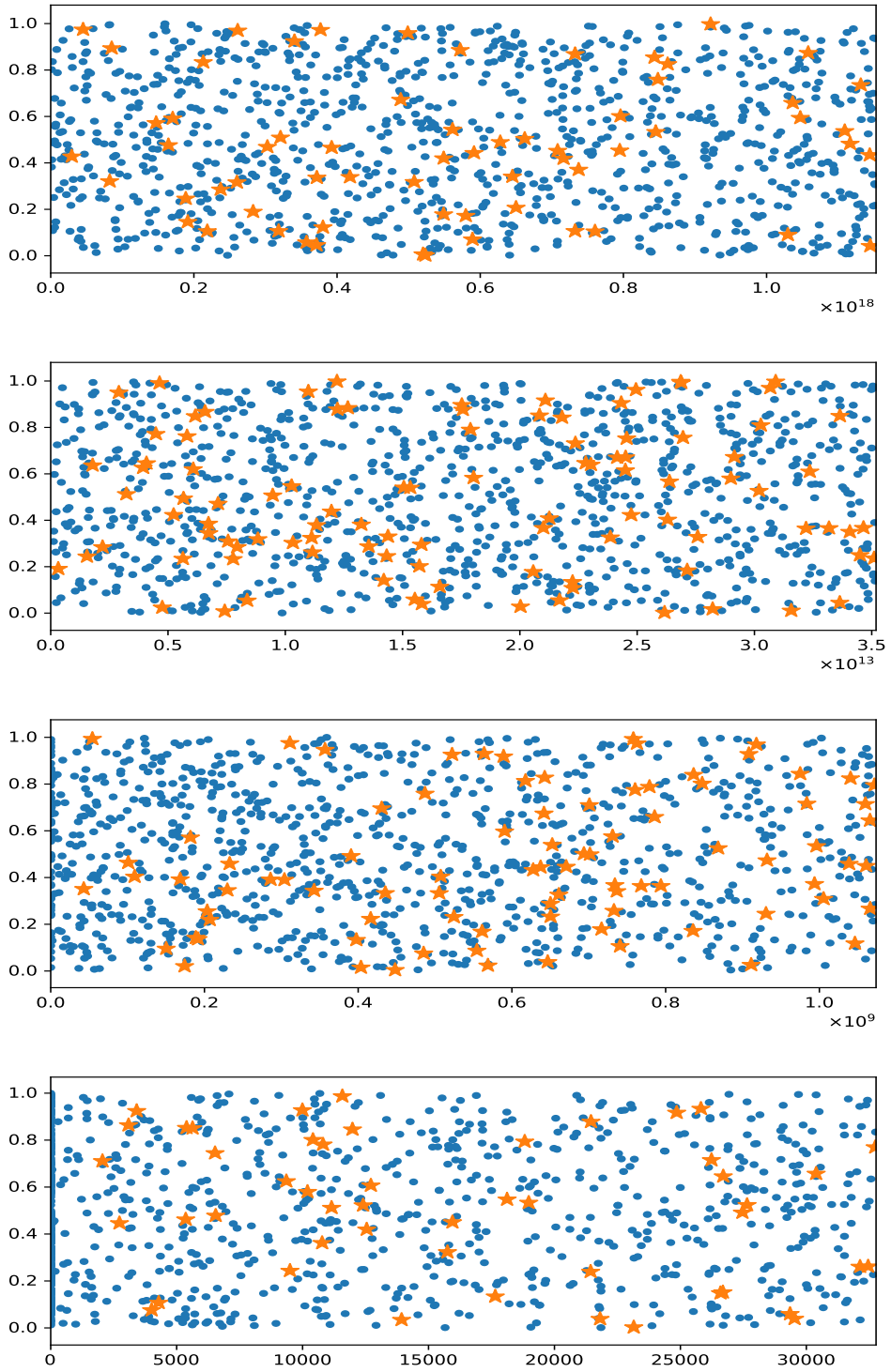


Figure 6.2 – First coordinate of vectors from L and centers at each iteration of preprocessing for lattice Λ of dimension $n = 30$ with volume $\mathrm{vol}(\Lambda) \approx 2^{60}$.
The horizontal axis represent the value of the first coordinate of the vectors. The i-th picture from the above represents the i-th iteration. Blue points denote vectors from the list L, orange stars denote centers. Blue points obtained as a random sample of size 1000 from the list L, orange starts is the sample from C of size $\left\lceil 1000 \cdot \frac{|C|}{|L|} \right\rceil$.

the CVP instances were successfully solved. The average distance from the target to the lattice point found by the algorithm is 8.91, which is more than 2 times smaller than the Minkowski's bound of the lattice Λ: $\sqrt{n} \cdot \mathrm{vol}(\Lambda)^{1/n} = \sqrt{30} \cdot 2^{60/30} \approx 21.9$.

success rate	time	average distance	maximum distance
953/1000	18.2 ms	8.91	12.08

Table 6.5 – Decoding for a lattice Λ of dimension $n = 30$ with prime volume $\mathrm{vol}(\Lambda) \approx 2^{60}$. *Success rate* denotes the number of targets for which close lattice vector is successfully recovered. *Time* is the average time of solving one CVP instance. *Average (maximum) distance* denotes the average (maximum) distance from recovered lattice vector to the target.

6.5.3 Dimension $n = 20$, volume $\mathrm{vol}(\Lambda) \approx 2^{100}$.

In this section, we present the results of applying the cylindrical sieving to solving CVPP for a random lattice Λ of dimension $n = 20$ with the volume equal to the largest prime number smaller than 2^{100}. The number of iterations and the decrease rate of the first coordinate of Algorithm 6.3 for these parameters are given by

$$k = \left\lceil \frac{\log(\mathrm{vol}(\Lambda))}{n/2} \right\rceil = 10, \quad N = 2^{10}.$$

In order to produce the initial list of lattice vectors of size $2k \cdot 2^{n/2}$, we enumerate all binary vectors with $m = 10$ ones.

Preprocessing. The running time of the preprocessing is 4.25 seconds.

iteration	d	R	$\lvert C\rvert$	$\lvert L\rvert$	unique values in L	coverage
1	2^{100}	4.47	1013	$90 \cdot 2^{10}$	100%	99%
2	2^{90}	6.32	1025	$89 \cdot 2^{10}$	45%	98%
3	2^{80}	8.94	1139	$88 \cdot 2^{10}$	31%	96%
4	2^{70}	12.65	1254	$87 \cdot 2^{10}$	24%	97%
5	2^{60}	17.88	1346	$85.8 \cdot 2^{10}$	18%	96%
6	2^{50}	25.29	1325	$84.6 \cdot 2^{10}$	15%	96%
7	2^{40}	35.77	1270	$83.3 \cdot 2^{10}$	11%	94%
8	2^{30}	50.59	1174	$82 \cdot 2^{10}$	8%	91%
9	2^{20}	71.55	1046	$80.89 \cdot 2^{10}$	6%	85%
10	2^{10}	101.19	637	$79.8 \cdot 2^{10}$	1.2%(96%)	98%

Table 6.6 – Preprocessing of a random lattice Λ of dimension $n = 20$ with prime volume $\mathrm{vol}(\Lambda) \approx 2^{100}$. The structure of the table is identical to Table 6.3
.

Decoding. We have tried to solve 1000 CVP instances using the description of the lattice, produced by the preprocessing. The results are summarized in Table 6.7.

The average distance from the target to the lattice vector recovered by the implementation of Algorithm 6.8 is lower than the Minkowski's bound of the lattice Λ, which is given by $\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n} \approx 143.1$.

success rate	time	average distance	maximum distance
998/1000	29.7 ms	85	123.2

Table 6.7 – Decoding for a lattice Λ of dimension $n = 20$ with prime volume $\mathrm{vol}(\Lambda) \approx 2^{100}$. The structure of this table is similar to Table 6.5.

6.5.4 Dimension $n = 40$, volume $\mathrm{vol}(\Lambda) \approx 2^{80}$.

In this section, we describe the results of running CVPP algorithm for a random lattice of dimension $n = 40$ with the volume equal to the largest prime number smaller than 2^{80}. The number of iterations and the decrease rate of the first coordinate of Algorithm 6.8 for these parameters are given by

$$k = \left\lceil \frac{\log(\mathrm{vol}(\Lambda))}{n/2} \right\rceil = 4, \quad N = 2^{20}.$$

In order to obtain the initial list of lattice vectors of size at least $2k \cdot 2^{n/2}$, we enumerate all binary vectors $m = 7$ ones.

Preprocessing. For this example, the preprocessing takes 6 minutes 58 seconds.

iteration	d	R	$\lvert C\rvert$	$\lvert L\rvert$	unique values in L	coverage
1	2^{80}	3.74	$0.94 \cdot 2^{20}$	$14.7 \cdot 2^{20}$	100%	94%
2	2^{60}	5.29	$1.37 \cdot 2^{20}$	$13.7 \cdot 2^{20}$	42%	91%
3	2^{40}	7.48	$0.99 \cdot 2^{20}$	$12.4 \cdot 2^{20}$	20.5%	77.4%
4	2^{20}	10.58	$0.48 \cdot 2^{20}$	$11.4 \cdot 2^{20}$	5.3%(60%)	77.3%

Table 6.8 – Preprocessing of a random lattice Λ of dimension $n = 40$ with prime volume $\mathrm{vol}(\Lambda) \approx 2^{80}$. The structure of the table is similar to Table 6.3.

Decoding. As before, we tried to solve 1000 CVP instances using the description of the lattice, produced by the preprocessing. The results are summarized in Table 6.9.

success rate	time	average distance	maximum distance
902/1000	1.7 s	9.79	12.45

Table 6.9 – Decoding for a lattice Λ of dimension $n = 40$ with prime volume $\mathrm{vol}(\Lambda) \approx 2^{80}$. The structure of the table is similar to Table 6.5.

The average distance from the target to the lattice vector recovered by our implementation of Algorithm 6.8 is more than two times smaller than the Minkowski's bound of the lattice Λ, given by $\sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n} \approx 25.3$.

6.5.5 Randomization of one target point

While analysing the complexity of the decoding algorithm (see Algorithm 6.5), we assume that at each iteration the target point behaves like uniformly distributed inside the corresponding hypercylinder (see Assumption 6.2). In order to produce a target that looks like random for the decoding algorithm, we randomize the input target by adding

a random lattice point to it (see Algorithms 6.6 and 6.11). In order to check how this randomization works in practice, we performed the following experiment.

For the experiment, we used the preprocessing of a random lattice of dimension $n = 30$ with a volume equal to the highest prime number smaller than 2^{60} (the same lattice as in Section 6.5.2). We randomly picked one target point and randomized it: we produced 10^5 different target points by adding random lattice points to the chosen target as in Algorithm 6.11.

Then, for each of the obtained target points we tried to find a close lattice vector using the decoding algorithm. The decoding was successful for 94651 targets out of 10^5, i.e. 94.6% of CVP instances were successfully solved. This ratio is close to the ratio that we get for independent random targets in Section 6.5.2 (see Table 6.5).

The visualisation of the list of randomized targets at each iteration of the decoding is presented by Figure 6.3.

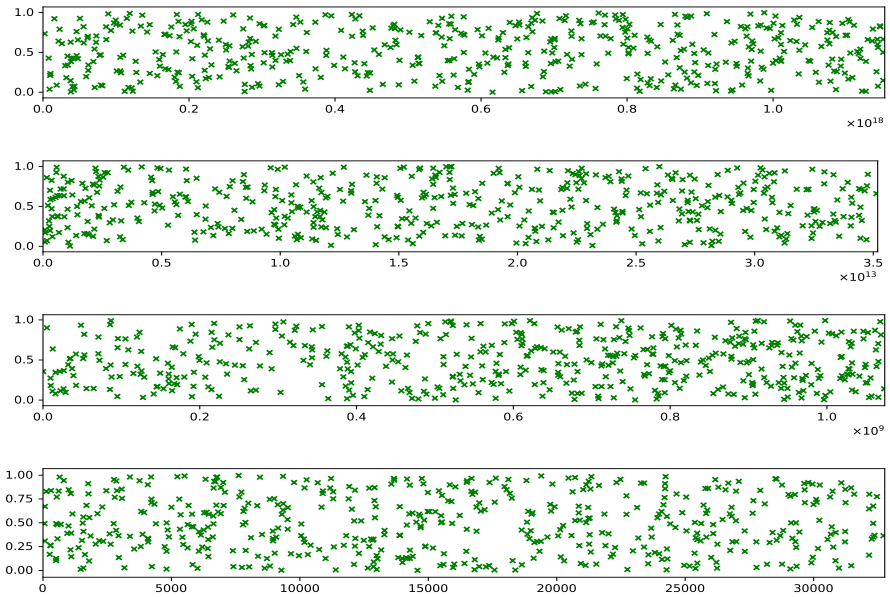


Figure 6.3 – First coordinate of randomized targets at each iteration of decoding for a random lattice of dimension $n = 30$ of volume $\mathrm{vol}(\Lambda) \approx 2^{60}$.
The i-th picture from the above represents i-th iteration. The i-th picture is obtained in the following way. First, we take a random subset of size 500 of the list of targets at the i-th iteration. Then, for each point from the obtained subset we sample a random number from the uniform distribution on the interval $[0; 1)$. The horizontal axis on the picture corresponds to the first coordinate of the vectors from the subset, the vertical axis corresponds to the random numbers from the interval $[0; 1)$ that were matched up with the points from the subset.

Part II

Security of the TFHE scheme

Chapter 7

Background on TFHE and security of LWE-based cryptosystems

In this chapter, we recall all the background information needed in the second part of the thesis. In Section 7.1, we recall some basic information about the TFHE scheme and its security. Then, in Section 7.2, we recall the definition and the properties of the modular Gaussian distribution, which occurs in TFHE. In Sections 7.3 and 7.4, we briefly review existing lattice and hybrid lattice attacks against LWE. Then, in Section 7.5, we recall practical models for the complexity of lattice reduction algorithms that we use later in this part to estimate the complexity of our attack. Finally, in Section 7.6, we recall some useful probability concentration inequalities.

Notation. We denote the set of real numbers modulo 1 as the torus $\mathbb{T}$. For a finite set S, we denote by $\mathcal{U}(S)$ the discrete uniform distribution on S. For any compact set $S \subset \mathbb{R}^n$, the uniform distribution over S is also denoted by $\mathcal{U}(S)$. When S is not specified, $\mathcal{U}$ denotes uniform distribution over $(-0.5; 0.5)$.

7.1 TFHE and its security.

In this part of the thesis, we consider the security of the Fast Fully Homomorphic Encryption Scheme over the Torus (TFHE) [CGGI16, CGGI17, CGGI20]. TFHE is currently the FHE scheme with the fastest implementation that we aware of: it performs the gate bootstrapping in time of 10-20 ms on a single core.

TFHE is a descendant of the GSW scheme [GSW13], therefore, the security of TFHE is based on the LWE problem and its ring variants (see Section 2.5.2 for the description of LWE). The authors of TFHE propose the generalization of the LWE problem and of the GSW construction over the real torus $\mathbb{T}$.

(T)LWE problem. Abstractly, all operations of the TFHE scheme are defined on the real torus $\mathbb{T}$. In order to estimate the security of the scheme it is convenient to consider a scale-invariant version of LWE problem.

Definition 7.1 (Learning with Errors, [BLP+13, Definition 2.11])**.** Let $n \geqslant 1$, $\mathbf{s} \in \mathbb{Z}^n$, ξ be a distribution over $\mathbb{R}$ and $\mathcal{S}$ be a distribution over $\mathbb{Z}^n$.

We define the *$LWE_{\mathbf{s},\xi}$ distribution* as the distribution over $\mathbb{T}^n \times \mathbb{T}$ obtained by sampling a vector $\mathbf{a}$ from the uniform distribution on $\mathbb{T}^n$, sampling e from ξ, and returning $(\mathbf{a}, \mathbf{a}^t\mathbf{s}+e)$. We call pairs $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$ sampled from the $\mathrm{LWE}_{\mathbf{s},\xi}$ distribution $\mathrm{LWE}_{\mathbf{s},\xi}$ samples.

Given access to outputs from this distribution, we can consider the two following problems:

— *D*ecision-LWE. Distinguish, given arbitrarily many samples, between $\mathcal{U}(\mathbb{T}^n \times \mathbb{T})$ and $\mathrm{LWE}_{\mathbf{s},\xi}$ distribution for a fixed $\mathbf{s}$ sampled from $\mathcal{S}$.
— *S*earch-LWE. Given arbitrarily many samples from the $\mathrm{LWE}_{\mathbf{s},\xi}$ distribution with fixed $\mathbf{s} \leftarrow \mathcal{S}$, recover the vector $\mathbf{s}$.

To complete the description of the LWE problem we need to choose the error distribution ξ and the distribution of the secret key $\mathcal{S}$. Following the description of the TFHE scheme, we choose $\mathcal{S}$ to be $\mathcal{U}(\{0,1\}^n)$ and ξ to be a centered continuous Gaussian distribution, i.e. , we consider the LWE problem with binary secret. In [BLP+13, Mic18], it is shown that LWE with binary secret remains hard. We use the notation $\mathrm{LWE}_{\mathbf{s},\sigma}$ as a shorthand for $\mathrm{LWE}_{\mathbf{s},\xi}$, when ξ is the Gaussian distribution centered at 0 and with the standard deviation σ.

The message space of the TFHE scheme is $\mathscr{M} = \left\{0, \frac{1}{2}\right\}$. The encryption of a message $\mu \in \mathscr{M}$ under a key $\mathscr{K} \in \{0,1\}^n$ is the sum of a random TLWE sample $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$ that corresponds to the key $\mathscr{K}$ and a pair $(\mathbf{0}, \mu) \subset \mathbb{T}^n \times \mathbb{T}$:

$$\mathrm{TLWE}_{\mathscr{K}}(\mu) := (\mathbf{a}, b + \mu) = (\mathbf{a}, \mathbf{a}^t\mathscr{K} + e + \mu), \tag{7.1}$$

where e is sampled from the distribution ξ.

