Post-Quantum Cryptography: Cryptanalysis and Implementation

Fernando Virdia

Thesis submitted to Royal Holloway, University of London
for the degree of Doctor of Philosophy

Information Security Group
Royal Holloway, University of London

2021

Declaration

These doctoral studies were conducted under the supervision of Prof. Martin R. Albrecht.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Information Security Group as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Fernando Virdia
March, 2021

Acknowledgements

This thesis is the result of work and study carried out during the last four and a half years. It has been a time full of opportunities and challenges, and many people have contributed to making it fruitful and memorable. Inevitably, I will not be able to thank everybody who contributed to making it so, but I would still like to give it a go.

First and foremost, my thanks go to my supervisor Martin Albrecht. The support, guidance, patience and trust he has shown to me from the first day has given me the confidence to ask questions, investigate topics and opportunities, propose ideas, fail and try again. Thank you, Martin.

I would also like to thank the Centre for Doctoral Training in Cyber Security at Royal Holloway, which provided me with countless opportunities through organised activities and travel support. It was a privilege to be able to attend plenty of events that sparked ideas and collaborations. Thank you also to the Information Security Group's staff and students, who exposed me to new ideas through discussions, seminars and reading groups.

Six months of my training were spent at Microsoft Research as a summer intern. The time spent there has been some of the most productive and fun I had. I would like to express my deep gratitude to Brian LaMacchia and the Security and Cryptography team, and to Martin Roetteler and Microsoft Quantum for making this possible. I would like to thank Craig Costello and Michael Naehrig for their excellent mentorship.

I am grateful to my co-authors for their effort, insight and help during our collaborations: Martin Albrecht, Ilaria Chillotti, Craig Costello, Benjamin Curtis, Jan-Pieter D'Anvers, Alex Davidson, Amit Deo, Ashley Fraser, Florian Göpfert, Christian Hanser, Andrea Hoeller, Samuel Jaques, Patrick Longa, Michael Naehrig, Rachel Player, Thomas Pöppelmann, Eamonn Postlethwaite, Joost Renes, Martin Roetteler, Mélissa Rossi, Andreas Wallner, Thomas Wunderer.

I would like to thank Carlos Cid and Alexander May for accepting to read my work and be my examiners.

Personally, writing this thesis was not made easier by the need to work from home in the last few months. I would like to thank Rob and Nick for helping me give some structure to time via remote work sessions during the week and gaming sessions during the weekend.

Thank you to Martin Albrecht, Michael Naehrig, Samuel Jaques, Thomas Wunderer, Eliana, Jacopo and Eleonora for reading portions of this thesis and discussing them with me.

During my studies, I was lucky enough to make many new friends, who cheered me during the good times and supported me during the less good ones. Thanks go to Eamonn, Feargus, Rob, Amit, Ben, Ashley, Nick, Reynold, Rachel, Alex, Lydia, Gio, Nina, Clara, Timur, Ilaria, Adi, Giulia, Joost, Sam, Rohan, Rishabh and everyone else studying at Royal Holloway and interning at Microsoft Research with me, and to my friends from previous adventures in Italy, Argentina and the UK who supported me in the last four years.

Finally, I thank for their support my family, *quienes estuvieron desde el principio y quienes llegaron después.*

Fernando Virdia
March, 2021

Publications

The content of this thesis is based on the following four publications.

1. Albrecht M. R., Göpfert F., Virdia F., Wunderer T. Revisiting the Expected Cost of Solving uSVP and Applications to LWE. In: Takagi T., Peyrin T. (eds) Advances in Cryptology, Asiacrypt 2017. Lecture Notes in Computer Science, vol 10624. Springer, Cham.

2. Albrecht M. R., Hanser C., Hoeller A., Pöppelmann T., Virdia F., Wallner A. Implementing RLWE-based Schemes Using an RSA Co-Processor. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019(1).

3. Jaques S., Naehrig M., Roetteler M., Virdia F. Implementing Grover Oracles for Quantum Key Search on AES and LowMC. In: Canteaut A., Ishai Y. (eds) Advances in Cryptology, Eurocrypt 2020. Lecture Notes in Computer Science, vol 12106. Springer, Cham.

4. Postlethwaite E. W., Virdia F. On the Success Probability of Solving Unique SVP via BKZ. In: Garay J.A. (eds) Public-Key Cryptography, PKC 2021. Lecture Notes in Computer Science, vol 12710. Springer, Cham.

The following three publications were also written during my studies at Royal Holloway.

5. Albrecht M. R., Curtis B. R., Deo A., Davidson A., Player R., Postlethwaite E. W., Virdia F., Thomas Wunderer T. Estimate All the LWE, NTRU Schemes!. In: Catalano D., De Prisco R. (eds) Security and Cryptography for Networks. SCN 2018. Lecture Notes in Computer Science, vol 11035. Springer, Cham.

6. Costello C., Longa P., Naehrig M., Renes J., Virdia F. Improved Classical Cryptanalysis of SIKE in Practice. In: Kiayias A., Kohlweiss M., Wallden P., Zikas V. (eds) Public-Key Cryptography, PKC 2020. Lecture Notes in Computer Science, vol 12111. Springer, Cham.

7. D'Anvers JP., Rossi M., Virdia F. (One) Failure Is Not an Option: Bootstrapping the Search for Failures in Lattice-Based Encryption Schemes. In: Canteaut A., Ishai Y. (eds) Advances in Cryptology, Eurocrypt 2020. Lecture Notes in Computer Science, vol 12107. Springer, Cham.

Research for publications 3. and 6. was partly carried out during internship placements at Microsoft Research in Redmond, WA.

All my research was supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/P009301/1).

Abstract

Post-quantum cryptography is the field of study and development of cryptographic primitives providing security in the presence of adversaries capable of running large-scale error-tolerant quantum computations. Works in this area span from theoretical analysis of security definitions and protocols, to the research of classical and quantum cryptanalytic algorithms, to the development of cryptographic schemes that can be deployed for real-world usage.

In this thesis, we investigate three topics in practical post-quantum cryptography. First, we research quantum circuit depth-width trade-offs in the case of Grover's algorithm and how these impact the cost of running key-search attacks against block ciphers. Such attacks have been proposed by the US National Institute of Standards and Technology as benchmarks to define quantum security, and hence their cost should be well understood. Furthermore, Grover speed-ups are a component of many quantum attacks, making the study of these trade-offs of independent interest.

Second, we study the "primal attack" on lattice-based cryptosystems. This consists of using lattice reduction to recover an unusually short vector in a q-ary lattice, which results in a break of LWE- and NTRU-based schemes. We compare two alternative heuristics used to estimate the expected cost of this attack due to Gama *et al.* (Eurocrypt 2008) and Alkim *et al.* (USENIX 2016) and provide experimental evidence of the validity of the latter. Then, using the techniques introduced in Dachman-Soled *et al.* (Crypto 2020), we continue this line of work to provide estimates on the full probability distribution of the cost of the attack, providing further experimental validation.

In the last chapter, we move our focus from cryptanalysis to implementation. We implement a lattice-based actively secure key encapsulation mechanism on a currently commercially available smart card from the SLE 78 family by Infineon. We do this by repurposing classic arithmetic techniques that enable us to take advantage of the card's RSA coprocessor to compute polynomial multiplications in $\mathbb{Z}_q[x]/(x^{256}+1)$. The resulting scheme, a variant of Kyber768, runs key generation in 79.6 ms, encapsulation in 102.4 ms, and decapsulation in 132.7 ms. Our techniques can be adapted to other RSA/ECC coprocessors and demonstrate the feasibility of repurposing already deployed cryptographic coprocessors to run post-quantum encryption with reasonable performances.

Contents

List of Figures

List of Tables

Chapter 0

Introduction

Cryptography is a field of study that researches techniques to provide various forms of *security* to communications[1], such as confidentiality, integrity and authenticity. As an academic subject, contemporary cryptography lives in the intersection of computer science, electronic engineering, and (applied) pure mathematics. It enjoys deep connections to, among others, information and complexity theory, algebra, algorithmics, and circuit design.

Outside of the academic world, cryptography has been practised throughout history, and its use has become pervasive since the beginning of the digital age and the mass deployment of personal computing devices. Widespread use of encrypted communications over the Internet, the availability of chip card payment terminals in retail stores and the adoption of cryptographic technology by mobile messaging services means that the general public relies on cryptography during everyday life. Thus, it is fundamental for cryptographic techniques to be *practical* and *secure*.

While these two words have multiple possible meanings in the context of information security, in this thesis we consider the following two: a cryptographic primitive or protocol is practical if its use by one or more parties does not hinder their ability to communicate, for example by significantly slowing them down or by increasing their communication costs; it is secure if it does provide a well understood and sufficient amount of security that would otherwise have been missing.

Hard problems. At the core of modern cryptography are *hardness assumptions*. These take the form of mathematical problems that are understood to

[1] And more recently also to delegated computations.

be hard to solve, in the sense that finding the solution to a random instance of the problem requires a significant amount of computation. Some examples are the problem of finding the prime factorisation of an integer $N = p \times q$, where p and q are unknown prime factors of similar size, such that $\log p \approx \log q$; the problem of finding what elements in some large family of random permutations $\{f\colon \{0,1\}^n \to \{0,1\}^n\}$ map given known inputs to their known output; the problem of solving a set of approximate linear equations $\mathbf{Ax} \approx \mathbf{b}$ over a finite integer ring $\mathbb{Z}_q$.

The mathematical nature of the hardness assumption may allow for different cryptographic functionality to be built from it and will affect the resulting protocols' practicality, while the assumption's computational hardness will affect the protocols' security. From this point of view, practicality and security are intrinsically connected: if the hardness of the problem scales badly with the size of its description, protocol designers may need to either compromise on the security or the practicality of any constructions derived from it. Ideally, cryptographers prefer hard problems that allow for a concise description.

Today, most constructions providing advanced functionality such as public-key encryption or digital signatures[2] rely on one of two (families of) hard mathematical problems: the Diffie-Hellman (DH) problem [DH76] and the RSA problem [RSA78]. The first is formulated using finite fields or elliptic curves and is closely related to the discrete logarithm problem [den90, Mau94, MW99, JN03], the latter is formulated using integer rings and is closely related to the factoring problem [BV98, Bro16, AM09].

Their security is established by the extensive cryptanalytic literature exploring the concrete and asymptotic hardness of these problems [Pol78, LLMP93, Gor93, BBB+09, BGG+20]. Their practicality is the result of decades of electronic circuit miniaturisation and arithmetic circuit optimisation that allow running protocols built on them on both general-purpose computers and embedded hardware with heavily constrained computational resources.

[2] Both being technologies that allow parties to communicate securely even if they have not previously met to agree on a shared secret key.

Post-quantum cryptography. In a breakthrough result, in 1994 Peter Shor published an algorithm [Sho94] that allows solving both the factoring and discrete logarithm problems in a short amount of time. To run Shor's algorithm, however, a radically different kind of computer is required, usually referred to as a *quantum computer*. While first conceived in the 1980s [Ben80, MY80, Fey82, Fey86], practical advancements in quantum computing hardware have been initially slow. In recent years, however, the pace of advancement has increased, and industry investments resulted in the first small-scale noisy[3] quantum computers being built [MQT18, MN18, AAB+19, Gib19, WFG21]. Due to Shor's algorithm, the availability of large-scale quantum computers would result in a complete loss of security for protocols built on RSA or DH assumptions, making many of the cryptographic constructions we rely on today unfit for purpose.

As a consequence of the potential risk posed by hypothetical future quantum computers, the topic of *post-quantum cryptography*, that is the research and development of cryptographic techniques not rendered insecure by quantum computers, has gained great traction. In 2016, Google conducted its first post-quantum cryptography at-scale test [Lan16]. Later the same year, the US National Institute of Standards and Technology (NIST) started a process to standardise post-quantum public-key encryption and digital signature algorithms [Nat16], to make these technologies available both to industry and to the public.

Whatever we may think of the timeline or even plausibility of the arrival of large-scale general quantum computers, post-quantum cryptography continues gaining momentum. Given the long shelf life of cryptographic standards and the high stakes of standardising primitives, the security of these schemes, and thus the concrete hardness of post-quantum mathematical problems, should be understood in detail.

Some of the most popular mathematical objects used to provide post-quantum security are lattices. Lattices consist of integer linear combinations of a set of

[3] "Noisy" as in not very precise. Personally, I have no idea whether they are loud, we can only hope.

vectors called a basis. Given a lattice basis, various computational problems can be defined, most famously that of finding the shortest non-zero vector in the lattice it describes. The research presented in this thesis focuses on some aspects of the concrete security and practicality of *lattice-based cryptography*, cryptography that uses lattice problems as its core hardness assumption.

0.1 Thesis overview

In this thesis, we present research carried out as part of four different publications, the content of each being presented in a different chapter. Overall, our work covers three different topics in post-quantum cryptography, with Chapters 3 and 4 investigating the same topic.

Throughout these chapters, special care has been put in trying to make our work reproducible. In the process of writing the four publications, source code was produced to further our understanding of the subject of study and to verify our hypotheses. In almost all cases this source code has been made publicly available, with any improvements made to third party tools being contributed back to their original projects, and any issues found with them being reported. The only exception has been the smart card implementation in Chapter 5, which could not be published due to intellectual property concerns of collaborators. However, proof of concept code demonstrating the fundamental algorithmic techniques employed in that chapter has been released.

We now give an overview of the content of each chapter.

Chapter 1. This chapter covers mathematical notation and preliminary notions used throughout the thesis. It also provides basic notions about lattices and quantum computation.

Chapter 2. Originally published as

> Jaques S., Naehrig M., Roetteler M., Virdia F.
> *Implementing Grover Oracles for Quantum Key Search on AES and LowMC.*
> In: Canteaut A., Ishai Y. (eds) Advances in Cryptology, Eurocrypt 2020.
> Lecture Notes in Computer Science, vol 12106. Springer, Cham.

As part of the call for standardisation of post-quantum public-key encryption and signature schemes [Nat16], the US NIST proposed a criterion to determine whether a considered scheme is secure enough. Quantum algorithmic speedups have so far proved less dramatic when attacking block ciphers, such as the Advanced Encryption Standard (AES) [DR01]. Therefore, NIST defined security "categories" that a proposed construction can achieve based on the three versions of AES. The assumption is that any scheme as hard to break as AES should be secure for the foreseeable future, with a scheme as hard to break as AES-256 being more secure ("Category 5") than one not significantly harder to break than AES-128 ("Category 1"). The comparative nature of these definitions of security means that establishing the cost of attacking AES using a quantum computer becomes an essential step required to carry out the standardisation process, since this cost becomes a criterion for adequacy.

Making use of recently developed tools [SGT+18] for designing and simulating quantum circuits, in Chapter 2 we investigate the non-asymptotic costs of attacking AES using a quantum computer, under a set of assumptions on the architecture of such a machine. Our techniques show that AES is slightly easier to attack than NIST predicted (under the same assumptions), meaning that in this particular model post-quantum security is slightly easier to achieve than first suggested. We also extend our analysis to the LowMC block cipher [ARS+15], which plays an important role as a building block for one of the digital signature schemes proposed for standardisation.