Gate bootstrapping in TFHE. The simplest FHE scheme consists of a NAND gate together with a bootstrapping procedure. At a high level, the gate bootstrapping in TFHE works in the following way. It takes as input an encryption of a message μ, given by a TLWE sample $\mathrm{TLWE}_{\mathscr{K}}(\mu)$, and returns a new encryption of the same message under the same key $\mathscr{K}$, but with a fixed amount of noise. The gate bootstrapping procedure [CGGI20, Algorithm 10] has two parts: bootstrapping and key switching.

– **Bootstrapping** [CGGI20, Algorithm 9] constructs an encryption of the message μ with a fixed amount of noise, independent of the input, but under a different key $\bar{\mathscr{K}}$. It uses an intermediate encryption scheme based on the Ring-LWE problem.
– **Key switching** [CGGI20, Algorithm 2] takes as input the encryption of μ produced by bootstrapping and constructs a new encryption of μ with a fixed amount of noise under the original key $\mathscr{K}$.

Thus, the security of TFHE is based on the security of the TLWE samples, produced by key switching and TRLWE samples, used in bootstrapping. Key switching and bootstrapping use different keys of different parameters, we further refer to them as switching key and bootstrapping key, correspondingly.

Parameters and security analysis. Practically, the security of the cryptosystem is defined by the complexity of the most efficient attack against it. The security of the TFHE scheme is based on the TLWE problem, which can be seen as a scale-invariant version of LWE. In order to analyze the security of the scheme, the authors of TFHE use the dual distinguishing lattice attack on LWE, similar to the one described in [Alb17]. In [CGGI17, Section 7], the dual attack is adapted to the settings of the TFHE scheme. We recall this version of the dual attack in details in Chapter 8.

The parameters of the bootstrapping and switching keys and their security level, proposed in the first papers on TFHE [CGGI16, CGGI17], are given in Table 7.1. The security level in Table 7.1 is estimated according to the dual distinguishing attack from [CGGI17, Section 7] (see Chapter 8).

The recent journal version of the paper on TFHE [CGGI20] also contains the adaptation of the dual attack and the parameters from Table 7.1. However, in addition to these parameters, the paper introduces a new set parameters (see [CGGI20, Remark 9]). We recall the new parameters in Table 7.2. The security of the new parameters is estimated according to the LWE estimator from [ACD+18].

	n	α	λ
switching key	500	$2.43 \cdot 10^{-5}$	**159**
bootstrapping key	1024	$3.73 \cdot 10^{-9}$	198

Table 7.1 – Parameters and security of TFHE from [CGGI16, CGGI17] (see [CGGI20, Table 3]). n and α are the parameters of the underlying TLWE problem, n denotes the dimension, α denotes the parameter of the noise distribution. λ is the security level in bits. The bold number denotes the overall security of the scheme.

	n	α	λ
switching key	612	2^{-15}	**128**
bootstrapping key	1024	2^{-26}	129

Table 7.2 – Parameters and security of TFHE from [CGGI20, Table 4]. n and α are the parameters of the underlying TLWE problem, n denotes the dimension, α denotes the parameter of the noise distribution. λ is the security level in bits. The bold number denotes the overall security of the scheme.

Until very recent time, the default parameters used by the implementation of TFHE, available on-line, were a composition of the sets presented by Tables 7.1 and 7.2. The bootstrapping key parameters used by the implementation corresponded to the new parameters from Table 7.2, while the keyswitching key was still as in Table 7.1.

Then, at 21/02/2020, the parameters of the implementation were updated again. At the moment of writing this text, the parameters used by the public implementation of TFHE are as in Table 7.3 (see [G+16] for the link to the github repository of TFHE). These new parameters are quite close to the ones presented in [CGGI20, Table 4]. Their security is also estimated according to LWE estimator from [ACD+18].

	n	α	λ
switching key	630	2^{-15}	**128**
bootstrapping key	1024	2^{-25}	130

Table 7.3 – Parameters and security of TFHE from its public implementation [G+16] on GitHub. n and α are the parameters of the underlying TLWE problem, n denotes the dimension, α denotes the parameter of the noise distribution. λ is the security level in bits. The bold number denotes the overall security of the scheme.

7.2 Modular Gaussian distribution

The TFHE scheme uses the continuous centered Gaussian distribution as an error distribution ξ to produce samples from the TLWE distribution. Thus, if $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$ is a sample from the TLWE distribution corresponding to the secret $\mathbf{s} \in \{0,1\}^n$, then $e = (b - \mathbf{a}^t\mathbf{s} \mod 1) \in \mathbb{T}$ has the distribution called *modular Gaussian distribution*. In this section, we give a formal definition of this distribution and recall some useful properties.

Let $\sigma > 0$. For all $x \in \mathbb{R}$, the density of the centered Gaussian distribution with standard deviation σ is defined as $\rho_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right)$.

Definition 7.2. We define the distribution that is obtained by sampling a centered Gaus-

sian distribution of standard deviation σ and reducing it modulo 1 as the *modular Gaussian distribution* of parameter σ and denote it as $\mathscr{G}_\sigma$.

The support of the distribution is $\big(-\frac{1}{2};\frac{1}{2}\big)$. The probability density function is given by the absolutely convergent series:

$$g_\sigma(x) = \sum_{k\in\mathbb{Z}} \rho_\sigma(x+k).$$

For large values of σ, the sum that defines the density of a modular Gaussian can be closely approximated.

Lemma 7.1. As $\sigma \to \infty$, $g_\sigma(x) = 1 + 2e^{-2\pi^2\sigma^2}\cos(2\pi x) + O(e^{-8\pi^2\sigma^2})$.

More precisely, for all $x \in \big(-\frac{1}{2};\frac{1}{2}\big)$, we have

$$\Big|g_\sigma(x) - \big(1 + 2e^{-2\pi^2\sigma^2}\cos(2\pi x)\big)\Big| \leqslant e^{-8\pi^2\sigma^2} \cdot \frac{2}{1-e^{-2\pi^2\sigma^2}}.$$

Proof. The Fourier transform of the Gaussian function $\rho_{\sigma,m}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x+m)^2}{2\sigma^2}}$ is given by $\widehat{\rho}_{\sigma,m}(y) = e^{-2\pi^2\sigma^2 m^2 + 2\pi i m x}$. Then, using the Poisson summation formula, we obtain:

$$\begin{aligned} g_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma}\sum_{k\in\mathbb{Z}} e^{-\frac{(k+x)^2}{2\sigma^2}} = 1 + 2\sum_{k>0} e^{-2\pi^2\sigma^2k^2}\cos(2\pi k x) = \\ 1 + 2e^{-2\pi^2\sigma^2}\cos(2\pi x) + 2\cdot\sum_{k=2}^{+\infty} e^{-2\pi^2\sigma^2k^2}\cdot\cos(2\pi k). \end{aligned} \tag{7.2}$$

Then, we have

$$\big|g_\sigma(x) - \big(1 + 2e^{-2\pi^2\sigma^2}\cos(2\pi x)\big)\big| = 2\cdot\Big|\sum_{k=2}^{+\infty} e^{-2\pi^2\sigma^2k^2}\cdot\cos(2\pi k)\Big| \tag{7.3}$$

The absolute value of the sum from (7.3) can be bounded:

$$\Big|\sum_{k=2}^{+\infty} e^{-2\pi^2\sigma^2k^2}\cdot\cos(2\pi k)\Big| \leqslant \sum_{k=2}^{+\infty} e^{-2\pi^2\sigma^2k^2} \leqslant \sum_{k=8}^{+\infty}(e^{-2\pi^2\sigma^2})^k = \frac{e^{-8\pi^2\sigma^2}}{1-e^{-2\pi^2\sigma^2}}. \tag{7.4}$$

In (7.4), we use the following facts:

- the first inequality: $|\sum_i x_i| \leqslant \sum_i |x_i|$ and $|\cos(t)| \leqslant 1$ for all t.
- the second inequality: $\sum_{k=m}^{+\infty} x^{k^2} \leqslant \sum_{k=m^2}^{+\infty} x^k$ for all $x \in (0;1)$.

□

When the parameter σ grows, the modular Gaussian distribution becomes closer to the uniform distribution. The following lemma estimates the statistical distance between the distributions for big values of σ.

Lemma 7.2. The statistical distance between the modular Gaussian distribution $\mathscr{G}_\sigma$ and the uniform distribution on $\big(-\frac{1}{2};\frac{1}{2}\big)$ can be estimated as follows:

$$\delta(\mathscr{G}_\sigma, \mathscr{U}) = \frac{4}{\pi}\cdot e^{-2\pi^2\sigma^2} + O(e^{-8\pi^2\sigma^2}).$$

Proof. By the definition, the total variation distance between the two distributions is given by

$$\delta(\mathscr{G}_\sigma, \mathscr{U}) = \sup_{A \subset (-\frac{1}{2}; \frac{1}{2})} |\mathbb{P}_{\mathscr{G}_\sigma}\{A\} - \mathbb{P}_{\mathscr{U}}\{A\}| = \frac{1}{2} \cdot \int_{-1/2}^{1/2} |g_\sigma(x) - 1| dx. \tag{7.5}$$

Then, substituting g_σ in (7.5) with its approximation from Lemma 7.1, we get

$$\delta(\mathscr{G}_\sigma, \mathscr{U}) = \int_{-\frac{1}{2}}^{\frac{1}{2}} \left|2 \cdot e^{-2\pi^2\sigma^2} \cos(2\pi x) + O(e^{-8\pi^2\sigma^2})\right| dx = \frac{4}{\pi} \cdot e^{-2\pi^2\sigma^2} + O(e^{-8\pi^2\sigma^2}). \tag{7.6}$$

In (7.6), we used the fact that for any $x, \varepsilon \in \mathbb{R}$, $|x| - |\varepsilon| \leqslant |x + \varepsilon| \leqslant |x| + |\varepsilon|$.

□

7.3 Lattice attacks against LWE

Since the appearance of the LWE problem, various attacks against it have been proposed. Up to the best of our knowledge, for the moment there is no single best attack against all the parameters of LWE and the security of concrete instances of the LWE problem is an area of on-going research. A survey on attacks against various types of LWE can be found in [APS15]. This survey outlines three strategies for attacks against LWE: exhaustive search, BKW algorithm [BKW03, ACF+15] and lattice reduction. Lattice attacks against LWE can be separated into three categories depending on the lattice used: distinguishing dual attacks [Alb17], decoding (primal) attacks [LP11, LN13], and solving LWE by reducing it to unique-SVP [AFG13].

In this section, we recall the basic ideas behind each of the three types of lattice attacks. To be consistent with the version of the LWE problem, used in TFHE, we always consider scale-invariant version of LWE when describing the attacks.

Distinguishing dual attack. The distinguishing dual attack is the attack against search-LWE, i.e., given m samples all from the LWE-distribution, or from the uniform distribution on $\mathbb{T}^n \times \mathbb{T}$, the attack guesses the distribution of the samples.

Let $(\mathbf{a}_1, b_1), \ldots, (\mathbf{a}_m, b_m) \in \mathbb{T}^n \times \mathbb{T}$ be m samples all drawn from one of the two distributions. In a matrix form, the samples can be written as a pair $(\mathbf{A}, \mathbf{b}) \in \mathbb{T}^{n \times m} \times \mathbb{T}^m$, where $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_m)$ is a matrix whose i-th column is the vector $\mathbf{a}_i$ and $\mathbf{b} = (b_1, \ldots, b_m)^t$.

The basic version of the dual distinguishing attack (as described, e.g., in [RS10]) is based on the following idea. Assume that we can find a short linear combination of the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_m$ that sums to zero modulo 1, i.e. assume that we know a short vector $\mathbf{v} \in \mathbb{Z}^m$ such that

$$\mathbf{A}\mathbf{v} = \mathbf{0} \mod 1, \tag{7.7}$$

i.e., $\mathbf{A}\mathbf{v} \in \mathbb{Z}^m$. The vector $\mathbf{v}$ can be seen as a short vector in the lattice dual to $\mathscr{L}(\mathbf{A}^t)$. Then, consider the scalar product $\mathbf{v}^t\mathbf{b}$. If the samples $(\mathbf{A}, \mathbf{b})$ are from the LWE distribution that corresponds to a secret $\mathbf{s} \in \mathbb{Z}^n$, then we have

$$\mathbf{v}^t\mathbf{b} = \mathbf{v}^t(\mathbf{A}^t\mathbf{s} + \mathbf{e}) = \mathbf{v}^t\mathbf{e}.$$

As both $\mathbf{v}$ and $\mathbf{e}$ are short, with high probability, the product $\mathbf{v}^t\mathbf{b}$ is close to 0 modulo 1. On the other hand, if the samples are from the uniform distribution, the product $\mathbf{v}^t\mathbf{b}$ mod 1 has distribution that is very close to the uniform on $\left(-\frac{1}{2}; \frac{1}{2}\right)$.

The dual distinguishing attack has two parts: lattice reduction and distinguishing distributions.

- First, the attack recovers a short vector $\mathbf{v} \in \mathbb{Z}^m$ that satisfies (7.7) by applying a lattice reduction algorithm to the lattice dual to $\mathscr{L}(\mathbf{A}^t)$.
- The attack guesses the distribution of $\mathbf{v}^t\mathbf{b}$, i.e. distinguishes the uniform distribution on $\left(-\frac{1}{2};\frac{1}{2}\right)$ from some concentrated around zero distribution.

The complexity of the attack consists of the costs of the two parts whose complexities can be balanced. If the attack spends more time on the lattice reduction, it produces a shorter vector $\mathbf{v}$. Shorter $\mathbf{v}$ gives a more concentrated distribution of $\mathbf{v}^t\mathbf{e}$, which is easier to distinguish from the uniform in the second part of the attack.

In [CGGI17], the security of the TFHE scheme is estimated using a variation of the dual distinguishing attack, very close to the one described in [Alb17]. This version of the dual attack takes into account the small secret settings and applies techniques from BKW-like algorithms, like modulus-switching. We recall the dual attack from [CGGI17] in details in Chapter 8.

Decoding attack. The decoding (primal) attack is the attack against search-LWE, i.e., it recovers the secret key $\mathbf{s}$, given a set of the samples from the LWE distribution. The decoding attack is based on the idea that an instance of the LWE problem $(\mathbf{A}, \mathbf{b} = \mathbf{A}^t\mathbf{s}+\mathbf{e})$ can be seen as an instance of the bounded distance decoding (BDD) problem for the lattice $\mathscr{L}(\mathbf{A}^t)$.

In BDD settings, the problem can be reformulated as follows. We are given a lattice $\Lambda = \mathscr{L}(\mathbf{A}^t)$ and a vector $\mathbf{b} \in \mathbb{R}^n$. The goal is to find a lattice vector $\mathbf{v}$ such that $\|\mathbf{v} - \mathbf{b}\| \leqslant \beta$, where the bound β is chosen depending on the distribution of the error vector $\mathbf{e}$. If we recover the vector $\mathbf{v}$, most probably, it coincides with $\mathbf{A}^t\mathbf{s}$, which allows to recover the secret $\mathbf{s} = (\mathbf{A}^t)^{-1}\mathbf{v}$.

The standard method to solve BDD is to use Babai's nearest plane algorithm (see Section 3.4). First, we apply some lattice reduction algorithm to $\mathbf{A}^t$ and get a reduced basis $\mathbf{B}$ of Λ. Then, we give the basis $\mathbf{B}$ together with the target $\mathbf{b}$ to the nearest plane algorithm as input. The algorithm returns a vector $\mathbf{v} \in \Lambda$ such that the difference $\mathbf{v} - \mathbf{b}$ belongs to the fundamental parallelepiped $\mathscr{P}(\mathbf{B}^*)$, where $\mathbf{B}^*$ is the Gram-Schmidt orthogonalization of $\mathbf{B}$. Thus, if $\mathbf{b} = \mathbf{v} + \mathbf{e}$, the Babai's nearest plane algorithm returns the desired vector $\mathbf{v}$ if and only if the error vector $\mathbf{e}$ belongs to $\mathscr{P}(\mathbf{B}^*)$. The probability of successful recovery of $\mathbf{v} = \mathbf{A}^t\mathbf{s}$ is given by

$$\mathbb{P}\{\mathbf{e} \in \mathscr{P}(\mathbf{B}^*)\} = \prod_{i=1}^{d} \mathbb{P}\Big\{|\mathbf{e}^t\mathbf{b}_i^*| \leqslant \frac{\|\mathbf{b}_i^*\|^2}{2}\Big\}, \tag{7.8}$$

where d is the dimension of the lattice Λ. The probability of success depends on the length of the Gram-Schmidt vectors of the reduced basis $\mathbf{B}$, i.e., longer Gram-Schmidt vectors imply higher probability of success. Usually, a reduced basis has relatively long first Gram-Schmidt vectors, while the last Gram-Schmidt vectors are very short, which can make the probability to recover the vector $\mathbf{v}$ very low. In [LP11], Lindner and Peikert propose a generalization of Babai's nearest plane algorithm, called NearestPlanes, that allows to increase the probability of successful recovery at the cost of spending more time. The algorithm proposed in [LP11] can be seen as a version of Schnorr-Euchner's enumeration algorithm [SE94] with pruning.

NearestPlanes is recursive and works similarly to Babai's nearest plane with the only difference that at the i-the level of the recursion, the algorithm considers several nearest hyperplanes to the projection of the current target instead of one. The running time of the NearestPlanes is described by the following lemma.

Lemma 7.3. [LP11, Lemma 4.1] For $i \in \{1, \ldots, d\}$, let c_i be the number of hyperplanes that the NearestPlane considers at the i-th level. Then, the NearestPlanes outputs $\prod_{i=1}^{d} c_i$ lattice vectors that lie inside the parallelepiped $\mathbf{b} + \mathscr{P}(\mathbf{B}^*\mathbf{C})$ in time that is equal to the time of $\prod_{i=1}^{d} c_i$ runs of Babai's nearest plane algorithm, where $\mathbf{C} = \mathbf{diag}(c_1, \ldots, c_d)$.

Lindner and Peikert also proposed an heuristic estimate of the probability of success of the algorithm under the assumption that the distribution of the error can be approximated by the continuous Gaussian with standard deviation σ. Then, using (7.8), we get:

$$\mathbb{P}\{\mathbf{e} \in \mathscr{P}(\mathbf{B}^*\mathbf{C})\} = \prod_{i=1}^{d} \operatorname{erf}\Big(\frac{c_i \cdot \|\mathbf{b}_i^*\|\sqrt{\pi}}{2\sigma}\Big). \tag{7.9}$$

The complexity of the decoding attack from [LP11] is the sum of the complexity of the lattice reduction and of the NearestPlanes. As before, if the attack spends more time on the lattice reduction and get a better basis $\mathbf{B}$, the second step of the attack requires less time to achieve the same probability of successful recovery. It is quite hard to determine optimal values for all the parameters of the attack, so, in [LP11], the authors suggests a simple heuristic method for choosing the values of c_is: they choose c_is so that the value $\min(c_i\|\mathbf{b}_i^*\|)$ is maximized.

The primal attack, as the dual attack, also has several variations. For example, in [LN13], Liu and Nguyen propose a randomized variant of the NearestPlanes algorithm from [LP11]. Besides that, noting the similarities between the NearestPlane and enumeration with pruning, they propose another primal lattice attack on LWE that uses enumeration with pruning to recover the close lattice vector.

Reducing to unique-SVP. Another way to solve LWE is to reduce the underlying BDD problem, described in the previous paragraph, to the unique Shortest Vector Problem (unique-SVP). This approach was proposed by Albrecht, Fitzparick, and Göpfert in [AFG13].

In unique-SVP, we are given a basis of a lattice Λ such that there is a gap between the first and second minima of the lattice, i.e., $\frac{\lambda_2(\Lambda)}{\lambda_1(\Lambda)} \geqslant \gamma$ for some $\gamma > 1$. As in SVP, the goal is to find the shortest nonzero vector of Λ. When the gap γ between the first two minima is huge, the unique-SVP problem is much easier to solve than usual SVP. For example, if the gap γ is exponential in the dimension, the unique-SVP problem can be solved using the LLL algorithm. In practice, for subexponential values of γ, the unique-SVP can be solved using lattice reduction algorithms like BKZ with some probability of success that depends on the value of the gap. The experimental model for the dependence of the probability of success of the lattice reduction on the gap can be found in [GN08].

The LWE problem can be reduced to unique-SVP using Kannan's embedding technique [Kan87]. Kannan's technique works by embedding the lattice that corresponds to the LWE problem into a higher-dimensional lattice with a gap between the first two minima. Let $(\mathbf{A}, \mathbf{b} = \mathbf{A}^t\mathbf{s} + \mathbf{e}) \in \mathbb{T}^{n\times m} \times \mathbb{T}^m$ be m samples from the LWE distribution written in the matrix form. Let Λ be a lattice that corresponds to the LWE samples:

$$\Lambda := \{\mathbf{v} \in \mathbb{T}^m \mid \exists \mathbf{x} \in \mathbb{Z}^n \text{ such that } \mathbf{v} = \mathbf{A}^t\mathbf{x} \mod 1\}. \tag{7.10}$$

Let $\widehat{\mathbf{A}} \in \mathbb{T}^{m\times m}$ be a basis of Λ. A basis $\mathbf{B}$ for Kannan's embedding technique can be constructed as follows:

$$\mathbf{B} := \begin{pmatrix} \widehat{\mathbf{A}} & \mathbf{b} \\ \mathbf{0} & t \end{pmatrix} \in \mathbb{T}^{(m+1)\times(m+1)}. \tag{7.11}$$

If the parameter t is smaller than $\frac{\lambda_1(\Lambda)}{2\gamma}$, then the lattice $\mathscr{L}(\mathbf{B})$ has a γ-unique shortest vector (see [LM09] for the equivalence of BDD and unique-SVP), which is given by $(\mathbf{e}, -t)^t$. Thus, finding the shortest vector in $\mathscr{L}(\mathbf{B})$ allows to recover the error $\mathbf{e}$ of the LWE samples, which implies recovering of the secret $\mathbf{s}$.

The attack from [AFG13] works by applying the lattice reduction to the basis of the form given by (7.11). The complexity of the attack depends on the complexity of the lattice reduction and on the probability that the lattice reduction successfully returns a multiple of a unique shortest vector of the lattice $\mathscr{L}(\mathbf{B})$. In [AFG13], the authors estimate the complexity of the attack using an experimental model of lattice reduction algorithms.

7.4 Hybrid attacks

The idea of hybrid lattice reduction attack was introduced by Howgrave–Graham in [HG07]. He proposed to combine a meet-in-the-middle attack with lattice reduction to attack NTRUEncrypt.

Then, several works [BGPW16, Wun16] adapted Howgrave–Graham's approach to the settings of the LWE problem. These works combine the decoding (primal) lattice attack with meet-in-the-middle strategy for recovering the error vector. In [BGPW16] it is shown that this approach can be efficient for small error distributions, i.e., for binary or ternary errors.

Also, very recently, two new works [SC19, CHHS19] on the hybrid attacks against LWE have appeared. Both works are concentrated on the LWE problem with sparse and ternary secrets, which is typical for many FHE schemes. In [SC19], the authors revisit the combination of primal lattice attack based on Babai's nearest plane algorithm with meet-in-the-middle approach. In [CHHS19], the authors study the combination of the dual lattice attack from [Alb17] with the meet-in-the-middle search for the secret key. Both works show that hybrid attacks can outperform other strategies in case of ternary and sparse secret.

The attack [CHHS19] is quite close to the attack that we propose in this work, as both attacks can be seen as a combination of the dual lattice attack with the search for some part of the binary key. However, the attacks consider different species of the LWE problem: the attack from [CHHS19] is for LWE with sparse and ternary secrets, while our attack is for binary non-sparse secrets used in TFHE. This implies different search strategies for the search parts of the attacks.

7.5 Lattice reduction in practice

A lattice reduction algorithm is an algorithm which, given as input some basis of the lattice, finds a basis that consists of relatively short and relatively pairwise-orthogonal vectors. The quality of the basis produced by lattice reduction algorithms is often measured by the Hermite factor $\delta = \frac{\|\mathbf{b}_1\|}{\det(\Lambda)^{1/d}}$, where $\mathbf{b}_1$ is the first vector of the output basis. Hermite factors bigger than $\left(\frac{4}{3}\right)^{n/4}$ can be reached in polynomial time using the LLL algorithm [LLL82]. In order to obtain smaller Hermite factors, blockwise lattice reduction algorithms, like BKZ-2.0 [CN11] or S-DBKZ [MW16], can be used. The BKZ algorithm takes as input a basis of dimension d and proceeds by solving SVP on lattices of dimension $\beta < d$ using sieving [BDGL16] or enumeration [GNR10]. The quality of the output of BKZ depends on the blocksize β. In [HPS11] it is shown that after a polynomial number of calls to SVP oracle, the BKZ algorithm with blocksize β produces a basis $\mathbf{B}$ that achieves the

following bound:

$$\|\mathbf{b}_1\| \leqslant 2\gamma_\beta^{\frac{d-1}{2(\beta-1)}+\frac{3}{2}} \cdot \mathrm{vol}(\mathbf{B})^{1/d}.$$

However, up to our knowledge, there is no closed formula that tightly connects the quality and complexity of the BKZ algorithm. In this work, we use experimental models proposed in [ACF+15, ACD+18] in order to estimate the running time and quality of the output of lattice reduction. They are based on the following two assumptions on the quality and shape of the output of BKZ. The first assumption states that the BKZ algorithm outputs vectors with balanced coordinates, while the second assumption connects the Hermite factor δ with the chosen blocksize β.

Assumption 7.1. Given as input a basis B of a d-dimensional lattice Λ, BKZ outputs a vector of the norm close to $\delta^d \cdot \det(\Lambda)^{1/d}$ with balanced coordinates. Each coordinate of this vector follows a distribution that can be approximated by a Gaussian with mean 0 and standard deviation $\frac{\delta^d}{\sqrt{d}} \det(\Lambda)^{1/d}$.

Assumption 7.2. BKZ with blocksize β achieves Hermite factor

$$\delta = \left(\frac{\beta}{2\pi e}(\pi\beta)^{1/\beta}\right)^{\frac{1}{2(\beta-1)}}.$$

This assumption is experimentally verified in [Che13].

BKZ cost models. To estimate the running time of BKZ, we use three different models. The first model is an extrapolation by Albrecht [ACF+15] et al. of the Liu–Nguyen datasets [LN13]. According to that model, the logarithm of the running time of BKZ-2.0 (expressed in bit operations) is a quadratic function of $\log(\delta)^{-1}$:

$$\log(T(\mathrm{BKZ}_\delta)) = \frac{0.009}{\log(\delta)^2} - 27.$$

We further refer to this model as the delta-squared model. This model was used in [CGGI17, CGGI20] to estimate the security of TFHE.

Another more recent cost model [ACD+18] assumes that the running time of BKZ with blocksize β for d-dimensional basis is $T(\mathrm{BKZ}_{\beta,d}) = 8d \cdot T(\mathrm{SVP}_\beta)$, where $T(\mathrm{SVP}_\beta)$ is the running time of an SVP oracle in dimension β. For the SVP oracle, we use the following two widely used models:

$$\begin{aligned} \text{Sieving model:} \quad & T(\mathrm{SVP}_\beta) \approx 2^{0.292\beta+16.4}, \\ \text{Enumeration model:} \quad & T(\mathrm{SVP}_\beta) \approx 2^{0.187\beta\log(\beta)-1.019\beta+16.1}. \end{aligned}$$

The sieving algorithm [BDGL16] yields around $\left(\frac{4}{3}\right)^{\frac{n}{2}}$ short vectors while solving SVP on an n-dimensional lattice. Therefore, when using the sieving model, we shall assume that one run of the BKZ routine produces $\left(\frac{4}{3}\right)^{\frac{\beta}{2}}$ short lattice vectors, where β is the chosen blocksize.

7.6 Probability background.

In this section, for completeness, we recall two important probability inequalities used later in this part of the thesis.

Berry-Esseen inequality. The Berry-Esseen inequality shows how closely the distribution of the sum of independent random variables can be approximated by a Gaussian distribution.

Theorem 7.1. Let $X_1, \ldots, X_n$ be independent random variables such that for all $i \in \{1, \ldots, n\}$ $\mathbb{E}\{X_i\} = 0$, $\mathbb{E}\{X_i^2\} = \sigma_i^2 > 0$, and $\mathbb{E}\{|X_i|^3\} = \rho_i < \infty$. Denote the normalized sum

$$\frac{\sum_{i=1}^{n} X_i}{\sqrt{\sum_{i=1}^{n} \sigma_i^2}}$$

as S_n. Also denote by F_n the cumulative distribution function of S_n, and by Φ the cumulative distribution function of the standard normal distribution. Then, there exists a constant C_0 such that

$$\sup_{x \in \mathbb{R}} |F_n(x) - \Phi(x)| \leqslant C_0 \frac{\sum_{i=1}^{n} \rho_i}{\left(\sum_{i=1}^{n} \sigma_i^2\right)^{3/2}}.$$

We use the Berry-Esseen inequality in order to estimate how closely the distribution that we obtain after the lattice reduction step of the dual attack can be approximated by a discrete Gaussian distribution (see Lemma 8.1). The Berry-Esseen inequality requires a finite third absolute moment of the random variables. In the proof of Lemma 8.1, we need the expression of third absolute moment of a Gaussian distribution. It can be obtained from the following lemma.

Lemma 7.4. Let $\sigma > 0$. Let X be a random variable of a Gaussian distribution with mean 0 and standard deviation σ^2. Then, $\mathbb{E}\{|X|^3\} = 2\sqrt{\frac{2}{\pi}}\sigma^3$.

Proof. Classically we have:

$$\mathbb{E}\{|X|^3\} = 2 \cdot \frac{1}{\sqrt{2\pi\sigma}} \int_0^\infty x^3 e^{-\frac{x^2}{2\sigma^2}} dx = 2\sqrt{\frac{2}{\pi}}\sigma^3.$$

□

Hoeffding's inequality. Hoeffding's inequality gives an exponentially decreasing upper bound on the probability that the sum of bounded independent random variables deviates from its expectation by a certain amount.

Theorem 7.2. Let $X_1, \ldots, X_N$ be independent random variables such that $a_i \leqslant X_i \leqslant b_i$ for all $i \in \{1, \ldots, N\}$. Denote the average $\frac{1}{N}\sum_{i=1}^{N} X_i$ as $\bar{X}$. Then, for $t > 0$, we have

$$\mathbb{P}\{\bar{X} - \mathbb{E}\{\bar{X}\} \geqslant t\} \leqslant \exp\left(-\frac{2N^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right), \tag{7.12}$$

$$\mathbb{P}\{\bar{X} - \mathbb{E}\{\bar{X}\} \leqslant -t\} \leqslant \exp\left(-\frac{2N^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right). \tag{7.13}$$

In this paper, we use Hoeffding's inequality to construct a distinguisher for the uniform and the modular Gaussian distributions (see Section 8.1.2).

Chapter 8

Hybridizing the dual distinguishing attack against LWE.

In this chapter, we revisit the distinguishing dual attack against LWE described in [CGGI20]) and generalize it by combining the dual approach with the search for a fraction of the secret key. First, in Section 8.1, we recall the dual attack from [CGGI20], providing complete proofs and introducing finer tools as a novel distinguisher for the uniform distribution and the modular Gaussian. Then, in Section 8.2, we construct a more efficient hybrid attack on top of the dual distinguishing attack from [CGGI20].

8.1 Dual distinguishing attack as described in [CGGI20]

Setting. Let $\mathbf{s} \in \{0,1\}^n$ be a secret vector and let $\alpha > 0$ be a fixed constant. The attack takes as input m samples $(\mathbf{a}_1, b_1), \ldots, (\mathbf{a}_m, b_m) \in \mathbb{T}^{n+1} \times \mathbb{T}$ which are either all from $\mathrm{LWE}_{\mathbf{s},\alpha}$ distribution or all from $\mathscr{U}(\mathbb{T}^n \times \mathbb{T})$, and guesses the input distribution.

We can write input samples in a matrix form:

$$\mathbf{A} := (\mathbf{a}_1, \ldots, \mathbf{a}_m) \in \mathbb{T}^{n \times m}, \quad \mathbf{b} = (b_1, \ldots, b_m)^t \in \mathbb{T}^m,$$

if input samples are from the $\mathrm{LWE}_{\mathbf{s},\alpha}$ distribution:

$$\mathbf{b} = \mathbf{A}^t\mathbf{s} + \mathbf{e} \mod 1.$$

Distinguisher reduction using a small trapdoor. In order to distinguish between the two distributions, the attack searches for a short vector $\mathbf{v} = (v_1, \ldots, v_m)^t \in \mathbb{Z}^m$ such that the linear combination of the left parts of the inputs samples defined by $\mathbf{v}$, i.e.:

$$\mathbf{x} := \sum_{i=1}^{m} v_i \mathbf{a}_i = \mathbf{A}\mathbf{v} \mod 1$$

is also a short vector. If the input was from the LWE distribution, then the corresponding linear combination of the right parts of the input samples is also small as a sum of two relatively small numbers:

$$\mathbf{v}^t\mathbf{b} = \mathbf{v}^t(\mathbf{A}^t\mathbf{s} + \mathbf{e}) = \mathbf{x}^t\mathbf{s} + \mathbf{v}^t\mathbf{e} \mod 1. \tag{8.1}$$

On the other hand, if the input is uniformly distributed, then independently of the choice of the non-zero vector $\mathbf{v}$, the product $\mathbf{v} \cdot \mathbf{b} \mod 1$ has uniform distribution on $(-1/2; 1/2)$. Recovering a suitable $\mathbf{v}$ thus turns the decisional-LWE problem into an easier problem of distinguishing two distributions on $\mathbb{T}$.

This remaining of section is organized in the following way. First, in Section 8.1.1 we describe how such a suitable vector $\mathbf{v}$ can be discovered by lattice reduction and analyze the distribution of $\mathbf{v}^t\mathbf{b}$. Then, in Section 8.1.2, we estimate the complexity of distinguishing two distributions on $\mathbb{T}$ that we obtain after this first part. Eventually, in Section 8.1.3 we estimate the time complexity of the whole attack.