The author of this thesis contributed towards the chapter's concept, the design and implementation of the Grover oracles, the analysis of the impact on the definitions of security proposed by NIST.

Chapter 3. Originally published as

> Albrecht M. R., Göpfert F., Virdia F., Wunderer T.
> *Revisiting the Expected Cost of Solving uSVP and Applications to LWE.*
> In: Takagi T., Peyrin T. (eds) Advances in Cryptology, Asiacrypt 2017.
> Lecture Notes in Computer Science, vol 10624. Springer, Cham.

Many lattice-based cryptographic schemes rely on the hardness of the Learning With Errors (LWE) problem, or a variant thereof, to provide security. The two most effective approaches to solving LWE are the *primal* and *dual* lattice attacks. In Chapter 3, we present an experimental investigation of the two different approaches used for estimating the complexity of the primal attack at the time of the chapter's publication. We verify that results align with the approach presented in [ADPS16], with minor explainable deviations from the expected behaviour. With such experimental evidence, we proceed to implement the [ADPS16] heuristic in the popular "LWE estimator" script[4] [APS15] and use this to demonstrate how the change in heuristic affects the estimated security of some cryptosystems proposed in the literature.

The author of this thesis contributed towards the design, implementation and running of all experiments, the analysis of the results, the analysis of the impact of the [ADPS16] approach on previously proposed cryptographic parameters.

Chapter 4. Originally published as

> Postlethwaite E. W., Virdia F.
> *On the Success Probability of Solving Unique SVP via BKZ.*
> In: Garay J.A. (eds) Public-Key Cryptography, 2021.
> Lecture Notes in Computer Science, vol 12710. Springer, Cham.

The heuristic presented in [ADPS16] for estimating the cost of the primal attack only predicts the expected cost for a random instance of the LWE problem. This however does not allow estimating the probability of the attack being cheaper than expected, a phenomenon sometimes observed in our experiments in Chapter 3. In Chapter 4, we repurpose the refined heuristic for computing

[4] `https://bitbucket.org/malb/lwe-estimator`.

the expected cost proposed in [DDGR20] to account for the probabilistic nature of lattice attacks. This allows us to predict the success probability of cheaper instances of the primal attack. We provide a heuristic analysis and extensive experiments and demonstrate how such an analysis could potentially impact the estimated cost of attacking three key encapsulation schemes proposed to NIST for standardisation.

The author of this thesis contributed towards all aspects of this chapter.

Chapter 5. Originally published as

> Albrecht M. R., Hanser C., Hoeller A., Pöppelmann T., Virdia F., Wallner A.
> *Implementing RLWE-based Schemes Using an RSA Co-Processor.*
> IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019(1).

In Chapter 5, we move our attention to the practicality of lattice-based encryption. We look at the space of embedded devices, which present some of the harshest restrictions on hardware capabilities. These devices are usually augmented with specific hardware arithmetic coprocessors to enable them to efficiently run the expensive computations required by RSA- and DH-based cryptography. We investigate to what extent these "pre-quantum" coprocessors can be used to speed up post-quantum lattice-based public-key encryption. Our experiments suggest that the resulting implementations are fast enough to potentially be used as a transitional approach to deploying post-quantum public-key encryption on embedded hardware while waiting for specialised lattice coprocessors to be developed and certified.

The author of this thesis contributed towards the design of all arithmetic strategies used to run computations on the RSA coprocessor, to their proof of correctness, and to their proof of concept implementation.

Chapter 1

Background

Contents

In this chapter we set the notation used in the rest of the thesis, and recall some fundamental background that our work builds upon.

1.1 Notation

For $x \in \mathbb{R}$, we write $\lfloor x \rceil$ to mean the closest integer to x (where $\lfloor y + \frac{1}{2} \rceil \coloneqq y + 1$ for $y \in \mathbb{Z}$). For $a, b \in \mathbb{Z}$, we write $a \bmod^{(+)} b$ or $[a]_b$ for the unique integer $\hat{a} \equiv a \bmod b$ such that $0 \leq \hat{a} < b$. We write $a \bmod^{(-)} b$ for the unique integer $\hat{a} \equiv a \bmod b$ such that $-b/2 \leq \hat{a} < b/2$. We may refer to an integer $a \in \mathbb{Z}_b$ as being "small" if $a \bmod^{(-)} b$ is small, smallness depending on the context. We extend this definition to tuples, vectors, matrices and polynomials over $\mathbb{Z}$ component-wise. Similarly as for integers, we may refer to a vector $\mathbf{v} \in \mathbb{Z}_b^n$ as being "short" if $\mathbf{v} \bmod^{(-)} b$ is short. We often write $\{a, \ldots, b\}$ to mean

the set $[a, b] \cap \mathbb{Z}$. We denote by $[n]$ the set $\{1, \ldots, n\}$. We write log to mean logarithms in base 2 and ln to mean logarithms in base e. We denote by $\|\cdot\|_p$ the ℓ_p norm, for $p \in \{\infty\} \cup \mathbb{Z}_{\geq 1}$. If unspecified, $p = 2$. We write

$$[\![\text{condition}]\!] \coloneqq \begin{cases} 1 & \text{if condition is true,} \\ 0 & \text{if condition is false.} \end{cases}$$

Linear algebra. We denote vectors by bold lowercase letters such as $\mathbf{v}$, and matrices by bold uppercase letters such as $\mathbf{M}$, and refer to their entries with a subscript index v_i, $M_{i,j}$, counting indices from 1. By abuse of notation we consider vectors to be row resp. column vectors depending on context, such that $\mathbf{vM}$ and $\mathbf{Mv}$ are meaningful. In particular, given column vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, we may write the concatenated vector $\mathbf{w} = (\mathbf{v}_1 \mid \mathbf{v}_2)$ as a row vector implicitly transposing the $\mathbf{v}_i$ vectors in order to avoid overloading the notation. We write inner products using angular brackets $\langle \mathbf{v}, \mathbf{w} \rangle$. The transpose of $\mathbf{v}$ is indicated as $\mathbf{v}^t$, while the adjoint of $\mathbf{v}$ (the complex conjugate of its transpose) is indicated as $\mathbf{v}^\dagger$. We write $\mathbf{I}_m$ for the $m \times m$ identity matrix over whichever base ring is implied from context. We write $\mathbf{0}_{m \times n}$ for the $m \times n$ all zero matrix. If the dimensions are clear from the context, we may omit the subscripts. Given a set of vectors S of size n and a ring R (e.g. $R = \mathbb{Z}$ or $\mathbb{R}$), we write $\text{span}_R(S)$ to mean the R-span of S, i.e. $\{\sum_{i=1}^{n} \mu_i \mathbf{v}_i \colon \mathbf{v}_i \in S,\, \mu_i \in R\}$.

Probability. Given a probability distribution D with support $S \subset \mathbb{R}$, we denote sampling an element $s \in S$ according to D as $s \leftarrow D$. If the sampling is done using coins r, we write $s \xleftarrow{r} D$. For a bounded support S, we denote the uniform distribution over S as $U(S)$. We denote the mean and variance of D as $\mathbb{E}(s)$ or $\mathbb{E}(D)$, and $\mathbb{V}(s)$ or $\mathbb{V}(D)$, respectively, where $s \leftarrow D$ and $\mathbb{V}(s) = \mathbb{E}(s^2) - \mathbb{E}(s)^2$. We sometimes use $\sqrt{\mathbb{V}}$ similarly to denote the standard deviation of a random variable. Given a discrete (resp. continuous) probability distribution D, we denote its probability mass function (resp. probability density function) as f_D and its cumulative mass function (resp. cumulative density function) as F_D. Given $s \leftarrow D$, by definition $P[s \leq x] = F_D(x)$.

Rings. We identify polynomials $f = \sum_i f_i x^i$ of degree $n-1$ with their corresponding coefficient vector $\mathbf{f} = (f_0, f_1, \ldots, f_{n-1})$. We abuse notation and write $\|f\|$ to mean the Euclidean norm of $\mathbf{f}$. Unless stated otherwise, we work in the polynomial rings $R = \mathbb{Z}[x]/(x^n+1)$ where n is a power of 2, and $R_q = R/(q)$ for some positive integer q. We let R^k (resp. R_q^k) be a ring module of dimension k over R (resp. R_q). Throughout we identify equivalence classes in R_q with their representative polynomial with coefficients $\bmod^{(-)} q$, and their lifted versions in R and in $\mathbb{Z}[x]$.

Asymptotic notation. Given two functions $f, g\colon \mathbb{R} \to \mathbb{R}$, following [Knu76] and [CLRS09, § 3.1] we write

- $f \in O(g)$ if there exists $y \in \mathbb{R}$ such that $|f(x)| < cg(x)$ for some $c > 0$ and all $x > y$,
- $f \in o(g)$ if $f(x)/g(x) \to 0$ as $x \to \infty$,
- $f \in \Omega(g)$ if and only if $g \in O(f)$,
- $f \in \omega(g)$ if and only if $g \in o(f)$,
- $f \in \Theta(g)$ if $f \in O(g) \cap \Omega(g)$, meaning that there exists some $y \in \mathbb{R}$ such that $c_1 g(x) < f(x) < c_2 g(x)$ for some $c_2 > c_1 > 0$ and all $x > y$ (which implies that $f \in \Theta(g) \Leftrightarrow g \in \Theta(f)$).

1.2 Linear algebra

We represent a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of $\mathbb{R}^d$ as the matrix $\mathbf{B}$ having the basis vectors as rows. Given a basis $\mathbf{B}$, we can derive an orthogonal basis $\mathbf{B}^*$ via the Gram–Schmidt process. The rows of $\mathbf{B}^*$ are

$$\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j<i} \mu_{i,j} \mathbf{b}_j^* \quad \text{for} \quad i \in [d], \quad \text{where} \quad \mu_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle / \|\mathbf{b}_j^*\|^2 \quad \text{for} \quad i > j.$$

We may refer to the $\mathbf{b}_i^*$ vectors as the Gram-Schmidt vectors of $\mathbf{B}$. In matrix form, the two bases are related by

$$\begin{bmatrix} \text{—} & \mathbf{b}_1 & \text{—} \\ & \vdots & \\ \text{—} & \mathbf{b}_d & \text{—} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ \mu_{2,1} & 1 & & \\ \vdots & \ddots & \ddots & \\ \mu_{d,1} & \dots & \mu_{d,d-1} & 1 \end{bmatrix} \begin{bmatrix} \text{—} & \mathbf{b}_1^* & \text{—} \\ & \vdots & \\ \text{—} & \mathbf{b}_d^* & \text{—} \end{bmatrix}.$$

Given a basis $\mathbf{B}$ of $\mathbb{R}^d$ we denote by $\pi_{\mathbf{B},k} \colon \mathbb{R}^d \to \mathbb{R}^d$ the linear operator projecting vectors orthogonally to the subspace $\operatorname{span}_{\mathbb{R}}(\{\mathbf{b}_1, \dots, \mathbf{b}_{k-1}\})$. Note $\pi_{\mathbf{B},1}$ is the identity on $\mathbb{R}^d$. We write π_i when the basis is clear from context. Given a vector space $V = \operatorname{span}_{\mathbb{R}}(\mathbf{B})$, its *projective* subspace $\pi_k(V)$ of dimension $d - k + 1$ has a basis $\{\pi_k(\mathbf{b}_k), \dots, \pi_k(\mathbf{b}_d)\}$, where

$$\pi_k(\mathbf{b}_i) = \mathbf{b}_i - \sum_{j<k} \mu_{i,j}\mathbf{b}_j^* = \mathbf{b}_i^* + \sum_{k \le j < i} \mu_{i,j}\mathbf{b}_j^* \quad \text{for} \quad i \ge k.$$

By definition, this implies that $\pi_k(\mathbf{b}_k) = \mathbf{b}_k^*$, and that $\pi_j(\pi_k(\mathbf{v})) = \pi_k(\mathbf{v})$ for any $j \le k$. Given an orthogonal basis $\mathbf{B}^*$ and a vector $\mathbf{v} = v_1^*\mathbf{b}_1^* + \cdots + v_d^*\mathbf{b}_d^*$, its projections are given by $\pi_k(\mathbf{v}) = v_k^*\mathbf{b}_k^* + \cdots + v_d^*\mathbf{b}_d^*$. We abuse notation and write $\pi_k(\mathbf{B}_{[i:j]})$ to mean the matrix with rows $\pi_k(\mathbf{b}_i), \dots, \pi_k(\mathbf{b}_j)$. For example, the matrix $\pi_k(\mathbf{B}_{[k:d]})$ is

$$\begin{bmatrix} \text{—} & \pi_k(\mathbf{b}_k) & \text{—} \\ & \vdots & \\ \text{—} & \pi_k(\mathbf{b}_d) & \text{—} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ \mu_{k+1,k} & 1 & & \\ \vdots & \ddots & \ddots & \\ \mu_{d,k} & \dots & \mu_{d,d-1} & 1 \end{bmatrix} \begin{bmatrix} \text{—} & \mathbf{b}_k^* & \text{—} \\ & \vdots & \\ \text{—} & \mathbf{b}_d^* & \text{—} \end{bmatrix}.$$

1.3 Probability

For real valued, independent random variables X, Y, Z and scalars $\lambda \in \mathbb{R}$, we have the following useful identities,

$$\begin{aligned} \mathbb{E}(X+Y) &= \mathbb{E}(X) + \mathbb{E}(Y), & \mathbb{E}(X \cdot Y) &= \mathbb{E}(X) \cdot \mathbb{E}(Y), \\ \mathbb{V}(X) &= \mathbb{E}(X^2) - \mathbb{E}(X)^2, & \mathbb{V}(\lambda X) &= \lambda^2 \mathbb{V}(X), \\ \mathbb{V}(X+Y) &= \mathbb{V}(X) + \mathbb{V}(Y), & \mathbb{V}(X \cdot Y) &= \mathbb{E}(X)^2 \mathbb{V}(Y) + \mathbb{E}(Y)^2 \mathbb{V}(X) \\ & & & \quad + \mathbb{V}(X)\mathbb{V}(Y). \end{aligned}$$

We recall the conditional probability chain rule. If $E_1, \ldots, E_n$ are events, then

$$P[E_1 \cap \cdots \cap E_n] = P[E_1 | E_2 \cap \cdots \cap E_n] P[E_2 \cap \cdots \cap E_n].$$

1.3.1 The Gaussian distribution

We recall some properties of the continuous Gaussian distribution. We denote by $N(\mu, \sigma^2)$ the probability distribution over $\mathbb{R}$ of mean μ and standard deviation σ, variance σ^2, with density function

$$f_{N(\mu,\sigma^2)}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

Given a random variable $X \sim N(\mu_X, \sigma_X^2)$ and a scalar $\lambda > 0$, the random variable $Y = \lambda \cdot X$ follows a distribution $N(\lambda\mu_X, \lambda^2\sigma_X^2)$. Given n independent and identically distributed random variables $X_i \sim N(0, 1)$, the random variable $X_1^2 + \cdots + X_n^2$ follows a chi-squared distribution χ_n^2 over $\mathbb{R}_{\geq 0}$ of mean n and variance $2n$, with probability density function

$$f_{\chi_n^2}(x) = \frac{1}{2^{n/2}\Gamma(n/2)}\, x^{n/2-1} e^{-x/2},$$

where Γ denotes the gamma function

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \quad \text{for} \quad x > 0.$$

Given n independent and identically distributed random variables $Y_i \sim N(0, \sigma^2)$, the random variable $Y_1^2 + \cdots + Y_n^2$ follows a distribution $\sigma^2 \cdot \chi_n^2$ of mean $n\sigma^2$ and variance $2n\sigma^4$, that is, a chi-squared distribution where every sample is scaled by a factor of σ^2. We call this a *scaled* chi-squared distribution.