8.1.1 Trapdoor construction by lattice reduction

Finding a vector $\mathbf{v}$ such that both parts of the sum (8.1) are small when the input has LWE distribution is equivalent to finding a short vector in the following $(m+n)$-dimensional lattice:

$$\mathscr{L}(\mathbf{A}) = \left\{ \begin{pmatrix} \mathbf{A}\mathbf{v} & \mod 1 \\ \mathbf{v} & \end{pmatrix} \in \mathbb{R}^{m+n} \,\middle|\, \forall\, \mathbf{v} \in \mathbb{Z}^m \right\}.$$

The lattice $\mathscr{L}(\mathbf{A})$ can be generated by the columns of the following matrix:

$$\mathbf{B} = \begin{pmatrix} \mathbf{I}_n & \mathbf{A} \\ \mathbf{0}^{m\times n} & \mathbf{I}_m \end{pmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}$$

A short vector in $\mathscr{L}(\mathbf{A})$ can be found by applying a lattice reduction algorithm to the basis $\mathbf{B}$. Using Assumption 7.1, we expect that the lattice reduction process produces a vector $\mathbf{w} = (\mathbf{x}||\mathbf{v})^t \in \mathbb{Z}^{n+m}$ with equidistributed coordinates. Our goal is to minimize the product $\mathbf{v}^t\mathbf{b} = \mathbf{x}^t\mathbf{s} + \mathbf{v}^t\mathbf{e}$. The vectors $\mathbf{e}$ and $\mathbf{s}$ come from different distributions and have different expected norms. For the TFHE scheme, the variance of $\mathbf{e}$ is much smaller than the variance of $\mathbf{s}$. To take this imbalance into account, one introduces an additional rescaling parameter $q \in \mathbb{R}_{>0}$. The first n rows of the matrix $\mathbf{B}$ are multiplied by q, the last m rows are multiplied by $q^{-n/m}$. Odiously, this transformation doesn't change the determinant of the matrix. A basis $\mathbf{B}_q$ of the transformed lattice is given by

$$\mathbf{B}_q = \begin{pmatrix} q\mathbf{I}_n & q\mathbf{A} \\ \mathbf{0}^{m\times n} & q^{-n/m}\mathbf{I}_m \end{pmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}.$$

We apply a lattice reduction algorithm to $\mathbf{B}_q$. Denote the first vector of the reduced basis as $\mathbf{w}_q$. By taking last m coordinates of $\mathbf{w}_q$ and multiplying them by $q^{n/m}$ we recover the desired vector $\mathbf{v}$. This technique can be thought as a continuous relaxation of the modulus switching technique. That part of the attack is summarized in Algorithm 8.1.

Algorithm 8.1: Transform m LWE samples to one sample from modular Gaussian distribution

input : $\mathbf{A} \in \mathbb{T}^{n\times m}$, $\mathbf{b} \in \mathbb{T}^m$, $S > 0$, $\alpha > 0$, $\delta \in (1; 1.1)$
output: $x \in \mathbb{T}$

`computeV(`$\mathbf{A}$, S, α, δ`):`
- $q := \left(\frac{S}{\alpha}\right)^{\frac{m}{n+m}}$
- $\mathbf{B}_q := \begin{pmatrix} q\mathbf{I}_n & q\mathbf{A} \\ \mathbf{0}^{m\times n} & q^{-n/m}\mathbf{I}_m \end{pmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}$
- $\mathbf{w}_q \leftarrow \mathrm{BKZ}_\delta(\mathbf{B}_q)$
- $\mathbf{v} := q^{n/m} \cdot (w_{q_{n+1}}, \dots w_{q_{n+m}})^t$
- **return** $(\mathbf{v})$

`LWEtoModGaussian(`$\mathbf{A}$, $\mathbf{b}$, S, α, δ`):`
- $\mathbf{v} \leftarrow$ COMPUTEV$(\mathbf{A}, S, \alpha, \delta)$
- **return** $\mathbf{v}^t\mathbf{b} \mod 1$

The following lemma describes the distribution of the output of Algorithm 8.1 under Assumption 7.1 that BKZ outputs vectors with balanced coordinates.

Lemma 8.1 (see [CGGI20, Section 7])**.** Let $\alpha > 0$ and $S \in (0;1)$ be fixed constants, $n \in \mathbb{Z}_{>0}$. Let $\mathbf{s} \in \{0,1\}^n$ be a binary vector such that all bits of $\mathbf{s}$ are sampled independently from the Bernoulli distribution with parameter S^2: for all $i \in \{1,\dots,n\}$: $\mathbb{P}\{s_i = 1\} = S^2$, $\mathbb{P}\{s_i = 0\} = 1 - S^2$. Suppose that Assumption 7.1 holds and let $\delta > 0$ be the quality of the output of the BKZ algorithm. Then, given as input $m = \sqrt{n \cdot \frac{\ln(S/\alpha)}{\ln(\delta)}} - n$ samples from the $\mathrm{LWE}_{\mathbf{s},\alpha}$ distribution, Algorithm 8.1 outputs a random variable x with distribution that can be approximated by a Gaussian distribution with mean 0 and standard deviation σ

$$\sigma = \alpha \cdot \exp\Big(2\sqrt{n \ln(S/\alpha) \ln(\delta)}\Big).$$

Denote as F_x the cumulative distribution function of x and denote as Φ_σ the cumulative distribution function of the Gaussian distribution with mean 0 and standard deviation σ. Then, the distance between the two distributions can be bounded:

$$\sup_{t\in\mathbb{R}} |F_x(t) - \Phi_\sigma(t)| = O\Big(\frac{1}{\sqrt{S^2(m+n)}}\Big),$$

as $n \to \infty$.

Lemma 8.1 can be proved using the Berry-Esseen theorem. We give a proof in Appendix A.1 for completeness.

8.1.2 Exponential kernel distinguisher for the uniform and the modular Gaussian distributions

We now describe a novel distinguisher for the uniform and the modular Gaussian distributions. Formally, we construct a procedure which takes as input N samples which are all sampled independently from one of the two distributions and guesses this distribution.

The crux of our method relies on the use of an empirical estimator of the Levy transform of the distributions, to essentially cancel the effect of the modulus 1 on the Gaussian. Namely, from the N samples $X_1,\dots,X_N$, we construct the estimator $\bar{Y} = \frac{1}{N} \cdot \sum_{i=1}^{N} \mathrm{e}^{2\pi i X_i}$. As N is growing to infinity, this estimator converges to the Levy transform at 0 of the underlying distribution, that is to say:

— to 0 for the uniform distribution

— to $e^{-2\pi^2\sigma^2}$ for the modular Gaussian.

Hence, in order to distinguish the distribution used to draw the samples, we now only need to determine whether the empirical estimator $\bar{Y}$ is closer to 0 or to $e^{-2\pi^2\sigma^2}$.

Remark 1. The optimal value for the corresponding threshold can be obtained as a log-likelihood estimator. However, this optimization is not giving a close formula. It appears that the gains obtained from a numerical optimization of this value are negligible compared to taking the natural threshold of $\frac{1}{2} \cdot e^{-2\pi^2\sigma^2}$.

Lemma 8.2. Let $\sigma > 0$ be a fixed constant. Assume that Algorithm 8.2 is given as input N points that are sampled independently from the uniform distribution $\mathscr{U}$ or from the modular Gaussian distribution $\mathscr{G}_\sigma$. Then, Algorithm 8.2 guesses the distribution of the input points correctly with probability at least

$$p_\sigma = 1 - \exp\Big(-\frac{e^{-4\pi^2\sigma^2}}{8} \cdot N\Big). \tag{8.2}$$

The time complexity of the algorithm is polynomial in the size of the input.

Algorithm 8.2: Distinguish $\mathscr{U}$ and $\mathscr{G}_\sigma$

input : $X_1, \ldots, X_N \in \left(-\frac{1}{2}; \frac{1}{2}\right)$, $\sigma > 0$, sampled independently from $\mathscr{U}$ or $\mathscr{G}_\sigma$
output: A guess: G if the samples are drawn under $\mathscr{G}_\sigma$ or U otherwise

1 `DistinguishGU(`$X_1, \ldots, X_N, \sigma$`):`
2 $\quad \bar{Y} = \frac{1}{N} \cdot \sum_{i=1}^{N} \exp(2\pi i X_i)$
3 $\quad$ **if** $(\bar{Y} \leqslant \frac{1}{2} \cdot e^{-2\pi^2\sigma^2})$ **then**
4 $\quad\quad$ **return** U
5 $\quad$ **else**
6 $\quad\quad$ **return** G

Proof. For all $i \in \{1, \ldots, N\}$, denote $e^{2\pi i X_i}$ as Y_i. As $X_i \in \left(-\frac{1}{2}, \frac{1}{2}\right)$, $\Re(Y_i) \in (-1; 1]$.

First, we compute the expectation of $\bar{Y} = \frac{1}{N} \cdot \sum_{i=1}^{N} Y_i$ in the two possible cases where X_is are sampled from the uniform distribution, and where X_is are sampled from the modular Gaussian with standard deviation σ. Note that, in both cases, as X_is are sampled independently and identically from the same distribution, $\mathbb{E}\{\bar{Y}\} = \mathbb{E}Y_i$.

In case of the uniform distribution, the expectation of the real part of $\bar{Y}$ is equal to zero, because the function $\Re(e^{2\pi i x})$ is symmetric around the origin:

$$\mathbb{E}_U\{\Re(\bar{Y})\} = \int_{-1/2}^{1/2} e^{2\pi i x} dx = 0. \tag{8.3}$$

Now in case of the modular Gaussian distribution, we exploit the 1-periodicity of $t \mapsto e^{2i\pi t}$ to cancel out the modulus 1:

$$\mathbb{E}_G\{\bar{Y}\} = \int_{-1/2}^{+1/2} e^{2\pi i x} \sum_{k\in\mathbb{Z}} \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{(x+k)^2}{2\sigma^2}} dx \tag{8.4}$$

$$= \sum_{k\in\mathbb{Z}} \int_{-1/2}^{+1/2} e^{2\pi i x} \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{(x+k)^2}{2\sigma^2}} dx \tag{8.5}$$

$$= \int_{-\infty}^{+\infty} e^{2\pi i x} \cdot \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{x^2}{2\sigma^2}} dx \tag{8.6}$$

$$= e^{-2\pi^2\sigma^2} \cdot \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} e^{-\frac{(x-2i\pi\sigma)^2}{2\sigma^2}} dx = e^{-2\pi^2\sigma^2}, \tag{8.7}$$

the sum-integral exchange being justified by uniform convergence of the sum.

Now, using the expectations of $\bar{Y}$ and the Hoeffding's inequality, we can estimate the probability of getting a correct guess.

First, consider the probability wrongly guessing when the distribution of the input is uniform. By Line 3 of Algorithm 8.2, it is given by:

$$\mathbb{P}\{G|U\} = \mathbb{P}_U\{\bar{Y} > \frac{1}{2} \cdot e^{2\pi^2\sigma^2}\}.$$

Since Y_is are bounded, i.e., for all $i \in \{1, \dots, N\}$, $Y_i \in (-1; 1]$, we can use Hoeffding's inequality (see Theorem 7.2) to bound the probability $\mathbb{P}\{G|U\}$:

$$\mathbb{P}\{G|U\} \leqslant \exp\Big(-\frac{e^{-4\pi^2\sigma^2}}{8} \cdot N\Big). \tag{8.8}$$

Similarly, we get the same bound on the probability of the wrong guess when the distribution of the input is the modular Gaussian:
$\mathbb{P}\{U|G\} \leqslant \exp\Big(-\frac{e^{-4\pi^2\sigma^2}}{8} \cdot N\Big)$. Together with (8.8), we get the bound on the probability of the successful guess, given by (8.2).

Since Algorithm 8.2 consists of computing the average of the input sample and performing one comparison, it is polynomial in the size of the input. □

Lemma 8.2 implies that in order to distinguish the uniform distribution and the modular Gaussian distribution with the parameter σ with a non-negligible probability, we need to take a sample of size $N = O(e^{4\pi^2\sigma^2})$.

Remark 2. The original dual attack, proposed in [CGGI20], does not specify, which algorithm is used for distinguishing the uniform and the modular Gaussian distributions. Instead, to estimate the size of the sample, needed to distinguish the distributions, they estimate the statistical distance ε between the distributions $\mathscr{U}$ and $\mathscr{G}_\sigma$ (see Lemma 7.2 for an estimate for ε) and use $O(1/\varepsilon^2)$ as an estimate for the required size of the sample (see [CGGI20, Section 7, Equation(3),(6)]). Such an estimate is however asymptotic and does not allow a practical instantiation in the security analysis.

It turns out that the exponential kernel distinguisher, described in Algorithm 8.2, (ignoring some constant factors), has the same complexity as the statistical distance estimate from [CGGI20] suggests, while enjoying a sufficiently precise analysis to provide non-asymptotic parameters estimation.

8.1.3 Complexity of the dual attack from TFHE article

The distinguishing attack is summarized in Algorithm 8.3. It takes as input $m \times N$ samples from an unknown distribution, then transforms them into N samples which have the uniform distribution if the input of the attack was uniform and the modular Gaussian distribution if the input was from the LWE distribution. Then, the attack guesses the distribution of N samples using Algorithm 8.2 and outputs the corresponding answer.

Algorithm 8.3: Dual distinguishing attack (adapted from [CGGI20, Section 7])

input : $\{(\mathbf{A}_i, \mathbf{b}_i)\}_{i=1}^N$, where $\forall i$ $\mathbf{A}_i \in \mathbb{T}^{n\times m}$, $\mathbf{b}_i \in \mathbb{T}^m$, $\alpha > 0$, $S > 0$, $\delta \in (1; 1.1)$
output: guess for the distribution of the input: Uniform or LWE distribution

```
DistinguishingAttack({A_i, b_i}_{i=0}^N, α, S, δ):
    X := ∅
    σ := α · exp(2√(n ln(S/α) ln(δ)))
    for i ∈ {1, ..., N} do
        x ← LWEtoModGaussian(A_i, b_i, S, α, δ)
        X ← X ∪ x
    if (DistinguishGU(X, σ) = 𝒢) then
        return LWE distribution
    else
        return Uniform
```

The following theorem states that the cost of the distinguishing attack can be estimated by solving a minimization problem. It revisits the estimate given in [CGGI20, Section 7].

Theorem 8.1. Let $\alpha > 0$ and $S \in (0;1)$ be some fixed constants, $n \in \mathbb{Z}_{>0}$. Let $\mathbf{s} \in \{0,1\}^n$ be a binary vector such that all bits of $\mathbf{s}$ are sampled independently from a Bernoulli distribution with parameter S^2. Suppose that Assumption 7.1 holds. Then, the time complexity of solving Decision-LWE$_{\mathbf{s},\alpha}$ with probability of success p by the distinguishing attack described in Algorithm 8.3 is

$$T_{\text{TFHEattack}} = \min_{\delta} \Big(N(\sigma, p) \cdot T(\text{BKZ}_\delta) \Big), \tag{8.9}$$

where $\sigma = \alpha \cdot \exp\left(2\sqrt{n \ln(S/\alpha) \ln(\delta)}\right)$, $N(\sigma, p) = 8 \ln(\frac{1}{1-p}) \cdot e^{4\pi^2\sigma^2}$.

Proof. The cost of the attack is the cost of the lattice reduction multiplied by the number of samples N needed to distinguish the uniform distribution and the modular Gaussian distribution with the parameter σ:

$$T = N \cdot T(\text{BKZ}_\delta). \tag{8.10}$$

By Lemma 8.2, Algorithm 8.2, given as an input a sample of size N, guesses its distribution correctly with the probability at least $1 - \exp\left(-N \cdot \frac{e^{-4\pi^2\sigma^2}}{8}\right)$. Thus, in order to achieve the probability p, we need to produce a sample of size $N(\sigma, p) = 8 \ln(\frac{1}{1-p}) \cdot e^{4\pi^2\sigma^2}$.

The parameter σ of the discrete Gaussian distribution as a function of δ can be estimated using Lemma 8.1. Then, the time complexity can be obtained by optimizing the expression, given by (8.10), as a function of δ. □

8.2 Hybrid key recovery attack

In this section, we show how the dual distinguishing attack recalled in Chapter 8 can be hybridized with exhaustive search on a fraction of the secret vector to obtain a continuum of more efficient key recovery attacks on the underlying LWE problem. Let $\mathbf{s} \in \{0,1\}^n$ be a secret vector and let $\alpha > 0$ be a fixed constant. Our attack takes as input samples from the LWE distribution of form

$$(\mathbf{A}, \mathbf{b} = \mathbf{A}^t\mathbf{s} + \mathbf{e} \mod 1) \in (\mathbb{T}^{n\times m}, \mathbb{T}^m), \tag{8.11}$$

where $\mathbf{e} \in \mathbb{R}^m$ has centered Gaussian distribution with standard deviation α. The attack divides the secret vector into two fractions:

$$\mathbf{s} = (\mathbf{s}_1 || \mathbf{s}_2)^t, \quad \mathbf{s}_1 \in \{0,1\}^{n_1}, \quad \mathbf{s}_2 \in \{0,1\}^{n_2}, \quad n = n_1 + n_2.$$

The matrix $\mathbf{A}$ is also fractionned into two parts corresponding to the separation of the secret $\mathbf{s}$:

$$\mathbf{A} = \left(\begin{array}{ccc} a_{1,1} & \dots & a_{1,m} \\ \vdots & & \vdots \\ a_{n_1,1} & \dots & a_{n_1,m} \\ \hline a_{n_1+1,1} & \dots & a_{n_1+1,m} \\ \vdots & \dots & \vdots \\ a_{n,1} & \dots & a_{n,m} \end{array}\right) = \begin{pmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{pmatrix} \tag{8.12}$$

Then, (8.11) can be rewritten as

$$\mathbf{A}_1^t\mathbf{s}_1 + \mathbf{A}_2^t\mathbf{s}_2 + \mathbf{e} = \mathbf{b} \mod 1.$$

By applying lattice reduction to matrix $\mathbf{A}_1$ as described in Algorithm 8.1, we recover a vector $\mathbf{v}$ such that $\mathbf{v}^t(\mathbf{A}_1^t\mathbf{s}_1+\mathbf{e})$ is small and it allows us to transforms m input LWE samples $(\mathbf{A}, \mathbf{b}) \in (\mathbb{T}^{n\times m}, \mathbb{T}^m)$ into one new LWE sample $(\widehat{\mathbf{a}}, \widehat{b}) \in (\mathbb{T}^{n_2}, \mathbb{T})$ of smaller dimension and bigger noise:

$$\underbrace{\mathbf{v}^t\mathbf{A}_2^t}_{\mathbf{a}}\mathbf{s}_2 + \underbrace{\mathbf{v}^t(\mathbf{A}_1^t\mathbf{s}_1 + \mathbf{e})}_{\widehat{e}} = \underbrace{\mathbf{v}^t\mathbf{b}}_{\widehat{b}} \mod 1. \tag{8.13}$$

The resulting LWE sample in smaller dimension can be used to find $\mathbf{s}_2$. Let $\mathbf{x} \in \{0,1\}_2^n$ be a guess for $\mathbf{s}_2$. If the guess is correct, then the difference

$$\widehat{b} - \widehat{\mathbf{a}}^t\mathbf{x} = \widehat{b} - \widehat{\mathbf{a}}^t\mathbf{s}_2 = (\widehat{e} \mod 1) \sim \mathscr{G}_\sigma \tag{8.14}$$

is small.

If the guess is not correct and $\mathbf{x} \neq \mathbf{s}_2$, then there exist some $\mathbf{y} \neq \mathbf{0}$ such that $\mathbf{x} = \mathbf{s}_2 - \mathbf{y}$. Then, we rewrite $\widehat{b} - \widehat{\mathbf{a}}^t\mathbf{x}$ in the following way:

$$\widehat{b} - \widehat{\mathbf{a}}^t\mathbf{x} = (\widehat{b} - \widehat{\mathbf{a}}^t\mathbf{s}_2) + \widehat{\mathbf{a}}^t\mathbf{y} = \widehat{\mathbf{a}}^t\mathbf{y} + \widehat{e}.$$

We can consider $(\widehat{\mathbf{a}}, \widehat{\mathbf{a}}^t\mathbf{y} + \widehat{e})$ as a sample from the LWE distribution that corresponds to the secret $\mathbf{y}$. Therefore, we may assume that if $\mathbf{x} \neq \mathbf{s}_2$, the distribution of $\widehat{b} - \widehat{\mathbf{a}}^t\mathbf{x} \mod 1$ is close to uniform, unless the decision-LWE is easy to solve.

In order to recover $\mathbf{s}_2$, the attack generates many LWE samples with reduced dimension. Denote by R the number of generated samples and put them into matrix form as $(\widehat{\mathbf{A}}, \widehat{\mathbf{b}}) \in \mathbb{T}^{n_2\times R} \times \mathbb{T}^R$. There are 2^{n_2} possible candidates for $\mathbf{s}_2$. For each candidate $\mathbf{x} \in \{0,1\}^{n_2}$, the attack computes an R-dimensional vector $\mathbf{e}_\mathbf{x} = \mathbf{b} - \mathbf{A}^t\mathbf{s}$. The complexity of this computation for all the candidates is essentially the complexity of multiplying the matrices $\widehat{\mathbf{A}}$ and $\mathbf{S}_2$, where $\mathbf{S}_2$ is a matrix whose columns are all binary vectors of dimension n_2. Naively, the matrix multiplication requires $O(n\cdot 2^{n_2}\cdot R)$ operations. However, by exploiting the recursive structure of $\mathbf{S}_2$, it can be done in time $O(R\cdot 2^{n_2})$.

Then, for each candidate $\mathbf{x}$ for $\mathbf{s}_2$ the attack checks whether the corresponding vector $\mathbf{e}_\mathbf{x}$ is uniform or concentrated around zero distribution. The attack returns the only candidate $\mathbf{x}$ whose corresponding vector $\mathbf{e}_\mathbf{x}$ has concentrated around zero distribution.

The rest of this chapter is organized as follows. First, we describe the auxiliary algorithm for multiplying a matrix by the matrix of all binary vectors that let us speed up the search for the second fraction of the secret key. Then, we evaluate the complexity of our attack.

8.2.1 Algorithm for computing the product of a matrix with the matrix of all binary vectors

For any $d \in \mathbb{Z}_{>0}$, define the function $\mathbf{bin}_d : \mathbb{Z} \cap [0; 2^d] \to \{0,1\}^d$ that maps any positive integer $k \leqslant 2^d$ to $\mathbf{bin}_d(k)$ the d-dimensional binary vector obtained from the binary representation of k.

For any positive integer d, denote by $\mathbf{S}_{(d)}$ the matrix of all binary vectors of dimension d, in lexicographic order. Thus, the i-th column of $\mathbf{S}_{(d)}$ is equal to $\mathbf{bin}_d(i)$. These matrices can be constructed recursively. For $d = 1$ it is given by $\mathbf{S}_{(1)} = \begin{pmatrix}0 & 1\end{pmatrix}$, and for any $d > 1$ the matrix $\mathbf{S}_{(d)}$ can be constructed by concatenating two copies of the matrix $\mathbf{S}_{(d-1)}$ and adding a row which consists of 2^{d-1} zeros followed by 2^{d-1} ones as the first row to the resulting matrix:

$$\mathbf{S}_{(d)} = \begin{pmatrix}0\ldots0 & 1\ldots1\\ \mathbf{S}_{(d-1)} & \mathbf{S}_{(d-1)}\end{pmatrix}. \tag{8.15}$$

Let $\mathbf{a} = (a_1, \ldots, a_d)^t$ be a d-dimensional vector. Our goal is to compute the scalar products of $\mathbf{a}$ with every column of $\mathbf{S}_{(d)}$. We can do it by using the recursive structure

of $\mathbf{S}_{(d)}$. Assume that we know the desired scalar products for $\mathbf{a}_{(d-1)} = (a_2, \ldots, a_d)^t$ and $\mathbf{S}_{(d-1)}$ Then, using (8.15), we get

$$\mathbf{a}^t\mathbf{S}_{(d)} = \begin{pmatrix} a_1 & \mathbf{a}^t_{(d-1)} \end{pmatrix} \cdot \begin{pmatrix} 0\ldots0 & 1\ldots1 \\ \mathbf{S}_{(d-1)} & \mathbf{S}_{(d-1)} \end{pmatrix} = \begin{pmatrix} \mathbf{a}^t_{(d-1)}\mathbf{S}_{(d-1)} \\ (a_1 \ldots a_1)^t + \mathbf{a}^t_{(d-1)}\mathbf{S}_{(d-1)} \end{pmatrix}, \tag{8.16}$$

that is, the resulting vector is the sum of the vector $\mathbf{a}^t_{(d-1)}\mathbf{S}_{(d-1)}$ concatenated with itself with the vector whose first 2^{d-1} coordinates are zeros and the last 2^{d-1} coordinates are all equal to a_1. The approach is summarized in Algorithm 8.4.

Algorithm 8.4: Compute a scalar product of a vector with all binary vectors

input : $\mathbf{a} = (a_1, \ldots, a_d)^t$
output: $\mathbf{a}^t\mathbf{S}_{(d)}$, where $\mathbf{S}_{(d)} \in \{0,1\}^{2^d \times d}$ is the matrix whose columns are all binary vectors of dimension d written in the lexicographical order

```
1 computeScalarProductWithBinaryVectors(a):
2     x ← (0, a_d)^t
3     for i ∈ {d − 1, ..., 1} do
4         y ← x
5         for j ∈ {1, ..., 2^{d−i}} do
6             y_j ← y_j + a_i
7         x' ← x ∪ y
8         x ← x'
9     return x
```

Lemma 8.3. Let d be a positive integer number. Algorithm 8.4, given as input a d-dimensional vector $\mathbf{a}$, outputs the vector $\mathbf{x}$ of dimension 2^d such that for all $i \in \{1, \ldots, 2^d\}$ $x_i = \mathbf{a}^t\mathbf{bin}_d(i)$. The time complexity of the algorithm is $O(2^d)$.

Proof. The correctness of the algorithm follows from the recursive structure of the matrix $\mathbf{S}_{(d)}$ (see (8.15) and (8.16)). The algorithm performs only additions of some coordinates of the vector $\mathbf{a}$. At the i-th iteration of the cycle (3-8) the algorithm performs 2^{d-i} additions. Number of iterations is $(d-1)$. The overall number of additions is $2 + 2^2 + \cdots + 2^{d-1} = 2^d - 2$. □

Corollary 8.1. Let $\mathbf{A}$ be a matrix with R rows and d columns. The product of $\mathbf{A}$ and $\mathbf{S}_{(d)}$ can be computed in time $O(R \cdot 2^d)$.

Proof. In order to compute $\mathbf{A} \cdot \mathbf{S}_{(d)}$ we need to compute the product of each of the R rows of A with $\mathbf{S}_d$. By Lemma 8.3 it can be done in time $O(2^d)$. Then the overall complexity of multiplying the matrices is $O(R \cdot 2^d)$. □

8.2.2 Complexity of the attack

The pseudo-code corresponding to the full attack is given in Algorithm 8.5.

Theorem 8.2. Let $\alpha > 0$, $p \in (0;1)$, $S \in (0;1)$, and $n \in \mathbb{Z}_{>0}$ be fixed constants. Let $\mathbf{s} \in \{0,1\}^n$ and $\sigma > 0$. Suppose that Assumption 7.1 holds. Then, the time complexity of solving the Search-LWE$_{\mathbf{s},\alpha}$ problem with probability of success p by the attack described in Algorithm 8.5 is

$$T_{\text{attack}} = \min_{\delta, n_2} \left(\left(2^{n_2} + T(\text{BKZ}_\delta)\right) \cdot R(n_2, \sigma, p) \right), \tag{8.17}$$

where $R(n_2, \sigma, p) = 8 \cdot e^{4\pi^2\sigma^2}(n_2 \ln(2) - \ln(\ln(1/p)))$.

Algorithm 8.5: Hybrid key recovery attack

input : $\{(\mathbf{A}_i, \mathbf{b}_i)\}_{i=1}^{R}$, where $\forall i\ \mathbf{A}_i \in \mathbb{T}^{n\times m}$, $\mathbf{b}_i \in \mathbb{T}^m$, $\alpha > 0$, $S > 0$, $\delta > 1$, $n_1 \in \{2, \ldots, n-1\}$

output: $\mathbf{s}_2 \in \{0,1\}^{n-n_1}$

`recoverS(`$\{(\mathbf{A}_i, \mathbf{b}_i)\}_{i=1}^{R}, \alpha, S, \delta, n_1$`):`

$n_2 \leftarrow (n - n_1)$

$\sigma \leftarrow \alpha \cdot \exp\left(2\sqrt{n_1 \ln(S/\alpha)\ln(\delta)}\right)$

$\widehat{\mathbf{A}} \leftarrow \varnothing$, $\widehat{\mathbf{b}} \leftarrow \varnothing$

`/* lattice reduction part */`

for $i \in \{1, \ldots, R\}$ **do**

  $\mathbf{A} \leftarrow \mathbf{A}_i$, $\mathbf{b} \leftarrow \mathbf{b}_i$

  $(\mathbf{A}_1, \mathbf{A}_2) \leftarrow \text{SPLITMATRIX}(\mathbf{A}, n_1)$ ▷ see (8.12)

  $\mathbf{v} \leftarrow \text{COMPUTEV}(\mathbf{A}_1, S, \alpha, \delta)$ ▷ Algorithm 8.1

  $\widehat{\mathbf{A}} \leftarrow \widehat{\mathbf{A}} \cup \{\mathbf{A}_2\mathbf{v}\}$, $\widehat{\mathbf{b}} \leftarrow \widehat{\mathbf{b}} \cup \{\mathbf{v}^t\mathbf{b}\}$

`/* search for` $\mathbf{s}_2$ `*/`

$\mathbf{S}_{(n_2)} \leftarrow$ matrix of all binary vectors of dimension n_2 in lexicographical order

$\widehat{\mathbf{B}} \leftarrow (\widehat{\mathbf{b}}, \ldots, \widehat{\mathbf{b}}) \in \mathbb{T}^{R\times 2^{n_2}}$

$\widehat{\mathbf{E}} \leftarrow \widehat{\mathbf{B}} - \widehat{\mathbf{A}}^t\mathbf{S}_{(n_2)} \mod 1$ ▷ see Corollary 8.1 and Algorithm 8.4

for $i \in \{1, \ldots, 2^{n_2}\}$ **do**

  $\widehat{\mathbf{e}} \leftarrow \widehat{\mathbf{E}}[i]$

  `/* guess the distribution of` e `(see Algorithm 8.2) */`

  if $(\text{DISTINGUISHGU}(\widehat{\mathbf{e}}, \sigma) = \mathscr{G})$ **then**

    return $\mathbf{S}_{(n_2)}[i]$

Proof. The attack can be divided in two steps: the lattice reduction step and the exhaustive search for the second fraction of the secret key. The first step of the attack takes $R \times m$ $\text{LWE}_{\mathbf{s},\alpha}$ samples and transforms them into R $\text{LWE}_{\mathbf{s}_2,\sigma}$ samples such that $\mathbf{s}_2$ is the second fraction of the secret key $\mathbf{s}$ and the noise parameter σ is bigger than the noise parameter α of the input. It takes time $R \cdot T(\text{BKZ}_\delta)$. Denote the matrix form of obtained LWE samples as $(\widehat{\mathbf{A}}, \widehat{\mathbf{b}}) \in (\mathbb{T}^{n_2\times R}, \mathbb{T}^R)$.

At the search step, the goal is to recover $\mathbf{s}_2$ using the obtained LWE samples. For each of the candidates for $\mathbf{s}_2$ the attack computes the error vector that corresponds to R LWE samples obtained at the previous step. It is equivalent to computing the following matrix expression:

$$\widehat{\mathbf{E}} = \widehat{\mathbf{B}} - \widehat{\mathbf{A}}^t\mathbf{S}_{(n_2)} \mod 1,$$

where $\mathbf{S}_{(n_2)}$ is the matrix composed of all binary vectors of length n_2 written in lexicographic order and $\widehat{\mathbf{B}} \in \mathbb{T}^{R\times 2^{n_2}}$ is the matrix formed of 2^{n_2} repetition of the vector $\widehat{\mathbf{b}}$. The complexity of computing that expression is dominated by the complexity of computing the product of $\widehat{\mathbf{A}}^t \in \mathbb{T}^{R\times n_2}$ and $\mathbf{S}_{(n_2)}$. By Corollary 8.1, it can be computed in $O(R \cdot 2^{n_2})$ operations. Once the attack obtain an error vector for each of the candidates, it guesses the distribution of each error vector using Algorithm 8.2 and returns the candidate whose error vector has concentrated around zero modular Gaussian distribution.

The time complexity of the attack is the sum of the complexities of the two steps:

$$T_{\text{attack}} = R \cdot \left(2^{n_2} + T(\text{BKZ}_\delta)\right). \tag{8.18}$$

Now the goal is to evaluate the number of samples R needed to recover $\mathbf{s}_2$ with probability p. By Lemma 8.2, using Algorithm 8.2, we can guess correctly the distribution of a sample of size R with probability at least $p_\sigma = 1 - \exp\left(-\frac{e^{-4\pi^2\sigma^2}}{8} \cdot R\right)$. In order to recover

$\mathbf{s}_2$, we need successfully guess the distribution for each of 2^{n_2} candidates. Assume that the distributions, produced by the candidates are independent. Then, the probability to correctly recover $\mathbf{s}_2$ is at least $p_\sigma^{2^{n_2}}$. Thus, to recover $\mathbf{s}_2$ we need to choose the size of the sample R that satisfies:

$$p_\sigma^{2^{n_2}} = \left(1 - \exp\left(-\frac{e^{-4\pi^2\sigma^2}}{8} \cdot R\right)\right)^{2^{n_2}} \geqslant p. \tag{8.19}$$

Let R be given by the following expression:

$$R = 8 \cdot e^{4\pi^2\sigma^2}(n_2 \ln(2) - \ln(\ln(1/p))). \tag{8.20}$$

Combining (8.19) and (8.20), we obtain:

$$p_\sigma^{2^{n_2}} = \left(1 - \frac{\ln(1/p)}{2^{n_2}}\right)^{2^{n_2}}. \tag{8.21}$$

Then, when $n_2 \to \infty$, $p_\sigma^{2^{n_2}} \to p$. Thus, the sample size R, given by (8.20) is sufficient to recover $\mathbf{s}_2$ with the probability p.

By combining (8.18) and (8.20) we obtain the time complexity of the attack. □

8.2.3 Using sieving in the hybrid attack

Assume that the BKZ algorithm uses the sieving algorithm (see for instance [BDGL16]) as an SVP oracle. At its penultimate step, the sieving algorithm produces many short vectors, so that by storing this pool of vectors, we may suppose that BKZ produces many short vectors in one run. Thus, if we need N short lattice vectors, we need to run the lattice reduction only $\lceil \frac{N}{m} \rceil$ times, where m is the number of short vectors, returned by the lattice reduction

In the following corollary from Theorem 8.2, we use this property of the sieving algorithm to revisit the time complexity of our attack under the sieving BKZ cost model.

Corollary 8.2. Let α, p, n, σ and $\mathbf{s} \in \{0;1\}^n$ be as in Theorem 8.2. Assume that the lattice reduction algorithm, used by Algorithm 8.3, uses the sieving algorithm from [BDGL16] as an oracle for solving SVP. Suppose that Assumption 7.1 holds. Then, the time complexity of solving the Search-LWE$_{\mathbf{s},\alpha}$ problem with probability of success p by the attack described in Algorithm 8.5 is

$$T_{\text{attack}} = \min_{\delta, n_2}\left(2^{n_2} \cdot R(n_1, \sigma, p) + T(\text{BKZ}_\delta)) \cdot \left\lceil \frac{R(n_2, \sigma, p)}{(4/3)^{\beta/2}} \right\rceil\right), \tag{8.22}$$

where β is the smallest blocksize such that the lattice reduction with the blocksize β achieves the Hermite factor δ; $R(n_2, \sigma, p)$ is as defined in Theorem 8.2.