Discrete Gaussians. We denote by $D_{\mu,\sigma}$ the discrete Gaussian distribution over $\mathbb{Z}$ with mean $\mu \in \mathbb{R}$ and standard deviation $\sigma \in \mathbb{R}^+$. It has probability mass function $f_{D_{\mu,\sigma}} \colon \mathbb{Z} \to [0, 1], x \mapsto f_{N(\mu,\sigma^2)}(x)/f_{N(\mu,\sigma^2)}(\mathbb{Z})$, where $f_{N(\mu,\sigma^2)}(\mathbb{Z}) = \sum_{x\in\mathbb{Z}} f_{N(\mu,\sigma^2)}(x)$. Discrete Gaussian distributions with $\mu = 0$, or the distributions these imply over $\mathbb{Z}_q$ for some modulus q, are widely used in lattice-based cryptography to sample entries of error and secret vectors from. In our analyses in Chapters 3 and 4, we will work with vectors $\mathbf{v}$ sampled

coefficient-wise from a discrete Gaussian distribution, and with their projections $\pi_i(\mathbf{v})$. We model the the squared norms $\|\pi_i(\mathbf{v})\|^2$ as continuous[1] random variables following a scaled chi-squared distribution with the appropriate degrees of freedom. For example, for some vector $\mathbf{v} = (v_1, \ldots, v_d)$ with each $v_i \leftarrow D_{0,\sigma}$ sampled independently, we model $\|\pi_{\mathbf{B},i}(\mathbf{v})\|^2 \sim \sigma^2 \cdot \chi^2_{d-i+1}$, where $\mathbf{B}$ is a lattice basis being reduced.

Rounded Gaussians. Similarly to the discrete Gaussian distribution, the *rounded* Gaussian distribution of mean μ and variance σ^2 is a distribution over $\mathbb{Z}$ obtained by sampling $x \leftarrow N(\mu, \sigma^2)$ and returning $\lfloor x \rceil$.

Centered binomial distribution. The centered binomial distribution (CBD) was proposed for use as an alternative to discrete Gaussian distributions by Alkim *et al.* [ADPS16], since it results in a similarly shaped mass cumulative function and is faster and easier to sample in a side-channel resistant fashion. Given a parameter η, a sample $x \leftarrow \mathrm{CBD}_\eta$ is distributed over $[-\eta, \eta] \cap \mathbb{Z}$, such that $x + \eta$ follows a binomial distribution where the yes-no events have equal probability $P[\text{yes}] = P[\text{no}] = 1/2$, and $x + \eta$ is the number of "yes" events after 2η fair coin flips.

1.4 Lattices

In this section we provide fundamental facts about lattices. For a more comprehensive treatment and proofs, see [MR09, NV10, Gal12].

Definition 1 (Real and integer lattices)**.** *Let $\mathbf{b}_1, \ldots, \mathbf{b}_n$ be n linearly independent row vectors in $\mathbb{R}^d$, which we collect into a basis $\mathbf{B}$. We say that their integer span*

$$\Lambda = \Lambda(\mathbf{B}) = \mathrm{span}_{\mathbb{Z}}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{x_1\mathbf{b}_1 + \cdots + x_n\mathbf{b}_n : x_i \in \mathbb{Z}\},$$

[1] While $\mathbf{v}$ has integer coefficients, its $\pi_i(\mathbf{v})$ projections will not necessarily do so.

is a real lattice *of dimension, or rank, n. If $n = d$ we say the lattice is* full-rank. *If $\Lambda \subset \mathbb{Z}^d$, we say that Λ is an* integer lattice. *It should be noted that any real lattice is a subgroup of $(\mathbb{R}^d, +)$, and any integer lattice is a subgroup of $(\mathbb{Z}^d, +)$.*

In this thesis, we concern ourselves in particular with q-ary lattices.

Definition 2 (q-ary lattices)**.** *Given an integer q, Λ is a* q-ary lattice *if it is an integer lattice such that $q\mathbb{Z}^d \subseteq \Lambda \subseteq \mathbb{Z}^d$.*

Remark 1. *Since a q-ary lattice Λ has $q\mathbb{Z}^d$ as a subgroup, for any $\mathbf{v} \in \Lambda$, we have the coset $\mathbf{v} + q\mathbb{Z}^d \subseteq \Lambda$. This means that Λ can be seen as a subgroup of $\mathbb{Z}_q^d$ rather than $\mathbb{Z}^d$ by reducing vectors modulo q. Throughout this thesis, we will sometimes abuse notation and make no distinction between considering $\Lambda \subset \mathbb{Z}^d$ and $\Lambda \subset \mathbb{Z}_q^d$ when dealing with q-ary lattices.*

A given lattice Λ can have multiple different lattice bases. Indeed $\Lambda(\mathbf{B}) = \Lambda(\mathbf{B}')$ if and only if $\mathbf{B}' = \mathbf{UB}$ for some unimodular matrix $\mathbf{U}$, that is a $n \times n$ integer matrix with determinant ± 1.

Given a lattice basis, we can define its *fundamental parallelepiped* $P(\mathbf{B})$.

Definition 3 (Fundamental parallelepiped)**.** *Given a lattice basis $\mathbf{B}$, its fundamental parallelepiped is the set*

$$P(\mathbf{B}) \coloneqq \{x_1\mathbf{b}_1 + \cdots + x_n\mathbf{b}_n \colon x_i \in [0, 1)\}.$$

An invariant of a lattice is its volume (also called *covolume* or *determinant*).

Definition 4 (Lattice volume)**.** *Given any basis $\mathbf{B}$ for a lattice Λ, the volume of Λ is*

$$\mathrm{vol}(\Lambda) \coloneqq \sqrt{\det(\mathbf{B}\mathbf{B}^t)}$$

Remark 2. *The volume of Λ is exactly the volume of $P(\mathbf{B})$. In the case of full-rank lattices, this is also exactly $|\det(\mathbf{B})|$. Similarly, given the Gram-Schimdt vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ of the basis $\mathbf{B}$,* $\mathrm{vol}(\Lambda) = \prod_{i=1}^{n} \|\mathbf{b}_i^*\|$.

Definition 5 (Sublattices)**.** *Let $\Lambda \subset \mathbb{R}^n$ be a real lattice of rank n. We call any subgroup $\Lambda' \subset \Lambda$ a sublattice of Λ. Sublattices may potentially have smaller rank or larger volume than Λ.*

Example 3. *Let* $\mathbf{e}_1, \ldots, \mathbf{e}_n \in \mathbb{R}^n$ *be the canonical basis of* $\mathbb{R}^n$*, where* $\mathbf{e}_i$ *has all coefficients set to* 0 *except for the* i*-th coefficient set to* 1*. Let* $\Lambda = \mathrm{span}_{\mathbb{Z}}(\mathbf{e}_1, \ldots, \mathbf{e}_n)$ *be an integer lattice of rank* n *and volume* 1*.*

- $\mathrm{span}_{\mathbb{Z}}(\mathbf{e}_2, \ldots, \mathbf{e}_n)$ *is a sublattice of rank* $n-1$ *and volume* 1*.*
- $\mathrm{span}_{\mathbb{Z}}(2 \cdot \mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n)$ *is a sublattice of rank* n *and volume* 2*.*
- $\mathrm{span}_{\mathbb{Z}}(2 \cdot \mathbf{e}_2, \mathbf{e}_3 \ldots, \mathbf{e}_n)$ *is a sublattice of rank* $n-1$ *and volume* 2*.*

Of particular interest to us will be working with *projective* sublattices.

Definition 6 (Projective sublattices)**.** *Given a lattice* Λ *of rank* n *and a basis* $\mathbf{b}_1, \ldots, \mathbf{b}_n$ *of* Λ*, we denote by* $\Lambda_{i,j}^{\perp}$ *the lattice with basis* $\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_j)$*, where* $i \leq j$*. If* $j = n$*, we may write* $\Lambda_i^{\perp}$ *instead. We say* $\Lambda_{i,j}^{\perp}$ *is a* projective sublattice*.*

Remark 4. *While we call the* $\Lambda_{i,j}^{\perp}$ *projective* sublattices*, they are not sublattices of* Λ*, but rather orthogonal projections of sublattices of* Λ*.*

A property of interest of a lattice is the set of its successive minima.

Definition 7 (Successive minima)**.** *Let* $B_d(r)$ *be the closed ball of radius* r *in* $\mathbb{R}^d$ *centered around* 0*, and let* $i \in [d]$*. We define the* i*-th minima of* Λ *as*

$$\lambda_i(\Lambda) = \min\left\{r \in \mathbb{R}^+ \colon \Lambda \cap B_d(r) \text{ contains } i \text{ linearly independent vectors}\right\}.$$

A quantity of interest when working with worst-case bounds on the successive minima of a lattice is Hermite's constant.

Definition 8 (Hermite's constant [Her50])**.** *Let* $\mathcal{L}_n$ *be the set of real lattices of rank* n*. Then Hermite's constant for rank* n *lattices,* γ_n*, is defined as*

$$\gamma_n \coloneqq \sup_{\Lambda \in \mathcal{L}_n} \frac{\lambda_1(\Lambda)^2}{\mathrm{vol}(\Lambda)^{2/n}}.$$

Hermite's constant is known for a few values of n [Mar03, § 6]. In particular, $\gamma_2 = \sqrt{4/3}$. A classical bound on Hermite's constant is given by Hermite's inequality.

Theorem 1 (Hermite's inequality [Her50])**.** *Let $n \geq 2$ be an integer. Then*

$$\gamma_n \leq \gamma_2^{n-1}.$$

Corollary 2. *Given any lattice Λ of rank n, it contains a vector $\mathbf{v}$ of norm*

$$\|\mathbf{v}\| \leq \gamma_2^{(n-1)/2} \cdot \mathrm{vol}(\Lambda)^{1/n}.$$

A generalisation to Hermite's inequality is given by Mordell.

Theorem 3 (Mordell's inequality [Mor44])**.** *Let $k \geq 2$ and $n \geq k$ be integers. Then*

$$\gamma_n^{k-1} \leq \gamma_k^{n-1}.$$

While Hermite's and Mordell's inequalities can be used to provide provable bounds on the norm of the shortest vector in any lattice, these bounds are not necessarily tight on average. However, heuristic estimates on the length of the shortest vector in random lattices can be obtained. Indeed, a lattice can be tessellated by centring a copy of the fundamental parallelepiped on each lattice point, and this fact can be used to approximate the number of lattice points in some "nice enough" measurable set. Using this intuition, the *Gaussian heuristic* says that the number of lattice points in a measurable set S is approximately $\mathrm{vol}(S)/\mathrm{vol}(\Lambda)$. This can be used to approximate the first minimum $\lambda_1(\Lambda)$, by assuming its value is the radius of the ball of volume $\mathrm{vol}(\Lambda)$.

Heuristic 1 (Gaussian heuristic for the shortest vector)**.** *Given a lattice Λ of rank n, the Gaussian heuristic approximates the first minimum of Λ as the radius of the smallest n-dimensional ball of volume* $\mathrm{vol}(\Lambda)$*,*

$$\begin{aligned}\lambda_1(\Lambda) &\approx \frac{\Gamma(1+n/2)^{1/n}}{\sqrt{\pi}}\mathrm{vol}(\Lambda)^{1/n} \\ &\approx (\pi n)^{\frac{1}{2n}}\sqrt{\frac{n}{2\pi e}}\mathrm{vol}(\Lambda)^{1/n} \quad \textit{by [Fel68, §II.9].}\end{aligned}$$

Hermite's constant and the Gaussian heuristic tell us something about the first minimum of a lattice. However, in general it is not easy to find a vector realising the first minimum from a random lattice basis. In Section 1.4.1.2 we will look at worst-case guarantees and average-case heuristics on the norm of vectors that can be found from a basis using lattice reduction.

Computational problems. Various computational problems can be defined using lattices. In this thesis we will often mention the following problems, and we will discuss lattice reduction as a means to solve some of them.

Definition 9 (Shortest Vector Problem (SVP))**.** *Given a lattice* Λ *find a vector* $\mathbf{v} \in \Lambda$ *of norm* $\lambda_1(\Lambda)$*.*

Definition 10 (γ-gap Shortest Vector Problem (GapSVP$_\gamma$))**.** *Given a lattice* Λ *and a real* $d > 0$*, decide whether* $\lambda_1(\Lambda) \leq d$ *or* $\lambda_1(\Lambda) > \gamma \cdot d$*.*

Definition 11 (γ-unique Shortest Vector Problem (uSVP$_\gamma$))**.** *Given a lattice* Λ *such that* $\lambda_2(\Lambda) > \gamma\lambda_1(\Lambda)$*, find the unique (up to sign) vector* $\mathbf{v} \in \Lambda$ *of norm* $\lambda_1(\Lambda)$*. Unless specified,* $\gamma = 1$*.*

Definition 12 (γ-approximate Shortest Vector Problem (approx-SVP$_\gamma$))**.** *Given a lattice* Λ*, find a non-zero vector* $\mathbf{v} \in \Lambda$ *of norm* $\leq \gamma \cdot \lambda_1(\Lambda)$*.*

Definition 13 (ζ-Hermite Shortest Vector Problem (Hermite-SVP$_\zeta$))**.** *Given a lattice* Λ *of rank* n*, find a non-zero vector* $\mathbf{v} \in \Lambda$ *of norm* $\leq \zeta \cdot \mathrm{vol}(\Lambda)^{1/n}$*.*

Definition 14 (α-Bounded Distance Decoding (BDD$_\alpha$))**.** *Given a lattice basis* $\mathbf{B}$ *and a vector* $\mathbf{v}$ *such that dist*$(\mathbf{v}, \mathbf{B}) < \alpha\lambda_1(\mathbf{B})$ *where dist*$(\mathbf{v}, \mathbf{B})$ *denotes the smallest Euclidean distance between* $\mathbf{v}$ *and any lattice point in* $\Lambda(\mathbf{B})$*, find the (unique) lattice vector* $\mathbf{t} \in \Lambda(\mathbf{B})$ *closest to* $\mathbf{v}$*.*

Definition 15 (γ-Shortest Independent Vectors Problem (SIVP$_\gamma$))**.** *Given a lattice* Λ *of rank* n*, find* n *linearly independent lattice vectors* $\mathbf{v}_i \in \Lambda$ *of norm at most* $\gamma \cdot \lambda_n(\Lambda)$*.*

Definition 16 (β-Short Integer Solution problem (SIS$_\beta$))**.** *Given a matrix* $\mathbf{A} \leftarrow U(\mathbb{Z}_q^{n \times m})$*, find a non-zero vector* $\mathbf{v} \in \mathbb{Z}^m$ *such that* $\mathbf{A}\mathbf{v} = \mathbf{0} \bmod q$ *and* $\|\mathbf{v}\|_p < \beta$*. Usually we consider* $p = 2$ *or* ∞*.*

1.4.1 Lattice reduction

Informally, lattice reduction is any algorithmic technique that takes as input a basis of a lattice and finds a basis of better *quality*. Many different notions of basis reducedness exist, which usually can be intuitively captured by a basis being formed of short and close to orthogonal vectors.