Proof. By Theorem 8.2, the time complexity of Algorithm 8.5 can be seen as the sum of complexities of the two parts of the algorithm. The first part is producing R short lattice vectors and the second part is evaluating R scalar products for each of 2^{n_2} candidates for the secret key. As in the sieving model one run of the lattice reduction produces $(4/3)^{\beta/2}$ short vectors, the first part of Algorithm 8.5 attack takes time $T(\text{BKZ}_\delta)) \cdot \left\lceil \frac{R(n_2,\sigma,p)}{(4/3)^{\beta/2}} \right\rceil$, which implies that the complexity of Algorithm 8.5 in the sieving BKZ cost model is given by (8.22). □

Remark on using the sieving model with the attack from Section 8.1. As the dual attack from [CGGI20] consists in running the BKZ algorithm many times, it can

also benefit from using all the vectors, produced by the sieving subroutine. Then, the complexity of the dual attack from [CGGI20] in the sieving model is essentially divided by the number of vectors, produced by the sieving subroutine. See Corollary 8.3 for the complexity of the dual attack under the sieving BKZ cost model.

Corollary 8.3. Let α, S and $\mathbf{s} \in \{0,1\}$ be as in Theorem 8.1. Suppose that Assumption 7.1 holds. Assume that the lattice reduction algorithm, used by Algorithm 8.3, uses the sieving algorithm from [BDGL16] as an oracle for solving SVP. Then, the time complexity of solving Decision-LWE$_{\mathbf{s},\alpha}$ with probability of success p by the distinguishing attack described in Algorithm 8.3 is given by

$$T_{\text{TFHEattack}} = \min_{\delta} \left(\left\lceil \frac{N(\sigma, p)}{(4/3)^{\beta/2}} \right\rceil \cdot T(\text{BKZ}_{\delta}) \right), \tag{8.23}$$

where β is the smallest blocksize such that the lattice reduction with the blocksize β achieves the Hermite factor δ; σ and $N(\sigma, p)$ are as defined in Theorem 8.1.

Chapter 9

Bit-security estimation and experimental verification

We implement a Python script that, given parameters of an LWE problem and a BKZ cost model as an input, finds optimal parameters for the dual attack and for our attack (see Chapter 8). Using this script, we evaluate the computational cost of the dual attack and our attacks for a wide range of LWE parameters and in particular for the parameters used in the TFHE scheme. In this chapter, we report the results of our numerical estimation and show that the security level of the TFHE scheme should be updated with regard to the hybrid attack. We support our argument by an implementation working on a small example.

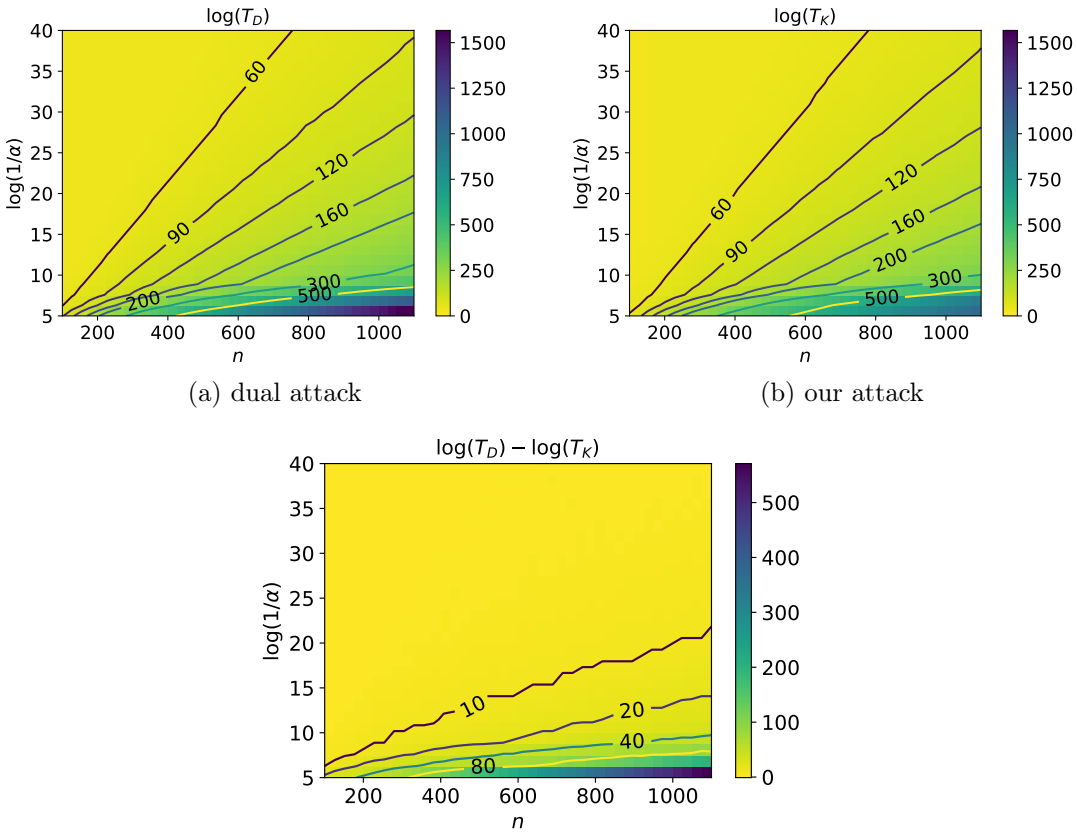
9.1 Bit-security of LWE parameters

We numerically estimate the cost of solving LWE problem by the dual attack and by our attack for all pairs of parameters (n, α) from the following set: $(n, -\log(\alpha)) \in \{100, 125, \ldots, 1050\} \times \{5, 6.25, \ldots, 38.5\}$. In all cases, we take $S^2 = 1/2$, which corresponds to choosing the secret key uniformly at random from $\{0, 1\}^n$ as done in the TFHE scheme. For each attack, we consider three BKZ cost models. For each case, we create a heatmap representing the cost of the attack as a function of parameters n and α. The results obtained using the sieving BKZ cost model are presented in Figure 9.1. The left heatmap in Figure 9.1 represents the logarithm of the time complexity of the dual attack, the right heatmap represents the logarithm of the time complexity of our attack. Figure 9.1 shows that for the same sets of parameters the cost of our attack is always less then or equal to the cost of the dual distinguishing attack and that the difference between the costs of the attacks grows with the hardness of the problem. We obtain similar pictures for the two other considered models. For completeness, we present the heatmaps for the other models in Appendix A.2.

9.2 Application to the TFHE scheme

The TFHE scheme uses two sets of parameters: for the switching key and for the bootstrapping key. The security of the scheme is, in fact, defined by the security of the switching key, which is the weaker link.

In [CGGI20], the authors of the TFHE scheme describe two sets of parameters for each of the keys. The first set of the parameters is given by [CGGI20, Table 3] and coincides with the parameters given in the previous papers on TFHE [CGGI16, CGGI17]. The bit-security for the parameters from [CGGI20, Table 3] is estimated according to the dual



(a) dual attack

(b) our attack

(c) difference

Figure 9.1 – Comparison of the costs of the attacks under the enumeration BKZ cost model. Here, n and α denote the dimension and the standard deviation of the noise of LWE samples, T_D denotes the time complexity of the dual distinguishing attack, T_K denotes the time complexity of our key recovery attack.

attack, described in Chapter 8.

Another set of the parameters is introduced by [CGGI20, Remark 9] and specified in [CGGI20, Table 4]. The security of the updated parameters is evaluated according to the LWE estimator from [ACD+18].

Also, very recently, new parameters for the public implementation of the TFHE scheme were introduced (see Table 7.3).

For completeness, we re-evaluate the security of all the three sets of the parameters under the dual attack as it is described in Chapter 8 and under our hybrid attack. In Table 9.1, we we present the results of our estimates for the old parameters (given by [CGGI20, Table 3]), in Table 9.2, we present the bit-security of the new parameters from [CGGI20, Table 4], and, in Table 9.3, we present the bit-security of the parameters of the public implementation that appeared at 21/02/2020.

In all cases, the cost of our attack is lower than the cost of the dual attack. In addition, the lattice reduction part is always easier for our attack than for the dual attack, because the required quality parameter of lattice reduction δ is always bigger for our attack than for the dual attack. However, the difference of the costs depends on the choice of the model: it is bigger for models that predict higher complexity of BKZ. For example, for the old switching key parameters, the difference under the sieving model is 8 bits while under enumeration model it is 58 bits.

In Figure 9.2 we present an estimation of the bit-security of the revisited LWE pa-

Table 9.1 – Security of the parameters of the TFHE scheme from [CGGI20, Table 3]. λ denotes security in bits, δ and n_1 are the optimal parameters for the attacks. "-" means that the distinguishing attack doesn't have the parameter n_1.

BKZ model	**switching key** $n = 500$, $\alpha = 2.43 \cdot 10^{-5}$				**bootstrapping key** $n = 1024$, $\alpha = 3.73 \cdot 10^{-9}$			
	attack	λ	δ	n_1	attack	λ	δ	n_1
delta-squared	dual	169	1.0052	-	dual	204	1.0046	-
	new attack	**119**	1.0059	406	**new attack**	**160**	1.0051	889
sieving	dual	102	1.0054	-	dual	117	1.0048	-
	new attack	**94**	1.0058	455	**new attack**	**112**	1.005	972
enumeration	dual	195	1.0052	-	dual	230	1.0046	-
	new attack	**137**	1.0062	388	**new attack**	**180**	1.0052	868

Table 9.2 – Security of the parameters of the TFHE scheme from [CGGI20, Table 4]. λ denotes security in bits, δ and n_1 are the optimal parameters for the attacks. "-" means that the distinguishing attack doesn't have the parameter n_1.

BKZ model	**switching key** $n = 612$, $\alpha = 2^{-15}$				**bootstrapping key** $n = 1024$, $\alpha = 2^{-26}$			
	attack	λ	δ	n_1	attack	λ	δ	n_1
delta-squared	dual	256	1.0043	-	dual	237	1.0043	-
	new attack	**169**	1.0051	474	**new attack**	**179**	1.0049	871
sieving	dual	127	1.0045	-	dual	126	1.0045	-
	new attack	**118**	1.0048	559	**new attack**	**120**	1.0047	970
enumeration	dual	279	1.0043	-	dual	261	1.0043	-
	new attack	**185**	1.0053	457	**new attack**	**179**	1.0049	871

Table 9.3 – Security of the parameters of the TFHE scheme: recent update of implementation's parameters [G+16]. λ denotes security in bits, δ and n_1 are the optimal parameters for the attacks. "-" means that the distinguishing attack doesn't have the parameter n_1.

BKZ model	**switching key** $n = 630$, $\alpha = 2^{-15}$				**bootstrapping key** $n = 1024$, $\alpha = 2^{-25}$			
	attack	λ	δ	n_1	attack	λ	δ	n_1
delta-squared	dual	270	1.0042	-	dual	256	1.0042	-
	new attack	**176**	1.005	485	**new attack**	**190**	1.0048	862
sieving	dual	131	1.0044	-	dual	131	1.0044	-
	new attack	**121**	1.0047	576	**new attack**	**125**	1.0046	967
enumeration	dual	292	1.0042	-	dual	280	1.0041	-
	new attack	**192**	1.0052	469	**new attack**	**209**	1.0049	842

rameters according to the combination of our attack and the collision attack, with time complexity $2^{n/2}$. Thus, Figure 9.2 represents the function $\min(T_{\text{ourAttack}}(n, \alpha), 2^{n/2})$, where $T_{\text{ourAttack}}(n, \alpha)$ is the cost of our attack for parameters n and α. Figure 9.2 is obtained under the enumeration BKZ cost model. See Appendix A.2 for other models.

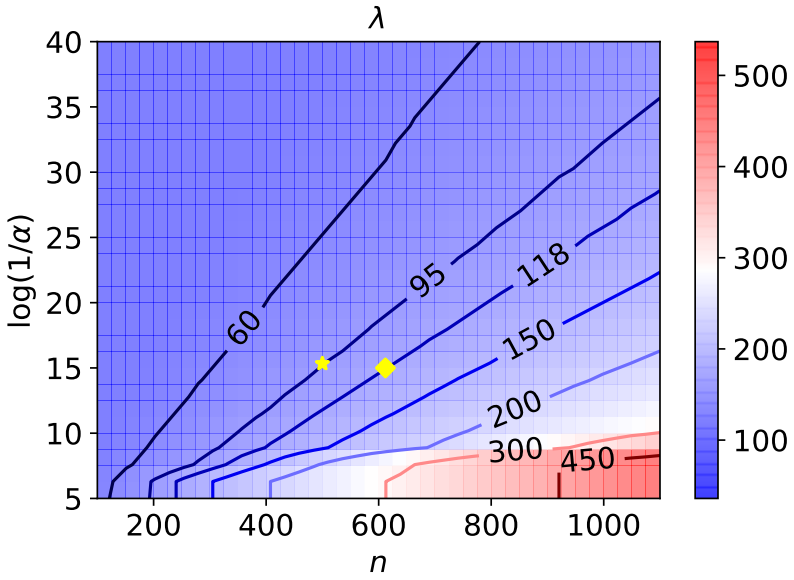


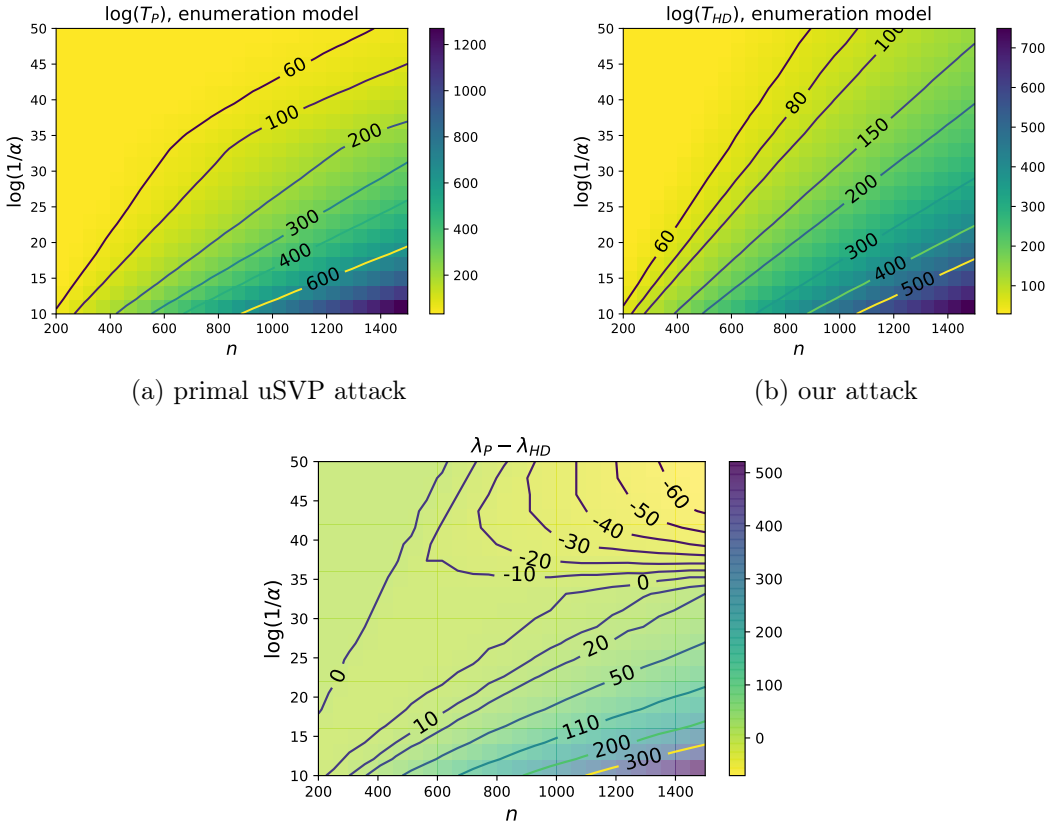
Figure 9.2 – Bit-security as a function of the LWE parameters n and α assuming the sieving BKZ cost model. Here, n denotes the dimension, α denotes the standard deviation of the noise. The picture represents the security level λ of LWE samples, $\lambda = \log(\min(T_{\text{ourAttack}}(n, \alpha), 2^{n/2}))$. The numbered lines on the picture represent security levels. The star symbol denotes the key switching parameters from the implementation of the TFHE scheme, the diamond symbol denotes the key switching parameters recommended in [CGGI20, Table 4].

9.3 Comparison with primal uSVP attack

The security of the recent parameters from TFHE's implementation is evaluated using the LWE estimator from [APS15, ACD+18]. As the results of this estimation suggest, under the sieving BKZ cost model, the best attack against the current parameters of the TFHE scheme among the attacks presented in the LWE estimator is the primal uSVP attack [BG14] (see also [APS15, Section 6.3] for the description of the attack). Therefore, it is interesting to compare our hybrid dual attack with the primal uSVP attack on a wider range of parameters.

In order to compare our attack with the primal uSVP attack, we estimate the time complexity of both attacks for each pair of the parameters (n, α) from the following set: $(n, -\log(\alpha)) \in \{200, 250, \ldots, 1450\} \times \{10, 12, \ldots, 48\}$. We evaluate the cost of the primal uSVP attack using the LWE estimator [APS15, ACD+18]. For this comparison, we consider two BKZ cost models: sieving and enumeration. The results of our estimation are presented in Figures 9.3 and 9.4.

Figures 9.3 and 9.4 show that under both BKZ cost models, it is not so that one attack is better than another for all the sets of the parameters. Under both BKZ cost models, the primal uSVP attack outperforms the hybrid dual attack when dimension is high (i.e., $n > 800$) and the noise parameter is small (i.e., $\alpha < 2^{-35}$). For the rest of the parameters that we consider, the hybrid dual attack outperforms the primal uSVP attack. The difference in the cost of the attacks depends on the chosen BKZ cost model; for the enumeration BKZ cost model the difference between attacks in more significant than for the sieving model.



(a) primal uSVP attack

(b) our attack

(c) difference

Figure 9.3 – Comparison of the costs of the hybrid dual attack and primal uSVP attack from [BG14] under the enumeration BKZ cost model. Here, n and α denote the dimension and the standard deviation of the noise of LWE samples, T_P denotes the time complexity of the primal uSVP attack, T_{HD} denotes the time complexity of our hybrid dual attack, $\lambda_P - \lambda_{HD} := \log(T_P) - \log(T_{HD})$.

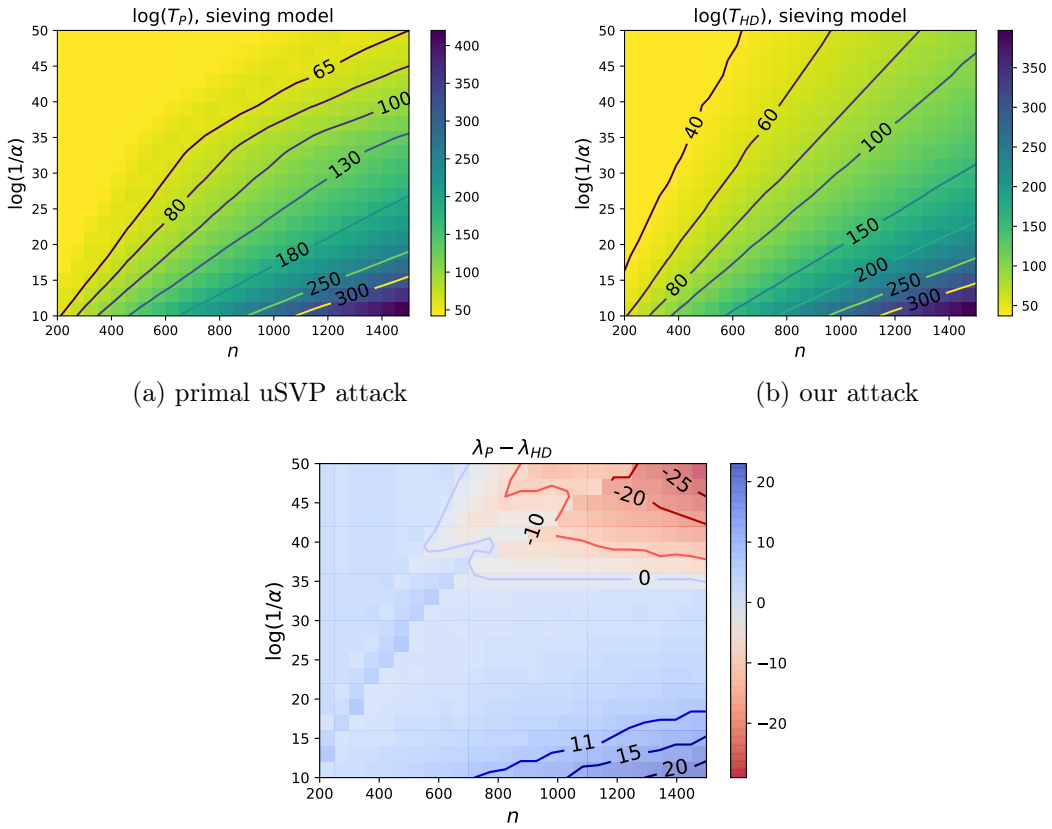
9.4 Experimental verification

In order to verify the correctness of our attack, we have implemented it on small examples. Our implementation recovers 5 bits of a secret key for LWE problems with the following two sets of parameters: $(n, \alpha) = (30, 2^{-8})$ and $(n, \alpha) = (50, 2^{-8})$.

For implementation purposes, we rescaled all the elements defined over torus $\mathbb{T}$ to integers modulo 2^{32}. For both examples, we use BKZ with blocksize 20, which yields the quality of the lattice reduction around $\delta \lesssim 1.013$. We computed the values of parameters of the attack required to guess correctly 5 bits of the key with probability 0.99 assuming that quality of the output of BKZ. The required parameters for both experiments are summarized in Table 9.4.

The first experiment was repeated 20 times, the second – 10 times. For both experiments, the last five bits of the key were successfully recovered at all attempts.

The correctness of both attacks rely on assumptions made in Lemma 8.1 for approximating the distribution of $\mathbf{v}^t(\mathbf{A}^t\mathbf{s} + \mathbf{e}) \mod 1$ by modular Gaussian distribution $\mathscr{G}_\sigma$. In order to verify these assumptions, while running both experiments we have collected samples to check the distribution: each time when the attack found correctly the last bits of the secret key $\mathbf{s}_2$, we collected the corresponding $\widetilde{e} = \widetilde{\mathbf{b}} - \widetilde{\mathbf{a}}^t\mathbf{s}_2 = \mathbf{v}^t(\mathbf{A}^t\mathbf{s}_1 + \mathbf{e})$. For the first experiment, the size of the collected sample is $20 \times R_1 = 640$, for the second experiment,



(a) primal uSVP attack

(b) our attack

(c) difference

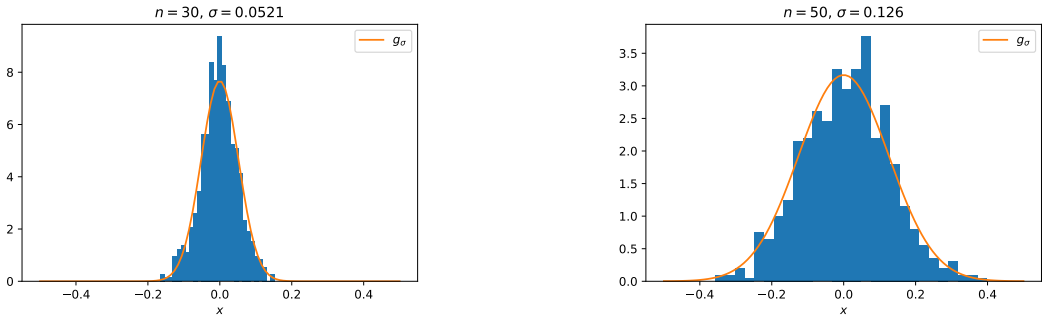
Figure 9.4 – Comparison of the costs of the hybrid dual attack and primal uSVP attack from [BG14] under the sieving BKZ cost model. Here, n and α denote the dimension and the standard deviation of the noise of LWE samples, T_P denotes the time complexity of the primal uSVP attack, T_{HD} denotes the time complexity of our hybrid dual attack, $\lambda_P - \lambda_{HD} := \log(T_P) - \log(T_{HD})$.

$(n, -\log(\alpha))$	m	σ	R
(30,8)	76	0.0521	32
(50,8)	90	0.126	74

Table 9.4 – Parameters required for guessing 5 bits of the key with $\delta = 1.013$. m is the number of samples needed for one lattice reduction (A.4), σ is the parameter of modular Gaussian distribution $\mathscr{G}_\sigma$ (Lemma 8.1), R is the number of samples needed to distinguish distributions $\mathscr{G}_\sigma$ and $\mathscr{U}$ (8.20).

it is $10 \times R_2 = 740$. The collected data is presented in Figure 9.5.

In Table 9.5, we compare theoretical predictions and estimations obtained from the experiments for the parameters of modular Gaussian distribution $\mathscr{G}_\sigma$. Experimental estimations of mean and variance in both cases match closely theoretical predictions.



(a) (b)

Figure 9.5 – Distribution of $\widetilde{e} = \mathbf{v}^t(\mathbf{A}^t\mathbf{s}_1 + \mathbf{e}) \mod 1$.
Figure 3a represents data from the experiment with parameters $(n, \alpha) = (30, 2^{-8})$, figure 3b – from the experiment with parameters $(n, \alpha) = (50, 2^{-8})$. Blue histograms denote observed data, orange lines – theoretical predictions of the distribution.

Table 9.5 – Estimated mean and variance. σ is the parameter of the modular Gaussian distribution $\mathscr{G}_\sigma$, $\mathrm{Var}(\mathscr{G}_\sigma)$ is variance of $\mathscr{G}$

(n, α)	sample size	σ	$\mathrm{Var}(\mathscr{G}_\sigma)$	estimated variance	average of sample
$(30, 2^{-8})$	640	0.0521	0.002714	0.002619	-0.00207
$(50, 2^{-8})$	740	0.126	0.1587	0.14515	0.0064

Conclusion

Cylindrical sieving

In the first part of the thesis, we introduced a new family of heuristic sieving algorithms, called cylindrical sieving, and showed that in some situations cylindrical sieving can be more efficient than the spherical sieving algorithms, namely, for solving SVP for lattices with a small prime volume and for solving the Closest Vector Problem with Preprocessing.

We should remark that the cylindrical sieving algorithm for solving CVPP can be used in the batch mode. That is, after the preprocessing stage, instead of reducing one target vector with the lists of lattice vectors prepared during the preprocessing, we can reduce many targets in the same time. Thus, we will iteratively obtain new targets lying in smaller and smaller hypercylinders. This approach allows to solve $2^{n/2}$ CVP instances in $\widetilde{O}(2^{n/2})$ time and space.

Concerning solving SVP in the general case with cylindrical sieving, the natural question to ask is whether it is possible to get a faster algorithm by using the nearest-neighbour search techniques for the spherical sieving part (i.e., sieving performed inside one chunk of the hypercylinder). Unfortunately, the answer is negative. Assume that we have N vectors on a sphere and an oracle that can find two "close" vectors among them in time less than $\sqrt{N}$ (which is true for the NNS technique used in [BDGL16]). If we optimize the complexity of the cylindrical sieving with such oracle used for the spherical sieving part, we get that the optimal growth factor γ tends to one, which means that the cylindrical sieving degenerates into a fully spherical sieving algorithm.

Another possible direction for the future work is to consider the cylindrical sieving algorithm for prime volume lattices when $\gamma = \sqrt{2}$. This special case is interesting because the requirements on the distribution of the lattice vectors during the sieving iterations are minimal and maybe there is a way to weaken the heuristic assumption. Essentially, the main requirement is that the first coordinate of lattice vectors behaves as if it is independent of the last coordinates. Probably some relaxation of this requirement might be made provable using the tools for analysis of modular sums.

Hybrid attacks against LWE

In the second part of the thesis, we considered the security of the TFHE scheme and showed that it should be re-evaluated under the hybrid dual attack against LWE. More generally, the hybrid dual attack described in the second part of the thesis can be used to estimate security of any binary-LWE-based cryptosystem. For example, it can be used to evaluate the security level of FHEW, another FHE scheme based on the LWE problem with binary secrets; under the hybrid dual attack we get 96 bits of security for FHEW's parameters.

One of the tools used by the attack is the algorithm for fast multiplication of an arbitrary matrix with the matrix of all binary vectors. The multiplication algorithm exploits the recursive structure of the matrix of all binary vectors. This algorithm can be easily

generalized for matrices that consist of all vectors whose coordinates are from some finite set. Such a generalization would allow to extend our hybrid dual attack against binary-LWE to an attack against more general versions of the LWE problem. We believe that adapting the attack to ternary- and small-secret-LWE is an interesting direction for the future work.

In order to estimate the complexity of the attack, we used three different models of lattice reduction. When estimating the complexity of the attack under the sieving model, we assume that the SVP-oracle used by the BKZ algorithm returns many short vectors instead of one. This assumption is based on the structure of the sieving algorithms that work with exponentially long lists of lattice vectors. However, the precise distribution of the lattice vectors at the last iteration of sieving is not well-studied. More work is needed to make the assumption more precise and provide a practical verification.

Appendix A

Omitted proofs and estimation results for the second part

A.1 Proof of Lemma 8.1.

Proof. Under Assumption 7.1, the coordinates of $\mathbf{w}_q$ are independent and distributed according to the Gaussian distribution with expectation 0 and standard deviation $\delta^{n+m}/\sqrt{n+m}$. Since $\mathbf{w}_q = (q \cdot \mathbf{x} \,||\, q^{-n/m} \cdot \mathbf{v})^t$, the coordinates of vectors $\mathbf{x}$ and $\mathbf{v}$ also have centered Gaussian distribution, but with different standard deviations. Let

$$\sigma_{\mathbf{x}} = \frac{1}{q} \cdot \frac{\delta^{m+n}}{\sqrt{m+n}} \quad \text{and} \quad \sigma_{\mathbf{v}} = q^{n/m} \cdot \frac{\delta^{m+n}}{\sqrt{m+n}}$$

be the standard deviation of coordinates of $\mathbf{x}$ and of $\mathbf{v}$ correspondingly. Consider the distribution of

$$\mathbf{v}^t\mathbf{b} = \mathbf{x}^t\mathbf{s} + \mathbf{v}^t\mathbf{e} = \sum_{i=1}^{n} x_i \cdot s_i + \sum_{i=1}^{m} v_i \cdot e_i.$$

$\mathbf{v}^t\mathbf{b}$ is a sum of $m+n$ independent random variables and, therefore, its distribution can be approximated by a Gaussian distribution according to the Central Limit Theorem. In order to learn the parameters of the Gaussian, we need to obtain the expectations and variances of $x_1 \cdot s_1$ and $v_1 \cdot e_1$.

First, consider the distribution of $x_1 \cdot s_1$. As s_1 has a Bernoulli distribution with parameter S^2, $x_1 s_1$ is a random variable from the distribution that can be obtained by sampling 0 with probability S^2 and sampling from a Gaussian distribution with mean 0 and variance $\sigma_{\mathbf{x}}^2$ with probability $1 - S^2$. Therefore, $\mathbb{E}(x_1 \cdot s_1) = 0$ and $\mathrm{Var}(x_i \cdot s_i) = S^2\sigma_{\mathbf{x}}^2$.

Then, consider $v_1 e_1$. As $\mathbf{v}$ and $\mathbf{e}$ are independent and $\mathbb{E}(v_1) = \mathbb{E}(e_1) = 0$, $\mathbb{E}(v_1 e_1) = \mathbb{E}(v_1)\mathbb{E}(e_1) = 0$ and $\mathrm{Var}(v_1 e_1) = \mathrm{Var}(v_1) \cdot \mathrm{Var}(e_1) = \alpha^2\sigma_{\mathbf{v}}^2$.

Thus, the distribution of $\mathbf{v}^t\mathbf{b}$ is close to the Gaussian distribution with expectation 0 and variance

$$\sigma^2 = n\,\mathrm{Var}(x_1 s_1) + m\,\mathrm{Var}(v_1 e_1) = nS^2\sigma_{\mathbf{x}}^2 + m\alpha^2\sigma_{\mathbf{v}}^2 = \frac{\delta^{2(m+n)}}{m+n}\Big(\frac{nS^2}{q^2} + m\alpha^2 q^{2n/m}\Big). \quad \text{(A.1)}$$

Our goal is to obtain a distribution that is as concentrated around zero as possible. Hence we choose parameters m and q in order to minimize variance of $\mathbf{v}^t\mathbf{b}$.

First, we find the optimal value of q by differentiation of Equation (A.1) :

$$\frac{\partial\sigma^2}{\partial q} = \frac{\delta^{2(m+n)}}{m+n} \cdot \Big(-\frac{2nS^2}{q^3} + \frac{2n}{m} \cdot m\alpha^2 q^{\frac{2n}{m}-1}\Big) = 0 \quad \to \quad q_{\mathrm{opt}} = \Big(\frac{S}{\alpha}\Big)^{\frac{m}{m+n}}.$$

After replacing q by q_{opt} in Equation (A.1) we obtain:

$$\sigma^2 = \left(S\delta^{m+n} \left(\frac{\alpha}{S}\right)^{\frac{m}{m+n}} \right)^2. \tag{A.2}$$

Also, for $\sigma_{\mathbf{x}}$ and $\sigma_{\mathbf{v}}$ we obtain the following relation

$$\frac{\sigma_{\mathbf{x}}}{\sigma_{\mathbf{v}}} = \frac{q^{-n/m}}{q} = \frac{\alpha}{S}. \tag{A.3}$$

Then, we find the optimal value of m by differentiating $\ln(\sigma)$:

$$\frac{\partial \ln(\sigma)}{\partial m} = \ln(\delta) + n \ln\left(\frac{\alpha}{S}\right) \cdot \frac{1}{(m+n)^2} = 0 \quad \rightarrow \quad m_{\text{opt}} = \sqrt{n \cdot \frac{\ln(S/\alpha)}{\ln(\delta)}} - n \tag{A.4}$$

Now, replacing m by m_{opt} in Equation (A.2), we find:

$$\sigma(\delta, n, S, \alpha) = \sigma(\widehat{m}, \delta, n, S, \alpha) = \alpha \cdot \exp\left(2\sqrt{n \ln(S/\alpha) \ln(\delta)}\right).$$

The distance between the distribution of $\mathbf{v}^t\mathbf{b}$ and the Gaussian distribution with mean 0 and variance σ^2 can be estimated by the Berry-Esseen inequality (see Theorem 7.1). To use this inequality, we need to compute the third absolute moments of $x_1 s_1$ and $v_1 e_1$.