1.4.1.1 Reduction algorithms

A fundamental notion in lattice reduction is that of a size-reduced lattice basis. Size-reduction (Algorithm 1) is a component of many lattice reduction algorithms. Given a lattice basis, it returns a size-reduced basis of the same lattice.

Definition 17 (Size-reduced)**.** *Let* $\mathbf{B}$ *be a lattice basis,* $\{\mathbf{b}_i^*\}_i$ *its Gram-Schmidt vectors and* $\mu_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle / \|\mathbf{b}_j^*\|^2$*.* $\mathbf{B}$ *is* size-reduced *if* $|\mu_{i,j}| \leq 1/2$ *for* $1 \leq j \leq i \leq n$*.*

Input: lattice basis $\mathbf{B}$
1 **for** $i \leftarrow 2$ **to** d **do**
2 **for** $j \leftarrow i-1$ **to** 1 **do**
3 $\mu_{ij} \leftarrow \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle / \|\mathbf{b}_j^*\|^2$
4 $\mathbf{b}_i \leftarrow \mathbf{b}_i - \lfloor \mu_{ij} \rceil \mathbf{b}_j$

Algorithm 1: Size-reduction.

The celebrated LLL algorithm [LLL82] (Algorithm 2) achieves the following notion of basis reducedness, while terminating in polynomial time.

Definition 18 (LLL reduced)**.** *For* $\delta_{L^3} \in (1/4, 1)$ *a basis* $\mathbf{B}$ *is* δ_{L^3}*-LLL reduced if it is size-reduced and* $\delta_{L^3} \cdot \|\mathbf{b}_i^*\|^2 \leq \|\pi_i(\mathbf{b}_{i+1})\|^2$ *holds for all* $i \in [d-1]$*, where the second constraint is also called "Lovász' condition".*

Input: lattice basis $\mathbf{B}$
Input: $\delta_{L^3} \in (0.25, 1)$
1 Run size-reduction on $\mathbf{B}$
2 **for** $i \leftarrow 1$ **to** $d-1$ **do**
3 **if** $\delta_{L^3} \cdot \|\mathbf{b}_i^*\|^2 > \|\pi_i(\mathbf{b}_{i+1})\|^2$ **then**
4 $\mathbf{b}_i, \mathbf{b}_{i+1} \leftarrow \mathbf{b}_{i+1}, \mathbf{b}_i$
5 **go to** *line 1*

Algorithm 2: LLL.

In [SE91, SE94], Schnorr and Euchner introduced a generalisation of LLL called BKZ (Algorithm 3). While LLL works on adjacent pairs of basis vectors, checking where the Lovász' condition does not hold and addressing this by swapping the basis vectors (Lines 3 and 4 of Algorithm 2), BKZ works with *blocks* of β adjacent basis vectors, and enforces the first Gram-Schmidt vector

in the block to be the shortest in a projective sublattice spanned by orthogonal projections of the block's vectors (Lines 4 to 9 of Algorithm 3).[2] Given a block size β, the basis output by BKZ-β is reduced in the following sense.

Definition 19 (BKZ-β reduced)**.** *A basis* $\mathbf{B}$ *is BKZ-*β *reduced if it is LLL reduced and* $\|\mathbf{b}_i^*\| = \lambda_1\left(\Lambda^{\perp}_{i,\min(i+\beta-1,d)}\right)$ *for all* $i \in [d-1]$.[3]

In order to do this, an oracle O_{SVP} is used, that, given a lattice, finds a vector attaining the lattice's first minimum. BKZ repeatedly calls O_{SVP} on the projective sublattices $\Lambda^{\perp}_{i,\min(i+\beta-1,d)}$. If the vector $\mathbf{v}$ output by O_{SVP} is shorter than $\mathbf{b}_i^*$, it is "lifted" (see Line 6 of Algorithm 3) and inserted into the basis at the beginning of the block. Then LLL is run on the basis to remove linear dependencies introduced by this insertion. In Chapters 3 and 4 we will run several experiments to investigate the behaviour of BKZ. We will make use of the popular implementation of BKZ 2.0 [GNR10, CN11] found in the Fplll library [DT17], which sets $\delta_{L^3} = 0.99$ in the underlying calls to LLL. In its original description, BKZ terminates after a full tour

Input: LLL reduced lattice basis $\mathbf{B}$
Input: block size β
1 **repeat** /* tour */
2 **for** $i \leftarrow 1$ **to** d **do**
3 $j \leftarrow \min(i+\beta-1, d)$
4 $\mathbf{v} = x_i\pi_i(\mathbf{b}_i) + \cdots + x_j\pi_i(\mathbf{b}_j) \leftarrow O_{\mathrm{SVP}}(\Lambda^{\perp}_{i,j})$
5 **if** $\|\mathbf{v}\| < \|\mathbf{b}_i^*\|$ **then**
6 $\mathbf{v}' \leftarrow x_i\mathbf{b}_i + \cdots + x_j\mathbf{b}_j$
7 extend $\mathbf{B}$ by inserting $\mathbf{v}'$ into $\mathbf{B}$ at index i
8 LLL on $\mathbf{B}$ to remove linear dependencies
9 drop row with all zero entries
10 **if** *if no insertion was made* **then yield** $\top$ **else yield** $\bot$
11 **if** $\top$ *for all* i **then return**

Algorithm 3: Simplified view of the BKZ Algorithm. The instructions inside the **repeat** context are called a BKZ *tour*.

is executed without insertions. We follow algorithmic improvements and do not necessarily run tours until this point. In particular, the notion of *early termination* (called *auto-abort* in some implementations [DT17]) was

[2] With $\beta = 2$, this would be equivalent to checking for Lovász' condition with $\delta_{L^3} = 1$. However, it is not known whether LLL terminates in polynomial time for $\delta_{L^3} = 1$ [NV10].

[3] The LLL parameter δ_{L^3} is left implicit in this definition.

introduced in [HPS11]. The idea is that the majority of improvement to the basis quality occurs in a few early tours of BKZ, whereas many tours are required before convergence. This approach is sound, since it has been shown that after polynomially many calls to the SVP oracle, the basis does not change much more [HPS11, LN20]. Following experimental analysis of BKZ [Che13, Figure 4.6], Albrecht [Alb17, §2.5] identifies $\tau = 16$ as the number of tours after which little improvement is made to the basis quality for cryptographically-sized parameters. Furthermore, BKZ 2.0 [GNR10, CN11] integrates local block re-randomisation and preprocessing into the originally proposed O_{SVP} oracle, enumeration. In Chapter 4 we will also consider another variant of BKZ by Aono *et al.* [AWHT16] that they name Progressive BKZ. Here, the basis is reduced using increasingly larger block sizes β, running a fixed number of BKZ-β tours for each block size and terminating after reaching a predetermined maximum block size.

1.4.1.2 Basis quality

As mentioned at the beginning of Section 1.4.1, lattice reduction is used to improve the *quality* of a lattice basis. To better reason about basis quality, we first introduce the notion of basis profile.

Definition 20 (Basis profile)**.** *Given a basis* $\mathbf{B}$ *of a lattice of rank* n*, we define the profile of* $\mathbf{B}$ *as the set of squared norms* $\{\|\mathbf{b}_i^*\|^2\}_{i=1}^n$ *of its orthogonal Gram–Schmidt vectors.*

Two metrics are usually considered to indicate the quality of a basis: how short its shortest vector $\mathbf{b}_1$ is (the shortest, the better), and how quickly the basis profile $\{\|\mathbf{b}_i^*\|^2\}_{i=1}^n$ decays (the slower, the better). LLL and block reduction algorithms affect both. In particular, the larger β is chosen in BKZ-β, the slower these norms decay, the closer to orthogonal the basis vectors are, and the shorter the shortest vector in the reduced basis is.

Lattice reduction algorithms usually provide worst-case guarantees and average-case heuristics on the norm of the shortest vector returned $\|\mathbf{b}_1\|$. These can be in terms of the first minimum of the lattice or in terms of its volume.

Definition 21 (Approximation factor)**.** *Let $\mathcal{A}$ be a lattice reduction algorithm and $\mathcal{L}_n$ be the set of real lattices of rank n. We define the approximation factor of $\mathcal{A}$ as the real-valued random variable $\eta_{\mathcal{A},n}$ such that*

$$\Pr[\eta_{\mathcal{A},n} \leq x] = \Pr\left[\frac{\|\mathbf{b}_1\|}{\lambda_1(\Lambda)} \leq x \colon \Lambda \xleftarrow{\$} \mathcal{L}_n;\ \mathbf{b}_1, \ldots, \mathbf{b}_n \xleftarrow{\$} \mathcal{A}(\Lambda)\right],$$

where by $\Lambda \xleftarrow{\$} \mathcal{L}_n$ we mean to sample a random basis for a random lattice Λ. We say $\mathbb{E}[\eta_{\mathcal{A},n}]$ (resp. $\sup \eta_{\mathcal{A},n}$) is the average-case (resp. worst-case) approximation factor of $\mathcal{A}$ in rank n. We drop the subscripts when the algorithm and the rank are understood from the context. We may abuse notation and write η to mean either $\mathbb{E}[\eta]$ or $\sup \eta$.

Definition 22 (Hermite factor)**.** *Let $\mathcal{A}$ be a lattice reduction algorithm and $\mathcal{L}_n$ be the set of real lattices of rank n. We define the Hermite factor of $\mathcal{A}$ as the real-valued random variable $\zeta_{\mathcal{A},n}$ such that*

$$\Pr[\zeta_{\mathcal{A},n} \leq x] = \Pr\left[\frac{\|\mathbf{b}_1\|}{\mathrm{vol}(\Lambda)^{1/n}} \leq x \colon \Lambda \xleftarrow{\$} \mathcal{L}_n;\ \mathbf{b}_1, \ldots, \mathbf{b}_n \xleftarrow{\$} \mathcal{A}(\Lambda)\right],$$

where by $\Lambda \xleftarrow{\$} \mathcal{L}_n$ we mean to sample a random basis for a random lattice Λ. We say $\mathbb{E}[\zeta_{\mathcal{A},n}]$ (resp. $\sup \zeta_{\mathcal{A},n}$) is the average-case (resp. worst-case) Hermite factor of $\mathcal{A}$ in rank n. We drop the subscripts when the algorithm and the rank are understood from the context. We may abuse notation and write ζ to mean either $\mathbb{E}[\zeta]$ or $\sup \zeta$.

Remark 5. *The approximation factor and the Hermite factor can also be found in the literature as the* length defect *and the* Hermite defect *of an output basis [NV10, Chap. 3].*

Remark 6. *In Definitions 21 and 22, we implicitly assume a probability distribution over the bases of lattices in $\mathcal{L}_n$. While defining the notion of a random lattice requires some care [NV10, Chap. 3], in our work we will be concerned with q-ary lattices generated from cryptographic problems. Since for given parameters the set of such lattices is finite, we can consider a random lattice as a lattice being uniformly sampled from its domain, with a basis deterministically constructed from the problem instance's description.*

We say that an algorithm has an average-case (resp. a worst-case) approximation factor η and an average-case (resp. worst-case) Hermite factor ζ if when

provided in input a basis for a lattice Λ of rank n, it returns on average (resp. in the worst-case) a basis containing a vector $\mathbf{b}_1$ (the shortest in the basis) of norm

$$\|\mathbf{b}_1\| \approx \eta \cdot \lambda_1(\Lambda), \quad \|\mathbf{b}_1\| \approx \zeta \cdot \mathrm{vol}(\Lambda)^{1/n},$$

with $\approx$ replaced by $\leq$ in worst-case bounds. For example, LLL with parameter $\delta_{L^3} \in (1/4, 1)$ has worst-case approximation factor $\eta = (\delta_{L^3} - 1/4)^{-(n-1)/2}$ and worst-case Hermite factor $\zeta = (\delta_{L^3} - 1/4)^{-(n-1)/4}$ [NV10, Chap. 2, Thm. 9].

While worst-case provable bounds give us certain guarantees on the output of lattice reduction, these may be overly pessimistic. In cryptanalysis we are often interested in the hardness of solving an average instance of a hard problem. We will therefore dedicate the rest of this section to discussing the average-case output of lattice reduction algorithms. We will use this opportunity also to discuss a small inconsistency commonly found in the lattice-based cryptography literature, that caused an unfortunate discrepancy in one of the papers [AGVW17, Footnote 7] that led to this thesis. In the following discussion we will omit "average-case" when referring to the average-case Hermite factor, and will assume that we work on lattices of rank n.

As mentioned above, a commonly used approach to measure the effectiveness of a lattice reduction algorithm is to compute its Hermite factor ζ. The value of ζ can be measured experimentally, or derived analytically. For example, famously the LLL algorithm has $\zeta = 1.02^{n-1}$ [NS06], with recent work suggesting a possible analytical derivation of this value using abelian sandpile models [DKTW20]. Similarly, experimental observations [GN08b] on the Hermite factor of BKZ-β are supported by the analytical result by Chen [Che13] who for BKZ-β-reduced bases of rank n derives the limit

$$\lim_{n\to\infty} \zeta^{1/n} = \left(\frac{\beta}{2\pi e}(\pi\beta)^{\frac{1}{\beta}}\right)^{\frac{1}{2(\beta-1)}}. \tag{1.1}$$

The fact that experimentally the Hermite factor for BKZ scales exponentially in the rank of the lattice being reduced was noted in [GN08b, § 3.1]. There, the authors suggest that ζ appears to scale as $e^{a\,n+b}$ where a and b are constants that depend on the lattice reduction algorithm (e.g. on the block size β in

the case of BKZ). However, they also suggest that approximating ζ as c^n for some constant c may be sufficient to make "rough estimations". A similar approximation was already present in [NS06, § 4], where in the case of LLL the authors suggest that the value of $(\delta_{L^3} - 1/4)^{-1/4}$ in the worst-case Hermite factor $(\delta_{L^3} - 1/4)^{-(n-1)/4}$ should be replaced on the average by 1.02, "so that the [Hermite factor] becomes $\approx 1.02^n$". The exponential scaling of the Hermite factor led to the introduction in [LP11, § 5.1] of the *root-Hermite factor* δ (not to be confused with the LLL parameter δ_{L^3}), a value such that $\zeta =: \delta^n$.[4] Chen's limit (1.1) was originally stated as "$\lim_{n\to\infty} \delta(n, \beta)$".

However, we would like to argue that a more natural definition for the root-Hermite factor may be $\zeta =: \delta^{n-1}$. Indeed, the $n-1$ exponent would better align with Hermite's and Mordell's inequalities, as well as with the provable guarantees on the output of LLL. Indeed, the value of 1.02^n proposed in [NS06, § 4] would appear to be an approximation from 1.02^{n-1}. Similarly, in the case of the "rough estimations" of [GN08b, § 3.1], approximating the Hermite factor as c^n does also not necessarily appear to match the plots in [GN08b, Fig. 3] (c^n implies that the linear fit for $\log \zeta$ would cross the origin, which does not seem to be the case).

While this change in definition is a minor difference, we will show that it leads to a more natural expression of δ under the Geometric Series Assumption (which we will define soon), and hence allows a natural derivation of Chen's limit (1.1) as an application of the Gaussian heuristic and the Geometric Series Assumption. Using $\delta^{n-1} \coloneqq \zeta$ as the definition of the root-Hermite factor will also resolve a discrepancy between [AGVW17] and [ADPS16] with respect to the exact formula of the BKZ win condition for solving uSVP reported in [ADPS16], and caused by the implicit assumption that $\zeta = \delta^{n-1}$ in [ADPS16]. Therefore, in the rest of this thesis we will define the root-Hermite factor δ as

$$\delta \coloneqq \zeta^{1/(n-1)}.$$

So far we have talked about predicting the norm $\|\mathbf{b}_1\|$ for bases output by lattice reduction. However, heuristic results also exist about the profile of reduced

[4] To the best of our knowledge, the term "root-Hermite factor" had not been used in the literature previous to [LP11].

bases for random q-ary lattices. A popular such heuristic is the Geometric Series Assumption.

Heuristic 2 (Geometric Series Assumption (GSA) [Sch03])**.** *Given a basis* $\mathbf{B}$ *output by a lattice reduction algorithm, the norms of the Gram-Schmidt vectors* $\mathbf{b}_i^*$ *satisfy*

$$\|\mathbf{b}_i^*\| = \alpha^{i-1} \cdot \|\mathbf{b}_1\|$$

for some constant $\alpha \in (0, 1)$*.*

A simple computation allows to deduce the Hermite factor of an algorithm in terms of α.

Lemma 7. *Under the GSA, a lattice reduction algorithm has Hermite factor*

$$\zeta = (\alpha^{-1/2})^{n-1}.$$

Proof. Let Λ be a lattice of rank n and let $\mathbf{b}_1, \ldots, \mathbf{b}_n$ be a reduced basis of Λ satisfying the GSA. By Remark 2 and by direct computation, we have

$$\begin{aligned}
\mathrm{vol}(\Lambda) &= \prod_{i=1}^{n} \|\mathbf{b}_i^*\| \quad \text{by Remark 2} \\
&= \prod_{i=1}^{n} \left(\alpha^{i-1} \cdot \|\mathbf{b}_1\|\right) \quad \text{by the GSA} \\
&= \alpha^{\sum_{i=1}^{n}(i-1)} \cdot \|\mathbf{b}_1\|^n \\
&= \alpha^{\frac{n(n-1)}{2}} \cdot \|\mathbf{b}_1\|^n.
\end{aligned}$$

Taking n-th roots and rearranging terms we get the desired result,

$$\|\mathbf{b}_1\| = (\alpha^{-1/2})^{n-1} \cdot \mathrm{vol}(\Lambda)^{1/n} = \zeta \cdot \mathrm{vol}(\Lambda)^{1/n}.$$

□

Remark 8. *Defining the root-Hermite factor as* $\delta := \zeta^{1/(n-1)}$ *results in* $\delta = \alpha^{-1/2}$ *under the GSA. Using* $\delta := \zeta^{1/n}$ *instead would result in a root-Hermite factor* $\delta = \alpha^{-\frac{n-1}{2n}}$*, slightly dependent on the rank of the lattice being reduced.*

In the case of BKZ-β, α can be also heuristically estimated as a function of β.

Lemma 9. *Under the GSA and the Gaussian heuristic, the basis profile output by BKZ-β follows a geometric series with*

$$\alpha \approx \left((\pi\beta)^{1/\beta}\frac{\beta}{2\pi e}\right)^{-1/(\beta-1)}.$$

Proof. Let Λ be a lattice of rank n and let $\mathbf{b}_1, \ldots, \mathbf{b}_n$ be a BKZ-β reduced basis of Λ satisfying the GSA. Let $\Lambda_i^\perp = \mathrm{span}_{\mathbb{Z}}(\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_n))$ be a projective sublattice of rank $n-i+1$. By Remark 2 applied to $\Lambda_i^\perp$ and by direct computation, we have

$$\begin{aligned}
\mathrm{vol}(\Lambda_i^\perp) &= \prod_{j=i}^{n} \|\mathbf{b}_j^*\| \quad \text{by Remark 2 applied to } \pi_i(\mathbf{B}_{[i:n]}) \\
&= \prod_{j=i}^{n} \left(\alpha^{j-1}\cdot\|\mathbf{b}_1\|\right) \quad \text{by the GSA} \\
&= \alpha^{\sum_{j=i}^{n}(j-1)}\|\mathbf{b}_1\|^{n-i+1} \\
&= \alpha^{\frac{n(n-1)-(i-1)(i-2)}{2}}\|\mathbf{b}_1\|^{n-i+1}.
\end{aligned}$$

We can then compute the ratio

$$\frac{\|\mathbf{b}_i^*\|}{\mathrm{vol}(\Lambda_i^\perp)^{1/(n-i+1)}} = \frac{\alpha^{i-1}\|\mathbf{b}_1\|}{\alpha^{\frac{n(n-1)-(i-1)(i-2)}{2(n-i+1)}}\|\mathbf{b}_1\|} = \alpha^{i-1-\frac{n(n-1)-(i-1)(i-2)}{2(n-i+1)}}. \tag{1.2}$$

By Definition 19, at index $i = n-\beta+1$ a BKZ-β reduced basis has

$$\|\mathbf{b}_{n-\beta+1}^*\| = \lambda_1\left(\Lambda_{n-\beta+1}^\perp\right).$$

Using the Gaussian heuristic for $\Lambda_{n-\beta+1}^\perp$ (which has rank β) we can estimate

$$\|\mathbf{b}_{n-\beta+1}^*\| \approx (\pi\beta)^{\frac{1}{2\beta}}\sqrt{\frac{\beta}{2\pi e}}\cdot\mathrm{vol}(\Lambda_{n-\beta+1}^\perp)^{1/\beta}.$$

Together with Equation (1.2) evaluated at $i = n-\beta+1$, this gives us

$$(\pi\beta)^{\frac{1}{2\beta}}\sqrt{\frac{\beta}{2\pi e}} \approx \frac{\|\mathbf{b}_{n-\beta+1}^*\|}{\mathrm{vol}(\Lambda_{n-\beta+1}^\perp)^{1/\beta}} = \alpha^{n-\beta-\frac{n(n-1)-(n-\beta)(n-\beta-1)}{2\beta}} = \alpha^{-\frac{\beta-1}{2}}.$$

Taking the $-\frac{2}{\beta-1}$-th power we obtain the desired result. □

Remark 10. *It should be noted that while in the proof of Lemma 9 we focused on index $i = n-\beta+1$, the condition $\|\mathbf{b}_i^*\| = \lambda_1\left(\Lambda_i^\perp\right)$ applies for any $i \geq n-\beta+1$.*

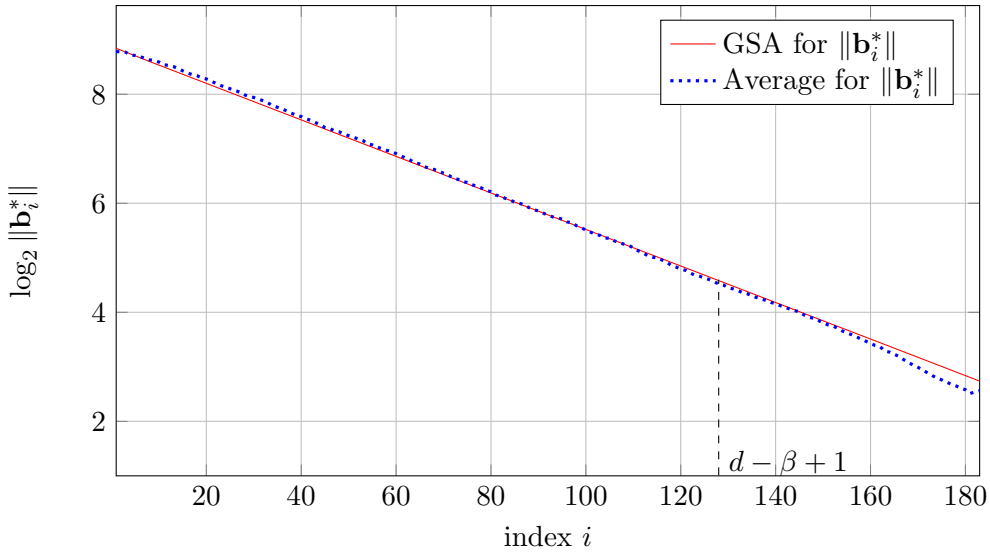


Figure 1.1: Comparison of a GSA prediction for the profile of a BKZ-β-reduced basis for a q-ary lattice of dimension 183 and volume q^{117} with $q = 521$, to the profile output by BKZ-β averaged over 16 bases. The block size $\beta = 56$.

However, the basis profile of a BKZ-β reduced basis stops closely following the GSA on the last β indices, as can be seen in Figure 1.1. This means that to determine α the most appropriate index to use is $i = n - \beta + 1$.

Remark 11. *Combining Lemma 7 with Lemma 9, we can obtain the Hermite factor for BKZ-β under the GSA as $\zeta = \left((\pi\beta)^{1/\beta}\frac{\beta}{2\pi e}\right)^{\frac{n-1}{2(\beta-1)}}$, which is compatible with Chen's limit in (1.1). Defining the root-Hermite factor δ as ζ^{n-1} leaves $\delta = \left((\pi\beta)^{1/\beta}\frac{\beta}{2\pi e}\right)^{\frac{1}{2(\beta-1)}}$ under the GSA.*

We can see an example of how the GSA heuristic compares to bases profiles output by BKZ in Figure 1.1. While the GSA provides a good approximation of the profile of a reduced lattice basis, it does not fully capture the shape of the initial and final indices of the basis. In the case of BKZ-reduced bases, simulator algorithms [CN11, BSW18, LN20] have been designed, that can generate the expected basis profile output by BKZ given an input basis profile, without actually performing lattice reduction. We will describe these algorithms in further detail in Chapter 4, where we will make use of them to estimate the probability that BKZ solves an instance of the uSVP problem, given a certain block size.

1.4.2 Learning With Errors

In 2005, Regev [Reg05] formalised a computational problem that has since become one of the cornerstones of lattice-based cryptography, the Learning With Errors (LWE) problem. Learning With Errors has since been used to construct provably secure public key encryption [Reg05, LP11, ADPS16], oblivious transfer [PVW08], fully homomorphic encryption [BV11, GSW13], identity-based encryption [GPV08], attribute-based encryption [GVW13, BGG+14], digital signatures [BG14a] and obfuscation of some families of circuits [BVWW16] amongst others. Learning With Errors is conjectured to be average-case hard, with a quantum reduction from worst-case SIVP [Reg05], meaning that an algorithm solving LWE implies an efficient quantum algorithm solving SIVP, and a classical reduction from worst-case GapSVP [Pei09, BLP+13]. We now provide definitions and observations on LWE and its variants, to which we will refer to in Chapters 3, 4 and 5.

Definition 23 (Learning With Errors (LWE) [Reg05, Reg09])**.** *Let n, q be positive integers, χ be a probability distribution on $\mathbb{Z}$ and $\mathbf{s}$ be a secret vector in $\mathbb{Z}_q^n$. We denote the LWE distribution $L_{\mathbf{s},\chi,q}$ as the distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ given by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \in \mathbb{Z}$ according to χ and considering it as an element of $\mathbb{Z}_q$, and outputting $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.* Decision-LWE *is the problem of distinguishing whether samples $\{(\mathbf{a}_i, b_i)\}_{i=1}^m$ are drawn from the LWE distribution $L_{\mathbf{s},\chi,q}$ or uniformly from $\mathbb{Z}_q^n \times \mathbb{Z}_q$.* Search-LWE *is the problem of recovering the vector $\mathbf{s}$ from a collection $\{(\mathbf{a}_i, b_i)\}_{i=1}^m$ of samples drawn according to $L_{\mathbf{s},\chi,q}$.*

A useful property of LWE is the polynomial-time equivalence of its decision and search formulations [Reg05]. The distribution χ from which the error is drawn tends to encode some notion of *smallness*, which is usually required for functionality. As originally defined in [Reg05], the LWE secret vector is sampled uniformly from $\mathbb{Z}_q^n$. A standard transformation [MR09, ACPS09] maps m samples from an LWE distribution $L_{\mathbf{s},\chi,q}$ with $\mathbf{s} \leftarrow U(\mathbb{Z}_q^n)$ to $m - n$ samples from an LWE distribution $L_{\mathbf{s}',\chi,q}$ where the secret vector $\mathbf{s}'$ is sampled coefficient-wise from χ. Such a distribution is said to be in *normal form*. In general, more efficient key exchange can be built from LWE distributions where

the secret is sampled from a narrow distribution such as χ (*small secret* LWE) or from a narrow distribution also imposing or implying few non zero entries in $\mathbf{s}$ (*sparse secret* LWE).

Given a fixed number m of LWE samples $\{(\mathbf{a}_i, b_i) \leftarrow L_{\mathbf{s},\chi,q}\}_{i=1}^{m}$, these can be written in matrix form as $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$, where the i-th row of $\mathbf{A}$ is set to $\mathbf{a}_i$, and the i-th component of $\mathbf{b}$ is b_i. Then $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q$, where the i-th component of $\mathbf{e}$ is the error term of the i-th sample from $L_{n,\chi,q}$. Note that with high probability[5] any n samples $(\mathbf{A}, \mathbf{b})$ from an LWE distribution with prime modulus q, $\mathbf{s} \leftarrow \chi_s^n$ and $\mathbf{e} \leftarrow \chi_e^n$ can be turned into n LWE samples $(\mathbf{A}^{-1}, \mathbf{A}^{-1}\mathbf{b})$ where the roles of χ_e and χ_s are swapped. This can be useful for creating embedding lattices when using $m \leq n$ samples during attacks is optimal and n samples are available (to allow inversion of $\mathbf{A}$).

1.4.2.1 Variants

Since LWE leads to public-key sizes at least quadratic in the security parameter, many schemes are based on its ring variant, called Ring-LWE [LPR10] or "RLWE" in short. Below, we give the definition of Polynomial-LWE [SSTX09] (or "PLWE") which is equivalent to the RLWE definition for power-of-two cyclotomic rings. For e.g. prime cyclotomic ring these two definitions are not equivalent, i.e. the geometry of the error polynomial distribution changes somewhat between the coefficient and canonical embeddings [LPR10]. However, as is common in the literature, we will abuse notation and refer to PLWE as RLWE.