We start with $x_1 s_1$. As x_1 and s_1 are independent,

$$\mathbb{E}\{|x_1 s_1|^3\} = \mathbb{E}\{|x_1|^3\}\mathbb{E}\{|s_1|^3\}.$$

By Lemma 7.4, $\mathbb{E}\{|x_1|^3\} = 2\sqrt{2/\pi}\sigma_x^3$. As s_1 has the Bernoulli distribution with parameter S^2, $\mathbb{E}\{|s_1|^3\} = \mathbb{E}\{s_1\} = S^2$. Putting two parts together, we get

$$\rho_{x_1 s_1} = \mathbb{E}\{|x_1 s_1|^3\} = 2\sqrt{2/\pi} S^2 \sigma_{\mathbf{x}}^3. \tag{A.5}$$

In the same way, we obtain

$$\rho_{v_1 e_1} \mathbb{E}\{|v_1 e_1|^3\} = \frac{8}{\pi} \alpha^3 \sigma_{\mathbf{v}}^3. \tag{A.6}$$

Denote the cumulative distribution function of $\mathbf{v}^t\mathbf{b}$ by $F_{\mathbf{v}^t\mathbf{b}}$, and denote the cumulative distribution function of the Gaussian distribution with mean 0 and variance σ^2 by Φ_σ. By the Berry-Esseen inequality, there exists a constant C_0 such that

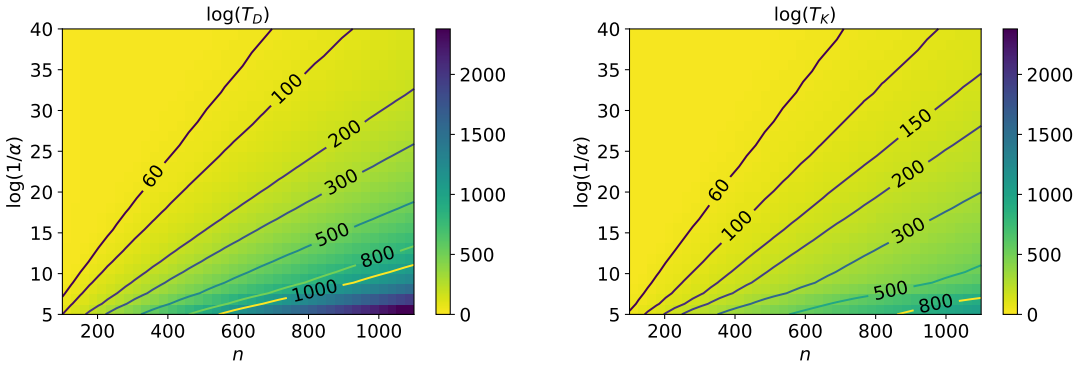
$$\sup_{x \in \mathbb{R}} |F_{\mathbf{v}^t\mathbf{b}}(x) - \Phi_\sigma(x)| \leqslant C_0 \cdot \frac{n\rho_{x_1 s_1} + m\rho_{v_1 e_1}}{(nS^2\sigma_{\mathbf{x}}^2 + m\alpha^2\sigma_{\mathbf{v}}^2)^{3/2}}. \tag{A.7}$$

Then, using Equations (A.3) and (A.5) to (A.7), for the distance between the distributions we get:

$$\sup_{x \in \mathbb{R}} |F_{\mathbf{v}^t\mathbf{b}}(x) - \Phi_\sigma(x)| \leqslant C_0 \sqrt{\frac{8}{S^2\pi}} \cdot \frac{n + mS\sqrt{8/\pi}}{(m+n)^{3/2}} \leqslant C_0 \sqrt{\frac{8}{S^2\pi}} \cdot \frac{1}{\sqrt{m+n}}. \tag{A.8}$$

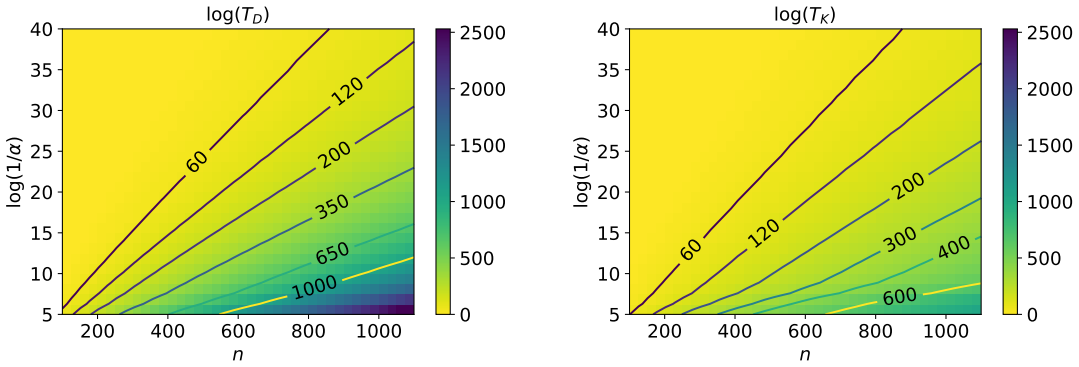
□

A.2 Heatmaps for the enumeration and delta-squared BKZ cost models



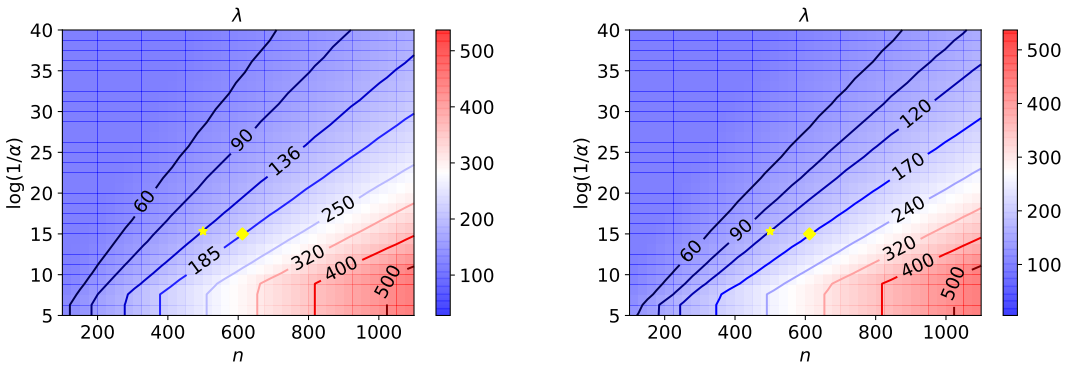
(a) dual attack (b) our attack

Figure A.1 – Comparison of the costs of the attacks under the enumeration BKZ cost model.



(a) dual attack (b) our attack

Figure A.2 – Comparison of the costs of the attacks under the delta-squared BKZ cost model.



(a) enumeration (b) delta-squared

Figure A.3 – Bit-security as a function of LWE parameters n and α under the sieving and delta-squared BKZ cost models.

Bibliography

[ACD+18] Martin R Albrecht, Benjamin R Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the {LWE, NTRU} schemes! In *International Conference on Security and Cryptography for Networks*, pages 351–367. Springer, 2018.

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

[ADRSD15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the Shortest Vector Problem in 2^n time using discrete Gaussian sampling. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 733–742. ACM, 2015.

[ADSD15] Divesh Aggarwal, Daniel Dadush, and Noah Stephens-Davidowitz. Solving the Closest Vector Problem in 2^n time – The Discrete Gaussian Strikes Again! In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 563–582. IEEE, 2015.

[AFG13] Martin R Albrecht, Robert Fitzpatrick, and Florian Göpfert. On the efficacy of solving LWE by reduction to unique-SVP. In *International Conference on Information Security and Cryptology*, pages 293–310. Springer, 2013.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108. ACM, 1996.

[Ajt98] Miklós Ajtai. The Shortest Vector Problem in L2 is NP-hard for randomized reductions. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 10–19. ACM, 1998.

[Ajt02] Miklos Ajtai. A conjectured 0-1 law about the polynomial time computable properties of random lattices. In *Electronic Colloquium on Computational Complexity (ECCC)*, number 061, 2002.

[AKS01] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 601–610. ACM, 2001.

[Alb17] Martin R Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 103–129. Springer, 2017.

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[AR05] Dorit Aharonov and Oded Regev. Lattice problems in NP $\cap$ coNP. *Journal of the ACM (JACM)*, 52(5):749–765, 2005.

[Bab86] László Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[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*, pages 10–24. Society for Industrial and Applied Mathematics, 2016.

[BG14] Shi Bai and Steven D Galbraith. Lattice decoding attacks on binary lwe. In *Australasian Conference on Information Security and Privacy*, pages 322–337. Springer, 2014.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 533–556. Springer, 2014.

[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.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography Conference*, pages 325–341. Springer, 2005.

[BGPW16] Johannes Buchmann, Florian Göpfert, Rachel Player, and Thomas Wunderer. On the hardness of LWE with binary error: Revisiting the hybrid lattice-reduction and meet-in-the-middle attack. In *International Conference on Cryptology in Africa*, pages 24–43. Springer, 2016.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, 2003.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of Learning with Errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 575–584. ACM, 2013.

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

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[Cas12] John William Scott Cassels. *An introduction to the geometry of numbers*. Springer Science & Business Media, 2012.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–33. Springer, 2016.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 377–408. Springer, 2017.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[Che13] Yuanmi Chen. *Réduction de réseau et sécurité concrete du chiffrement completement homomorphe*. PhD thesis, Paris 7, 2013.

[CHHS19] Jung Hee Cheon, Minki Hhan, Seungwan Hong, and Yongha Son. A Hybrid of Dual and Meet-in-the-Middle Attack on Sparse and Ternary Secret LWE. *IEEE Access*, 7:89497–89506, 2019.

[CL15] Jung Hee Cheon and Changmin Lee. Approximate algorithms on lattices with small determinant. Technical report, Cryptology ePrint Archive, Report 2015/461, 2015.

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *International Workshop on Public Key Cryptography*, pages 311–328. Springer, 2014.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Annual Cryptology Conference*, pages 487–504. Springer, 2011.

[CN11] Yuanmi Chen and Phong Q Nguyen. Bkz 2.0: Better lattice security estimates. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer, 2011.

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 446–464. Springer, 2012.

[Coh13] Henri Cohen. *A course in computational algebraic number theory*, volume 138, chapter $\mathbb{Z}$-Modules and the Hermite and Smith Normal Forms. Springer Science & Business Media, 2013.

[CS15] Jung Hee Cheon and Damien Stehlé. Fully homomophic encryption over the integers revisited. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 513–536. Springer, 2015.

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

[DKS98] Irit Dinur, Guy Kindler, and Shmuel Safra. Approximating-CVP to within almost-polynomial factors is NP-hard. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 99–109. IEEE, 1998.

[DLdW19] Emmanouil Doulgerakis, Thijs Laarhoven, and Benne de Weger. Finding closest lattice vectors using approximate Voronoi cells. *PQCRYPTO. Springer (2019, to appear)*, 2019.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 617–640. Springer, 2015.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[G+16] Nicolas Gama et al. Github repository. TFHE: Fast fully homomorphic encryption library over the torus. https://github.com/tfhe/tfhe, 2016.

[GB09] Craig Gentry and Dan Boneh. *A fully homomorphic encryption scheme*, volume 20. Stanford University Stanford, 2009.

[GH11a] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 107–109. IEEE, 2011.

[GH11b] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 129–148. Springer, 2011.

[GINX16] Nicolas Gama, Malika Izabachène, Phong Q Nguyen, and Xiang Xie. Structural lattice reduction: generalized worst-case to average-case reductions and homomorphic cryptosystems. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 528–558, 2016.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[GM03] Daniel Goldstein and Andrew Mayer. On the equidistribution of Hecke points. In *Forum Mathematicum*, volume 15, pages 165–190. Berlin; New York: De Gruyter, c1989-, 2003.

[GMP] GMP, The GNU Multiple Precision Arithmetic Library. https://gmplib.org.

[GMSS99] Oded Goldreich, Daniele Micciancio, Shmuel Safra, and J-P Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55–61, 1999.

[GN08] Nicolas Gama and Phong Q Nguyen. Predicting lattice reduction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 31–51. Springer, 2008.

[GNR10] Nicolas Gama, Phong Q Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 257–278. Springer, 2010.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008.

[GS87] Yuri Gurevich and Saharon Shelah. Expected computation time for hamiltonian path problem. *SIAM Journal on Computing*, 16(3):486–502, 1987.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Annual Cryptology Conference*, pages 75–92. Springer, 2013.

[HG07] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In *Annual International Cryptology Conference*, pages 150–169. Springer, 2007.

[HKL18] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. Speed-ups and time–memory trade-offs for tuple lattice sieving. In *IACR International Workshop on Public Key Cryptography*, pages 407–436. Springer, 2018.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *Annual Cryptology Conference*, pages 447–464. Springer, 2011.

[HS07] Guillaume Hanrot and Damien Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm. In *Annual International Cryptology Conference*, pages 170–186. Springer, 2007.

[Kah96] David Kahn. *The Codebreakers: The comprehensive history of secret communication from ancient times to the internet*. Simon and Schuster, 1996.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 193–206. ACM, 1983.

[Kan87] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, 1987.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. journal des sciences militaries. *IX (38)*, 5, 1883.

[Kho05] Subhash Khot. Hardness of approximating the Shortest Vector Problem in lattices. *Journal of the ACM (JACM)*, 52(5):789–808, 2005.

[KL74] GA Kabatyanskiı and VI Levenshteın. Bounds for packings on a sphere and in space. *Problems of Information Transmission*, 95:148–158, 1974.

[Kle00] Philip Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 937–941. Society for Industrial and Applied Mathematics, 2000.

[KPV12] Subhash Khot, Preyas Popat, and Nisheeth K Vishnoi. $2^{\log^{1-\varepsilon} n}$ hardness for the Closest Vector Problem with Preprocessing. In *44th Annual ACM Symposium on Theory of Computing, STOC'12*, pages 277–288, 2012.

[Laa15a] Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, Eindhoven University of Technology, 2015. http://www.thijs.com, 2015.

[Laa15b] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *Annual Cryptology Conference*, pages 3–22. Springer, 2015.

[Laa16] Thijs Laarhoven. Sieving for closest lattice vectors (with preprocessing). In *International Conference on Selected Areas in Cryptography*, pages 523–542. Springer, 2016.

[LdW15] Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In *International Conference on Cryptology and Information Security in Latin America*, pages 101–118. Springer, 2015.

[Li11] Shengqiao Li. Concise formulas for the area and volume of a hyperspherical cap. *Asian Journal of Mathematics and Statistics*, 4(1):66–70, 2011.

[LK14] Yongjae Lee and Woo Chang Kim. Concise formulas for the surface area of the intersection of two hyperspherical caps. *KAIST Technical Report*, 2014.

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In *International Colloquium on Automata, Languages, and Programming*, pages 144–155. Springer, 2006.

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *Annual International Cryptology Conference*, pages 577–594. Springer, 2009.

[LN13] Mingjie Liu and Phong Q Nguyen. Solving BDD by enumeration: An update. In *Cryptographers' Track at the RSA Conference*, pages 293–309. Springer, 2013.

[Lov86] László Lovász. *An algorithmic theory of numbers, graphs, and convexity*, volume 50. SIAM, 1986.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In *Cryptographers' Track at the RSA Conference*, pages 319–339. Springer, 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and Learning with Errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[Lyu08] Vadim Lyubashevsky. Lattice-based identification schemes secure under active attacks. In *International Workshop on Public Key Cryptography*, pages 162–179. Springer, 2008.

[MG12] Daniele Micciancio and Shafi Goldwasser. *Complexity of lattice problems: a cryptographic perspective*, volume 671. Springer Science & Business Media, 2012.

[Mic01] Daniele Micciancio. The hardness of the Closest Vector Problem with Preprocessing. *IEEE Transactions on Information Theory*, 47(3):1212–1215, 2001.

[Mic02] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 356–365. IEEE, 2002.

[Mic18] Daniele Micciancio. On the hardness of Learning with Errors with binary secrets. *Theory of Computing*, 14(1):1–17, 2018.

[MKVOV96] Alfred J Menezes, Jonathan Katz, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[MO90] James E Mazo and Andrew M Odlyzko. Lattice points in high-dimensional spheres. *Monatshefte für Mathematik*, 110(1):47–61, 1990.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In *Annual Cryptology Conference*, pages 21–39. Springer, 2013.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007.

[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*, pages 1468–1480. Society for Industrial and Applied Mathematics, 2010.

[MW16] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 820–849. Springer, 2016.

[NV08] Phong Q Nguyen and Thomas Vidick. Sieve algorithms for the Shortest Vector Problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008.

[Odl90] Andrew M Odlyzko. The rise and fall of knapsack cryptosystems. *Cryptology and computational number theory*, 42:75–88, 1990.

[P+16] Chris Peikert et al. A decade of lattice cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424, 2016.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case Shortest Vector Problem. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 333–342. ACM, 2009.

[PR06] Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *Theory of Cryptography Conference*, pages 145–166. Springer, 2006.

[RAD+78] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Ran53] Robert Alexander Rankin. On positive definite quadratic forms. *Journal of the London Mathematical Society*, 1(3):309–314, 1953.

[Reg05] O Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography, 2005. In *STOC*, pages 84–93. ACM, 2005.

[RS10] Markus Rückert and Michael Schneider. Estimating the security of lattice-based cryptosystems. *IACR Cryptology ePrint Archive*, 2010:137, 2010.

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

[SC19] Yongha Son and Jung Hee Cheon. Revisiting the hybrid attack on sparse and ternary secret LWE. *IACR Cryptology ePrint Archive*, 2019:1019, 2019.

[SE94] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994.

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

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[Sie45] Carl Ludwig Siegel. A mean value theorem in geometry of numbers. *Annals of Mathematics*, pages 340–347, 1945.

[SS10] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 377–394. Springer, 2010.

[SV10] Nigel P Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography*, pages 420–443. Springer, 2010.

[VDGHV10] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer, 2010.

[Wen62] James G Wendel. A problem in geometric probability. *Math. Scand*, 11:109–111, 1962.

[WLTB11] Xiaoyun Wang, Mingjie Liu, Chengliang Tian, and Jingguo Bi. Improved nguyen-vidick heuristic sieve algorithm for shortest vector problem. In *AsiaCCS*, pages 1–9, 2011.

[Wun16] Thomas Wunderer. Revisiting the hybrid attack: Improved analysis and refined security estimates. *IACR Cryptology ePrint Archive*, 2016:733, 2016.

[ZPH13] Feng Zhang, Yanbin Pan, and Gengran Hu. A three-level sieve algorithm for the Shortest Vector Problem. In *International Conference on Selected Areas in Cryptography*, pages 29–47. Springer, 2013.