Definition 24 (RLWE [SSTX09, LPR10])**.** *Let q be a positive integer, R be a polynomial ring, χ be a probability distribution on $\mathbb{Z}$ and s be a secret polynomial in R_q. We denote by $L_{s,\chi}$ the probability distribution on $R_q \times R_q$ obtained by choosing $a \in R_q$ uniformly at random, choosing $e \in R$ by sampling each of its coefficients according to χ and considering it in R_q, and returning $(a, b) = (a, a \cdot s + e) \in R_q \times R_q$.*
Decision-RLWE *is the problem of deciding whether pairs $(a_i, b_i) \in R_q \times R_q$ are sampled according to $L_{s,\chi}$ or the uniform distribution on $R_q \times R_q$.*

[5] That is, the probability that a uniformly sampled matrix in $\mathbb{Z}_q^{n \times n}$ is invertible.

Search-RLWE *is the problem of recovering* s *from pairs* $(a_i, b_i) = (a_i, a_i \cdot s + e_i) \in R_q \times R_q$ *sampled according to* $L_{s,\chi}$.

As in the case of LWE, decision and search variants of RLWE are polynomial-time equivalent (for cyclotomic rings R) [LPR10].[6] The increased efficiency of RLWE compared to LWE is achieved by adding algebraic structure. Informally, for a polynomial ring $R = \mathbb{Z}[x]/(f)$ with f of degree n, each RLWE sample can be viewed as n correlated LWE samples. While, so far, no cryptanalytic algorithm is known which exploits this additional structure for appropriate rings [Pei16], some designs hedge against such hypothetical attacks by considering problems which require the attacker to find short vectors in a lattice of larger module rank [SAB+17, DKRV17]. In particular, Module-LWE (or "MLWE") interpolates between the plain and the ring variants of LWE.

Definition 25 (MLWE [LS15])**.** *Let* q, k *be positive integers,* R *be a polynomial ring,* χ *be a probability distribution on* $\mathbb{Z}$ *and* $\mathbf{s}$ *be a secret module element in* R_q^k*. We denote by* $L_{\mathbf{s},\chi}$ *the probability distribution on* $R_q^k \times R_q$ *obtained by choosing* $\mathbf{a} \in R_q^k$ *uniformly at random, choosing* $e \in R$ *by sampling each of its coefficients according to* χ *and considering it in* R_q*, and returning* $(\mathbf{a}, b) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in R_q^k \times R_q$.
Decision-MLWE *is the problem of deciding whether pairs* $(\mathbf{a}_i, b_i) \in R_q^k \times R_q$ *are sampled according to* $L_{\mathbf{s},\chi}$ *or the uniform distribution on* $R_q^k \times R_q$.
Search-MLWE *is the problem of recovering* $\mathbf{s}$ *from pairs* $(\mathbf{a}_i, b_i) = (\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i) \in R_q^k \times R_q$ *sampled according to* $L_{\mathbf{s},\chi}$.

Again, the search and the decision variants of this problem are polynomial-time equivalent [LS15, Thm. 4.7].

One can view RLWE and MLWE instances as LWE instances by interpreting the "e, s and b" elements in $\mathcal{R}_q$ as their coefficient vectors in $\mathbb{Z}_q^n$ and the "a" elements in $\mathcal{R}_q$ as structured matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ where the i-th column of $\mathbf{A}$ is the coefficient vector of $a \cdot x^i \bmod f$ (such that the polynomial product $a \cdot s$ can be computed as a matrix-vector product $\mathbf{As}$), ignoring their algebraic

[6] Equivalence is true for any cyclotomic ring R when using the canonical embedding, or for the ring $R = \mathbb{Z}[x]/(x^n + 1)$ for n a power of 2 when using the coefficient embedding.

structure. This identification with LWE is the standard approach to costing the complexity of solving RLWE and MLWE due to the absence[7] of known cryptanalytic techniques exploiting the algebraic structure for appropriate cyclotomic rings (such as $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ for n a power of 2) [Pei16].

1.4.3 Solving LWE

A few approaches exist for solving the LWE problem, given access to an LWE oracle returning samples $(\mathbf{a}_i, b_i) \leftarrow L_{\mathbf{s},\chi,q}$. We will briefly describe these here. Throughout, we assume that we are given access to m LWE samples in matrix form $(\mathbf{A}, \mathbf{b})$, such that $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q$, where the i-th row of $\mathbf{A}$ is $\mathbf{a}_i$. It should be noted that all these attacks use classical algorithms. Quantum attacks can be derived essentially by speeding up any search subroutines using Grover's algorithm [Gro96].

The asymptotically cheapest attack on LWE is an algebraic attack by Arora and Ge [AG11]. It uses linearisation to solve the set of equations

$$\left\{ \prod_{\eta \in \mathrm{Supp}(\chi)} (\mathbf{b}_i - \langle \mathbf{a}_i,\ \mathbf{s} \rangle - \eta) = 0 \right\}_i$$

over $\mathbb{Z}_q$ with $\mathbf{s}$ unknown, where $\mathrm{Supp}(\chi)$ is the support of the error distribution χ, and hence solve Search-LWE. While asymptotically the best approach whenever χ has width $O(\sqrt{n})$, for practical parameters it is slower than other methods.

Another method going back to the Learning Parity with Noise (LPN) literature is the BKW algorithm [BKW00]. This attack uses combinatorial techniques to distinguish the distribution of the b_i from uniform over $\mathbb{Z}_q$, in other words solving Decision-LWE. Notably, BKW variants [KF15, GJS15] result in a subexponential-time algorithm against LWE with binary secret. However, BKW requires access to a number of samples m larger than usually available

[7] In recent years a rich literature on the Ideal-SVP problem has developed [CDW17, DPW19, PHS19]. However, while Ideal-SVP is related to RLWE, it is not known whether techniques for solving the first could be adapted to the RLWE or MLWE settings.

in the cryptographic setting[8], and often results in higher expected costs than for lattice reduction attacks.

Finally, the most efficient attacks on LWE are lattice reduction attacks. Using the notation of [MR09], in its basic variant the *primal attack* uses lattice reduction to solve Bounded Distance Decoding with respect to $\mathbf{b}$ in the *primal lattice* of $\mathbf{A}$, that is $\Lambda_q(\mathbf{A}) \coloneqq \{\mathbf{y} \in \mathbb{Z}^m \colon \mathbf{y} = \mathbf{A}\mathbf{x} \bmod q \text{ for } \mathbf{x} \in \mathbb{Z}^n\}$. This results in recovering the vector $\mathbf{v} = \mathbf{A}\mathbf{s} \bmod q$, and hence $\mathbf{e} = \mathbf{b} - \mathbf{v} \bmod q$, solving Search-LWE. Also in its basic variant, the *dual attack* uses lattice reduction to find a short vector in the *dual lattice*[9] of $\mathbf{A}$, that is $\Lambda_q^\perp(\mathbf{A}) \coloneqq \{\mathbf{y} \in \mathbb{Z}^m \colon \mathbf{y}^t\mathbf{A} = \mathbf{0} \bmod q\}$, hence solving SIS. Such a vector can be then multiplied with $\mathbf{b}$ resulting in $c = \langle \mathbf{y}, \mathbf{b} \rangle \bmod q$. If $\mathbf{b} \leftarrow U(\mathbb{Z}_q^m)$, c will have a relatively large absolute value with respect to q. If $(\mathbf{A}, \mathbf{b}) \leftarrow L_{\mathbf{s},\chi,q}$, then $c = \langle \mathbf{y}, \mathbf{b} \rangle = \mathbf{y}^t\mathbf{A}\mathbf{s} + \mathbf{y}^t\mathbf{e} = \langle \mathbf{y}, \mathbf{e} \rangle \bmod q$ will be significantly smaller than q when considering its residue class in $[-\frac{q}{2}, \frac{q}{2})$. Observing the size of c solves Decision-LWE. Lattice reduction attacks can also be combined with combinatorial strategies that reduce the dimensionality of the problem, resulting in *hybrid* attacks [How07].

Lattice reduction attacks are practically the cheapest attacks on LWE. Given a basis for one of the lattices described above, estimating the cost of the corresponding attack when using block reduction algorithms such as BKZ, requires us to find the *block size* necessary to sufficiently reduce the basis. In the case of the dual attack, the state of the art analysis for doing this estimation can be found in [Alb17]. Chapters 3 and 4 will be dedicated to explaining in detail how this is done for the primal attack.

1.5 Quantum computation

In this section, we provide some fundamental concepts and notation for describing quantum algorithms. For a concise introduction, see Chapters 1 to

[8] Although these could possibly be available in a fully homomorphic encryption-with-bootstrapping setting.

[9] In this paragraph we don't mean $\Lambda_q^\perp$ as a projective sublattice.

4 of [KLM+07]. For a more comprehensive discussion, see [NC10]. In Chapter 2, we will use these notions to investigate trade-offs in the use of the quantum algorithm for unstructured search by Grover [Gro96] in the setting of non-asymptotic cryptanalysis of block ciphers.

Classical algorithms can be implemented using digital circuits. These encode information as strings of bits valued in $\{0, 1\}^n$, and perform Boolean logic by applying logic gates. Analogously to their classical counterpart, quantum algorithms can be thought of in terms of quantum circuits. These encode information in qubit registers valued in $(\mathbb{C}^2)^{\otimes n}$, and perform operations on these by applying quantum gates.[10] We will now describe the basic components of quantum circuits.

Qubits. The fundamental unit of quantum information is the qubit. These are unit vectors in $\mathbb{C}^2$ (considered with complex inner product), and their state is usually denoted using bra-ket notation as $|b\rangle$, with the state's adjoint $|b\rangle^\dagger$ also written as $\langle b|$, such that $\||b\|^2 = \langle b|b\rangle = 1$. Qubits can be grouped into registers by means of tensor products. For example, the state of a register with n qubits independently set to states $b_1, \ldots, b_n$ is $|b_1\rangle \otimes \cdots \otimes |b_n\rangle$, or using more compact notation, $|b_1\rangle \cdots |b_n\rangle$ or $|b_1 \ldots b_n\rangle$. Generally speaking, the state of an n-qubit register has value in the Hilbert space $\mathcal{H} = (\mathbb{C}^2)^{\otimes n}$. As a complex vector space, $\mathbb{C}^2$ can be given an explicit orthonormal basis $\{|0\rangle, |1\rangle\}$, where the labels 0 and 1 are used in analogy to classical bits. Then the state of any qubit can be expressed as a linear combination $|b\rangle = \alpha|0\rangle + \beta|1\rangle$, with $\alpha, \beta \in \mathbb{C}$ such that $|\alpha|^2 + |\beta|^2 = 1$. This expression is unique up to multiplication by a global phase $e^{i\theta}$ with $\theta \in \mathbb{R}$. A basis for $\mathcal{H}$ can be constructed by tensoring the basis of $\mathbb{C}^2$ with itself, resulting in the *computational basis* of $\mathcal{H}$, $\{|b_1 \ldots b_n\rangle : b_1 \ldots b_n \in \{0,1\}^n\}$. A register of n qubits will have a state $|\psi\rangle \in \mathcal{H}$ described by a linear combination of the computational basis of $\mathcal{H}$. If $|\psi\rangle$ can be written as a tensor product of qubits, e.g. $|\psi\rangle = |00\rangle$ where $n = 2$, we say it is in a *separable* state. Otherwise, if $|\psi\rangle$ can't be expressed as a tensor product, e.g. $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, we say it is *entangled*.

[10] It should be noted that both classical and quantum circuits can also be built in terms of d-ary logic, where information is stored in terms of digit strings in $[d]^n$ or qu$d$it registers in $(\mathbb{C}^d)^{\otimes n}$. Without loss of generality, in this thesis we assume binary logic.

Gates. Operations on n qubits are expressed in the form of unitary operators acting on $\mathcal{H}$. Fixing the computational basis for $\mathcal{H}$, these operators can be described as matrices $U \in \mathbb{C}^{2^n \times 2^n}$ such that $U^\dagger U = UU^\dagger = I$, where $U^\dagger$ indicates the adjoint of U. Quantum gates are reversible, and for every quantum gate U, its inverse gate is described as $U^\dagger$. This also means that in theory no information is lost by the application of a quantum gate[11]. Applying a gate with operator U on a register in state $|\psi\rangle$ results in the register being put in state $U|\psi\rangle$. The identity matrix can be seen as the no-op operator, which leaves the value in a register intact. Similarly to quantum states, operators can themselves be tensored, to construct operators acting on larger registers in parallel (an example will be given in the paragraph on circuits, below). In the case of classical circuits, a universal set of gates is a set of logic gates such that for every truth table a circuit can be obtained using only the gates in the set. For example, the sets $\{\text{AND}, \text{NOT}\}$ and $\{\text{NAND}\}$ are two universal sets. In the case of quantum circuits, a universal set of gates is a set of unitary operators such that every unitary operator can be approximated to arbitrary precision using only the gates in the set. One such set is the "Clifford+T" set of gates, that is the set comprised of 1-qubit Clifford gates (including NOT, with symbol $\oplus$), controlled-NOT (or CNOT, with symbol ), and T, the 1-qubit gate with $\exp(\pm i\pi/8)$ on the diagonal and 0 otherwise (up to a global phase) in the computational basis. One quantum gate of particular interest is the Toffoli gate, also known as the controlled-controlled-NOT gate (CCNOT). Classically, Toffoli maps bits (b_1, b_2, b_3) to $(b_1, b_2, b_3 \oplus b_1 \cdot b_2)$. By fixing $b_3 = 1$, this results in the NAND gate, which is classically universal. This means that any classical truth table $f\colon \{0,1\}^n \to \{0,1\}^m$ can be implemented as a quantum circuit by first transforming it into the reversible function $\hat{f}\colon (x, y) \mapsto (x,\, y \oplus f(x))$ (which is self-inverse), and then implementing $\hat{f}$ using Toffoli gates.

Measurements. As we mentioned above, the state $|\psi\rangle$ of an n-qubit register can be expressed as a linear combination of the computational basis of $\mathcal{H}$, $|\psi\rangle = \sum_{i\in\{0,1\}^n} \alpha_i |i\rangle$ with $\langle\psi|\psi\rangle = \sum_{i\in\{0,1\}^n} |\alpha_i|^2 = 1$. Whenever $\alpha_i \neq 0$ for more than one index i, we say that $|\psi\rangle$ is in *superposition*. As part of quantum computation, a state can be operated on in two ways. One is via quantum gates,

[11] In practice, quantum circuits will not be closed systems, hence this modelling won't exactly apply.

and the other is via state measurement. While applying a quantum operator U on a state $|\psi\rangle$ will result in a possibly superposed state $U|\psi\rangle$, measuring $|\psi\rangle$ will result in a non-superposed state from the computational basis $|i\rangle$, for some $i \in \{0,1\}^n$. In particular, it will result in $|i\rangle$ with probability $|\alpha_i|^2$. This operation allows one to write non-superposed states to a classical bit string of length n by measuring them, but will result in a loss of information whenever $|\psi\rangle$ was in a superposed state. Measurement of part of an entangled register will affect the other components in the register, as to satisfy the output of the measurement. For example, if measuring the first qubit of a register in state $|\psi\rangle = (|00\rangle + |01\rangle + |11\rangle)/\sqrt{3}$ returned a value of $|0\rangle$, after the measurement the register would be in the state $|\psi'\rangle = (|00\rangle + |01\rangle)/\sqrt{2}$, reducing the probability of measuring $|1\rangle$ on the *second* qubit from 2/3 to 1/2. On the other hand, if measuring the first qubit returned a value of $|1\rangle$, this would leave the register in the state $|\psi'\rangle = |11\rangle$, causing measurements of the second qubit to return $|1\rangle$ with probability 1.

Circuits. Having introduced quantum registers, gates and measurements, we can now describe a circuit. A quantum circuit consists of a register of n qubits, or *wires*, usually assumed to be initially set to the state $|0^n\rangle$, and a sequence of gates and measurements performed on the wires in a given order. We can use different metrics to describe the size of the circuit. The circuit *width* is the number of wires used, the *gate count* is the number of gates used (this may also include the number of measurements performed on single wires), the *depth* is the maximum length of any path from an input state to an output state when interpreting the circuit as an undirected graph with gates as its nodes and wires as its edges. Figure 1.2 depicts a small toy circuit comprised of three qubits on which three gates are applied (a T gate, a NOT gate, and a controlled-NOT gate) with input state $|\psi\rangle$ and time flowing from left to right. The circuit has width 3, gate count 3, and depth 2. It maps the input state $|\psi\rangle$ to $U|\psi\rangle$ where $U = (U_{\text{CNOT}} \otimes I_2) \cdot (U_{\text{T}} \otimes I_2 \otimes U_{\text{X}})$, where U_{CNOT}, U_{T} and U_{X} are the unitary matrices for the CNOT, T, and X (also denoted as NOT or $\oplus$) gates respectively. It should be noted that we will be always working on theoretical circuits, without specific constraints on what gates can be applied on which wires. In practice, while the small circuit in Figure 1.2 is described

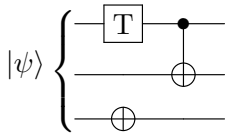


Figure 1.2: Example of a small quantum circuit.

over three "logical" qubits, likely more qubits would be necessary to implement a noiseless version of it in the real world.

Copying information. Due to the no-cloning theorem [WZ82, Die82], given a register in state $|\psi\rangle$, it is not possible to copy its state and obtain a second register in state $|\psi\rangle$ un-entangled from the first. That is, given a superposed state $|\psi\rangle$, there is no unitary transformation U such that $U|\psi\rangle|0\rangle = |\psi\rangle|\psi\rangle$. Rather, if two separable registers in the same state are required, these should be independently prepared, for example by running the same circuit twice on two different quantum registers. It is however possible to copy classical information using the CNOT gate, which maps $|x\rangle|y\rangle \mapsto |x\rangle|y \oplus x\rangle$ where $x, y \in \{0, 1\}$. A simple computation using the matrix form of the CNOT gate U_{CNOT} on a superposed qubit state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ demonstrates the difference between using CNOT and cloning a state:

$$
\begin{aligned}
|\phi\rangle &= U_{\text{CNOT}} \, |\psi\rangle \otimes |0\rangle \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} \alpha \\ 0 \\ \beta \\ 0 \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \\ 0 \\ \beta \end{pmatrix} \\
&= \alpha|00\rangle + \beta|11\rangle \\
&\neq |\psi\rangle \otimes |\psi\rangle \\
&= \alpha^2|00\rangle + \alpha\beta|01\rangle + \alpha\beta|10\rangle + \beta^2|11\rangle .
\end{aligned}
$$

While measuring the first register of $|\psi\rangle|\psi\rangle$ would not affect the second register, measuring the first register of $U_{\text{CNOT}}|\psi\rangle|0\rangle = \alpha|00\rangle + \beta|11\rangle$ will collapse the state in the second register.

1.5.1 Grover's algorithm

We have described the basic components used to implement quantum algorithms as circuits. We now move our focus to a milestone algorithm in quantum computing, Grover's algorithm [Gro96].

Definition 26 ((N, M)-unstructured search problem)**.** *Given a randomly sorted list L of size N and a property P such that exactly M elements of L satisfy P, find one such element.*

Classically, the best algorithm for solving the $(N, 1)$-unstructured search problem has complexity $O(N)$. Intuitively, this is because the only possible approach is to check every element in L until we find the one satisfying P, since the list is not sorted in any useful way. This will have an average runtime of $\frac{N+1}{2}$ checks, and a worst-case runtime of $N-1$ checks. In [Gro96], Grover proposed a solution for this problem, using quantum computation. He describes a quantum algorithm for searching such an unstructured list with complexity $O(\sqrt{N})$. Grover's algorithm is optimal in the sense that any quantum search algorithm needs at least $\Omega(\sqrt{N})$ oracle queries to solve the problem [BBHT98]. In [Zal99], Zalka shows that for any number of oracle queries, Grover's algorithm gives the largest probability to find a solution, meaning that it is the exactly optimal quantum search algorithm (and not just asymptotically). We now provide a description of the algorithm and its runtime. In Chapter 2, we will look in detail at the complexity of Grover's algorithm from a non-asymptotic point of view, when used to perform cryptanalysis of block ciphers.

Grover's circuit components. For simplicity, we restrict to $N = 2^k$ and label the elements of L by their indices in $\{0,1\}^k$. Let $W \subset L$ be the set of solutions to the unstructured search problem, with $M = \#W$ (we call this the M-solution version of Grover's algorithm). Let $|\psi\rangle = \sum_{i \in L} |i\rangle / \sqrt{N}$ be the fully entangled state[12] where the $|i\rangle$ form the computational basis for $(\mathbb{C}^2)^{\otimes k}$. We define the superposed state with solutions to the search problem

[12] $|\psi\rangle$ can be cheaply constructed as $|\psi\rangle = H^{\otimes k} |0\rangle^{\otimes k}$.

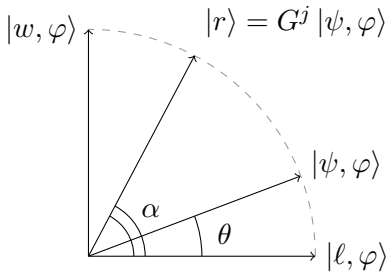


Figure 1.3: The $|w,\varphi\rangle$—$|\ell,\varphi\rangle$ plane P.

$|w\rangle \coloneqq \sum_{i\in W} |i\rangle / \sqrt{M}$ and the state corresponding to the complement $L\backslash W$, $|\ell\rangle \coloneqq \sum_{i\in L\backslash W} |i\rangle / \sqrt{N-M}$.

Given a Boolean function $f : \{0,1\}^k \to \{0,1\}$ that marks solutions to the search problem, i.e. $f(x) = 1$ if and only if $x \in W$, Grover's algorithm makes use of an operator U_f implementing f, such that U_f maps $|x\rangle\,|y\rangle \mapsto |x\rangle\,|y \oplus f(x)\rangle$. When $|y\rangle$ is in the state $|\varphi\rangle \coloneqq (|0\rangle - |1\rangle)/\sqrt{2}$, then this action can be written as $|x\rangle\,|\varphi\rangle \mapsto (-1)^{f(x)}\,|x\rangle\,|\varphi\rangle$. This means that the oracle applies a phase shift to exactly the solutions' indices. We call U_f the "Grover oracle" for f. We also define operators $U_\psi = (2\,|\psi\rangle\langle\psi| - I_N) \otimes I_2$ and $G = U_\psi U_f$. Grover's algorithm amounts to applying j times the operator G on a quantum register $|r\rangle$ initially set to $|r\rangle \leftarrow |\psi\rangle\,|\varphi\rangle$. The objective is to eventually attain $|r\rangle \approx |w\rangle\,|\varphi\rangle$, so that measuring the first k qubits of $|r\rangle$ results in a solution to the search problem $x \in W$ with high probability.

The states $|w\rangle$ and $|\ell\rangle$ are orthogonal, $\langle w|\ell\rangle = 0$, since $W \cap L\backslash W = \varnothing$, and have unit norm. To derive the runtime of Grover's algorithm, we will be working on the real plane P spanned by $|w,\varphi\rangle = |w\rangle\,|\varphi\rangle$ and $|\ell,\varphi\rangle = |\ell\rangle\,|\varphi\rangle$ when they are identified with the canonical basis of $\mathbb{R}^2$, depicted in Figure 1.3. We start by decomposing the initial state $|\psi,\varphi\rangle = |\psi\rangle\,|\varphi\rangle$ as

$$|\psi,\varphi\rangle = \sqrt{\frac{M}{N}}\,|w,\varphi\rangle + \sqrt{\frac{N-M}{N}}\,|\ell,\varphi\rangle\,.$$

By direct calculation, $|\psi,\varphi\rangle$ and $|\ell,\varphi\rangle$ form an angle

$$\theta = \arccos\left(\langle \ell,\varphi|\psi,\varphi\rangle\right) = \arccos\left(\sqrt{\frac{N-M}{N}}\right).$$

Applying G. As discussed above, the application of U_f maps $|x\rangle|\varphi\rangle \mapsto (-1)^{f(x)}|x\rangle|\varphi\rangle = (-1)^{[\![x\in W]\!]}|x\rangle|\varphi\rangle$. When looking in particular at the basis of the $|w,\varphi\rangle$—$|\ell,\varphi\rangle$ plane, we see that

$$U_f|\ell,\varphi\rangle = |\ell,\varphi\rangle \quad \text{and} \quad U_f|w,\varphi\rangle = -|w,\varphi\rangle\,.$$

This means that given $|w\rangle$ and restricting our attention to the $|w,\varphi\rangle$—$|\ell,\varphi\rangle$ plane, we can write $U_f\big|_P = (I_N - 2|w\rangle\langle w|) \otimes I_2$.

By writing $|\psi\rangle$ in terms of $|w\rangle$ and $|\ell\rangle$ we can directly compute G's action on the $|w,\varphi\rangle$—$|\ell,\varphi\rangle$ plane as

$$\begin{aligned} G\big|_P = U_\psi U_f\big|_P &= (2|\psi\rangle\langle\psi| - I_N)(I_N - 2|w\rangle\langle w|) \otimes I_2 \\ &= \left[\left(1-2\frac{M}{N}\right)|w\rangle\langle w| + 2\frac{\sqrt{N-M}\sqrt{M}}{N}|w\rangle\langle\ell| \right. \\ &\qquad \left. -2\frac{\sqrt{N-M}\sqrt{M}}{N}|\ell\rangle\langle w| + \left(1-2\frac{M}{N}\right)|\ell\rangle\langle\ell|\right] \otimes I_2. \end{aligned}$$

By inspection, we can notice that given θ the angle between $|\psi,\varphi\rangle$ and $|\ell,\varphi\rangle$, $G\big|_P$ corresponds to the rotation matrix

$$\begin{aligned} G\big|_P = \big[&\cos 2\theta\,|w\rangle\langle w| + \sin 2\theta\,|w\rangle\langle\ell| \\ &- \sin 2\theta\,|\ell\rangle\langle w| + \cos 2\theta\,|\ell\rangle\langle\ell|\,\big] \otimes I_2, \end{aligned}$$

which rotates vectors on the $|w\rangle|\varphi\rangle$—$|\ell\rangle|\varphi\rangle$ plane by an angle 2θ. This means that the registry $|r\rangle$ will always lie on P, since it is originally set to $|\psi,\varphi\rangle$.

Runtime of Grover's algorithm. Given the geometric interpretation above, after j applications of G, the register state $|r\rangle \leftarrow G^j|\psi,\varphi\rangle$ will form an angle $\alpha = (2j+1)\,\theta$ with $|\ell,\varphi\rangle$. Therefore, when measuring the first k qubits after $j > 0$ iterations of G, the success probability $p(j)$ for obtaining one of the solutions is $p(j) = \sin^2((2j+1)\theta)$ [BBHT98], the squared component of $|r\rangle$ along the $|w,\varphi\rangle$ direction. Since we aim to measure the state when $|r\rangle \approx |w,\varphi\rangle$, the optimal number of iterations should be j such that $\alpha \approx \frac{\pi}{2} \Leftrightarrow j \approx \frac{\pi}{4\theta} - \frac{1}{2}$, which corresponds to $p(j) \approx 1$. Note that whenever $M \ll N$, $\sin\theta$ will be small, and $\theta \approx \sin\theta = \sqrt{M/N}$. This allows us to replace θ in the expression for j above, and deduce that after $j = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rfloor$ iterations, measurement yields

a solution to the search problem with overwhelming probability of at least $1 - \frac{M}{N}$ [BBHT98]. This results in the asymptotic cost $O(\sqrt{N})$, where the O essentially hides the fixed cost of each application of G.

Chapter 2

Quantum Key Search Under a Depth Restriction

Contents

Grover's search algorithm gives a quantum attack against block ciphers by searching for a key that matches a small number of plaintext-ciphertext pairs. This attack uses $O(\sqrt{N})$ calls to the cipher to search a key space of size N. Previous work in the specific case of AES derived the full gate cost by analysing quantum circuits for the cipher, but focused on minimizing the number of qubits.

In this chapter, we study the cost of quantum key search attacks under a depth restriction and introduce techniques that reduce the oracle depth, even if at the cost of requiring more qubits. As cases in point, we design quantum circuits for the block ciphers AES and LowMC. Our circuits give a lower overall attack cost in both the gate count and depth-times-width cost models when compared to those implied by circuits in the previous literature. In NIST's post-quantum cryptography standardisation process, security categories are defined based on the concrete cost of quantum key search against AES. We present new, lower cost estimates for each category, so our work has immediate implications for the security assessment of post-quantum cryptography.

As part of the original publication of this chapter, we released Q# implementations of the full Grover oracle for AES-128, -192, -256 and for the three LowMC instantiations used in Picnic, including unit tests and code to reproduce our quantum resource estimates. To the best of our knowledge, these are the first two such full implementations and automatic resource estimations.

2.1 Motivation

The prospect of a large-scale, cryptographically relevant quantum computer has prompted increased scrutiny of the post-quantum security of cryptographic primitives. Shor's algorithm for factoring and computing discrete logarithms introduced in [Sho94] and [Sho97] will completely break public-key schemes

such as RSA, ECDSA and ECDH. But symmetric schemes like block ciphers and hash functions are widely considered post-quantum secure. The only caveat thus far is a security reduction due to key search or pre-image attacks with Grover's algorithm [Gro96]. As Grover's algorithm only provides at most a square root speed-up, the rule of thumb is to simply double the cipher's key size to make it post-quantum secure. Such conventional wisdom reflects the asymptotic behaviour of Grover's algorithm and only gives a rough idea of the security penalties that quantum computers inflict on symmetric primitives. In particular, the cost of evaluating the Grover oracle is often ignored.

In their call for proposals to the standardisation of post-quantum cryptography [Nat16], the National Institute of Standards and Technology (NIST) proposes security categories for post-quantum public-key schemes such as key encapsulation and digital signatures. Categories are defined by the cost of quantum algorithms for exhaustive key search on the block cipher AES and collision search for the hash function SHA-3, and measure the attack cost in the number of quantum gates. Because the total gate count of Grover's algorithm increases with parallelisation, they impose a total upper bound on the depth of a quantum circuit, called `MAXDEPTH`. There is no bound on circuit width. A submitted algorithm meets the requirements of a specific security category if the best known attack uses more resources (gates) than are needed to solve the reference problem. Hence, a concrete and meaningful definition of these security categories depends on precise resource estimation of the size of the Grover oracle used for key search on AES.

Security categories 1, 3 and 5 correspond to key recovery against AES-128, AES-192 and AES-256, respectively. The NIST proposal derives gate cost estimates from the concrete, gate-level descriptions of the AES oracle by Grassl, Langenberg, Roetteler and Steinwandt [GLRS16]. Grassl *et al.* aim to minimize the circuit width, i.e. the number of qubits needed.

Prior work. Since the publication of [GLRS16], other works have studied quantum circuits for AES, the AES Grover oracle and its use in Grover's algo-

rithm[1]. Almazrooie, Samsudin, Abdullah and Mutter [ASAM18] improve the quantum circuit for AES-128. As in [GLRS16], the focus is on minimizing the number of qubits. The improvements are a slight reduction in the total number of Toffoli gates and the number of qubits required, by using a binary field inversion circuit that saves one multiplication. Kim, Han and Jeong [KHJ18] discuss time-space trade-offs for key search on block ciphers in general and use AES as an example. They discuss NIST's `MAXDEPTH` parameter and hence study parallelisation strategies for Grover's algorithm to address the depth constraint. They take the Toffoli gate depth as the relevant metric for the `MAXDEPTH` bound arguing that it is a conservative approximation.

Recently, independent and concurrent to parts of this work, Langenberg, Pham and Steinwandt [LPS20] developed quantum circuits for AES that demonstrate significant improvements over those presented in [GLRS16] and [ASAM18]. The main source of optimization is a different S-box design derived from work by Boyar and Peralta in [BP10] and [BP12], which greatly reduces the number of Toffoli gates in the S-box as well as its Toffoli depth. Another improvement is that fewer auxiliary qubits are required for the AES key expansion. Again, this work aligns with the objectives in [GLRS16] to keep the number of qubits small.

Bonnetain *et al.* [BNS19] study the post-quantum security of AES within a new framework for classical and quantum structured search. The work cites [GLRS16] for deducing concrete gate counts for reduced-round attacks.

Our contributions. We present implementations of the full Grover oracle for key search on AES and LowMC in Q# [SGT+18], including full implementations of the block ciphers themselves. In contrast to previous work [GLRS16, ASAM18, LPS20], having a concrete implementation allows us to get more precise, flexible and automatic estimates of the resources required to compute these operations. It also allows us to unit test our circuits, to make sure that the implementations are correct.

[1] As well as for other symmetric primitives such as SHA2/3 [AMG+16].

All of our code and data can be found at `github.com/microsoft/grover-blocks`. The source code is available under a free license to allow independent verification of our results, further investigation of different trade-offs and cost models and re-costing as the Q# compiler improves and as automatic optimization software becomes available. We hope that it can serve as a useful starting point for cryptanalytic work to assess the post-quantum security of other schemes.

We review the literature on the parallelisation of Grover's algorithm [BBHT98, Zal99, GR04, KHJ18] to explore the cost of attacking AES and LowMC in the presence of a bound on the total depth, such as `MAXDEPTH` proposed by NIST. We conclude that using parallelisation by dividing the search space is advantageous. We also give a rigorous justification for the number of plaintext-ciphertext blocks needed in Grover's oracle in the context of parallelisation. Smaller values than those proposed by Grassl *et al.* [GLRS16] are sufficient, as is also pointed out by Langenberg *et al.* [LPS20].

Our quantum circuit optimization approach differs from those in the previous literature [GLRS16, ASAM18, LPS20] in that our implementations do not aim for the lowest possible number of qubits. Instead, we designed them to minimize the gate-count and depth-times-width cost metrics for quantum circuits under a depth constraint. The gate-count metric is relevant for defining the NIST security categories and the depth-times-width cost metric is a more realistic measure of quantum resources when quantum error correction is deployed. Favouring lower depth at the cost of a slightly larger width in the oracle circuit leads to costs that are smaller in both metrics than for the circuits presented in [GLRS16, ASAM18, LPS20]. Grover's algorithm does not parallelise well, meaning that minimizing depth rather than width is crucial to make the most out of the available depth.

To the best of our knowledge, our work resulted in the most shallow quantum circuit of AES at the time of publication, and the first ever for LowMC. We chose to also implement LowMC as an example of a quantum circuit for another block cipher. It is used in the Picnic signature scheme [CDG+17, ZCD+17], a round-2 candidate in the NIST standardisation process. Thus, our implementation

can contribute to more precise cost estimates for attacks on Picnic and to its post-quantum security assessment.

We present our results for quantum key search on AES in the context of the NIST post-quantum cryptography standardisation process and derive new and lower cost estimates for the definition of the NIST security strength categories. We see a consistent gate cost reduction between 11 and 13 bits, making it easier for submitters to claim a given quantum security category.

Chapter roadmap. In Section 2.2 we will review the basic idea of attacking block ciphers using Grover's algorithm. In Section 2.3 we describe the techniques we will use for quantum circuit design and cost estimation. In Sections 2.4 and 2.5 we will describe our implementations of AES and LowMC, respectively. In Section 2.6 we will derive our cost estimates for key search, compare them to previous work, and discuss their implications. In Section 2.7 we will mention developments in this field since the publication of this chapter as a paper and future research directions.

2.2 Finding a block cipher key with Grover's algorithm

Given plaintext-ciphertext pairs created by encrypting a small number of messages with a block cipher under a common key, Grover's quantum search algorithm [Gro96] can be used to find such key [YI00].

In Section 1.5.1, we have introduced Grover's algorithm. In this section we describe how it can be applied to the key search problem and how it parallelises under depth constraints.

2.2.1 Block ciphers

Block ciphers are a versatile cryptographic primitive used extensively in cryptography. Mathematically, a block cipher can be seen as a family of permutations

$C_{(\cdot)}$ over a finite set $\mathcal{M}$, indexed by elements K of a finite set $\mathcal{K}$, called *keys*. Usually, we let $\mathcal{M} = \{0,1\}^n$ and $\mathcal{K} = \{0,1\}^k$, and say that $C_{(\cdot)}$ has *key size* or *key length* k and *block size* n. Given a key $K \in \mathcal{K}$, the function $C_K \colon \mathcal{M} \to \mathcal{M}$ is a permutation. Ideally, block ciphers are designed such that any C_K is indistinguishable from a random permutation over $\mathcal{M}$.

Block ciphers can be used to build, among others, stream ciphers [KL14, §3.5.1], cryptographic hash functions [MVO96, §9.4.1], message authentication codes [MVO96, §9.5.1] and pseudo-random number generators [DE05, §6.2.1], via various modes of operation.

2.2.2 Key search for a block cipher

Let $C_{(\cdot)}$ be a block cipher with block size n and key length k. For a key $K \in \{0,1\}^k$ denote by $C_K(m) \in \{0,1\}^n$ the encryption of message block $m \in \{0,1\}^n$ under the key K. Given r plaintext-ciphertext pairs $\{(m_i, c_i)\}_{i=1}^r$ with $c_i = C_K(m_i)$, we aim to apply Grover's algorithm to find the unknown key K [YI00]. The Boolean function f for the Grover oracle takes a key K as input, and is defined as

$$f(K) = \begin{cases} 1, & \text{if } C_K(m_i) = c_i \text{ for all } 1 \leq i \leq r, \\ 0, & \text{otherwise.} \end{cases}$$

Possibly, there could exist other keys than K that encrypt the known plaintexts to the same ciphertexts. We call such keys *spurious keys*. If their number is known to be, say, $M - 1$, the M-solution version of Grover's algorithm has the same probability of measuring each spurious key as of measuring the correct K.

Spurious keys. We start by determining the probability that a single message encrypts to the same ciphertext under two different keys, for which we make the usual heuristic assumptions about the block cipher $C_{(\cdot)}$. We assume that under a fixed key K, the map $\{0,1\}^n \to \{0,1\}^n, m \mapsto C_K(m)$ is a pseudo-random permutation; and under a fixed message block m, the map

$\{0,1\}^k \to \{0,1\}^n, K \mapsto C_K(m)$ is a pseudo-random function[2]. Now let K be the correct key, i.e. the one used for the encryption. It follows that for a single message block of length n, $\Pr_{K \neq K'}(C_K(m) = C_{K'}(m)) = 2^{-n}$.

This probability becomes smaller when the equality condition is extended to multiple blocks. Given r distinct messages $m_1, \ldots, m_r \in \{0,1\}^n$, we have

$$\Pr_{K \neq K'}((C_K(m_1), \ldots, C_K(m_r)) = (C_{K'}(m_1), \ldots, C_{K'}(m_r))) = \prod_{i=0}^{r-1} \frac{1}{2^n - i}, \quad (2.1)$$

which is $\approx 2^{-rn}$ for $r^2 \ll 2^n$. Since the number of keys different from K is $2^k - 1$, we expect the number of spurious keys for an r-block message to be $\approx (2^k - 1)2^{-rn}$. Choosing r such that this quantity is very small ensures with high probability that there are no spurious keys, such that we can parametrise Grover's algorithm for a single solution.

Remark 12. *Grassl* et al. *[GLRS16, §3.1] work with a similar argument. They take the probability over pairs (K', K'') of keys with $K' \neq K''$. Since there are $2^{2k} - 2^k$ such pairs, they conclude that about $(2^{2k} - 2^k)2^{-rn}$ satisfy the above condition that the ciphertexts coincide on all r blocks. But this also counts pairs of keys for which the ciphertexts match each other, but do not match the images under the correct K. Thus, using the number of pairs overestimates the number of spurious keys and hence the number r of message blocks needed to ensure a unique key.*

Based on the above heuristic assumptions, one can determine the probability for a specific number of spurious keys. Let X be the random variable whose value is the number of spurious keys for a given set of r message blocks and a given key K. Then, X is distributed according to a binomial distribution:

$$\Pr(X = t) = \binom{2^k - 1}{t} p^t (1-p)^{2^k - 1 - t},$$

where $p = 2^{-rn}$. We use the Poisson limit theorem [Fel68, Chapter VI.5] to conclude that this is approximately a Poisson distribution with

$$\Pr(X = t) \approx e^{-\frac{2^k - 1}{2^{rn}}} \frac{(2^k - 1)^t (2^{-rn})^t}{t!} \approx e^{-2^{k-rn}} \frac{2^{t(k-rn)}}{t!}. \quad (2.2)$$

[2]Intuitively, a pseudo-random function is a function that cannot be easily distinguished from having random output; a similar intuition holds for a pseudo-random permutation. We refer the reader to [KL14, §3.5] for a more thorough formal discussion.

The probability that K is the unique key consistent with the r plaintext-ciphertext pairs is $\Pr(X = 0) \approx e^{-2^{k-rn}}$. Thus we can choose r such that rn is slightly larger than k; $rn = k + 10$ gives $\Pr(X = 0) \approx 0.999$. In a block cipher where $k = b \cdot n$ is a multiple of n, taking $r = b + 1$ will give the unique key K with probability at least $1 - 2^{-n}$, which is negligibly close to 1 for typical block sizes. If $rn < k$, then K is almost certainly not unique. Even $rn = k - 3$ gives less than a 1% chance of a unique key. Hence, r must be at least $\lceil k/n \rceil$.

The case $k = rn$, when the total message length is equal to the key length, remains interesting if one aims to minimize the number of qubits. The probability for a unique K is $\Pr(X = 0) \approx 1/e \approx 0.3679$, and the probability of exactly one spurious key is the same. Kim *et al.* [KHJ18, Eq. 7] describe the success probability after a certain number of Grover iterations when the number of spurious keys is unknown. The optimal number of iterations gives a maximum success probability of 0.556, making it likely that the first attempt will not find the correct key and one must repeat the algorithm if aiming for a larger success probability.

Remark 13. *While for some cryptanalytic applications, it is important to find the correct key, for others, any key that matches the plaintext-ciphertext pairs can be sufficient. For example, the Picnic signature scheme [CDG+17, ZCD+17] uses a block cipher $C_{(\cdot)}$ and encrypts a message m to c, and (m, c) is the public key. The signature is a zero-knowledge proof that the signer knows a secret key K such that $C_K(m) = c$. Any other key K' with $C_{K'}(m) = c$ produces a valid signature for the original public key. Thus, to forge signatures, a spurious key works just as well. However, since in general the number of spurious keys for a given plaintext-ciphertext pair is unknown, Grover's algorithm needs to be adjusted for example as in [BBHT98, §4] or by running a quantum counting algorithm first [BBHT98, §5]. This requires repeated runs of various Grover instances. Under a total depth limitation, this reduces the success probability of the attack compared to when using enough plaintext-ciphertext pairs such that no spurious keys are present.*

Depth constraints for cryptanalysis. In this chapter, we assume that any quantum adversary is bounded by a constraint on the total depth of

any quantum circuits they can evaluate. In its call for proposals to the post-quantum cryptography standardisation effort [Nat16], NIST introduces the parameter `MAXDEPTH` as such a bound and suggests that reasonable values[3] are between 2^{40} and 2^{96}. Whenever an algorithm's overall depth exceeds this bound, parallelisation becomes necessary. We do assume that `MAXDEPTH` constitutes a hard upper bound on the total depth of a quantum attack, including possible repetitions of a Grover instance.

In general, an attacker can be assumed to have a finite amount of resources, in particular a finite time for an attack. This is equivalent to postulating an upper bound on the total depth of a quantum circuit as suggested by NIST. Unlike in the classical case, the required parallelisation increases the gate cost for Grover's algorithm, which makes it important to study attacks with bounded depth.

We consider it reasonable to expect that the overall attack strategy is guaranteed to return a solution with high probability close to 1 within the given depth bound. E.g., a success probability of 1/2 for a Grover instance to find the correct key requires multiple runs to increase the overall probability closer to 1. These runs, either sequentially or in parallel, need to be taken into account for determining the overall cost and must respect the depth limit. While this setting is our main focus, it can be adequate to allow and cost a quantum algorithm with a success probability noticeably smaller than 1. Where not given in this chapter, the corresponding analysis can be derived in a straightforward manner.

2.2.3 Parallelization

Grover's algorithm is known to parallelise badly. Indeed, Zalka [Zal99] concludes that when using S parallel Grover oracles, the number of Grover iterations can be at most reduced by a factor $\Theta(\sqrt{S})$. In particular, we will aim at a

[3] Suggested `MAXDEPTH` values are justified by assumptions about the total available time and speed of each gate. The limit 2^{96} is given as "the approximate number of gates that atomic scale qubits with speed of light propagation times could perform in a millennium" [Nat16]. An adversary could only run a higher-depth circuit if they were able to use smaller qubits, faster propagation, or had more available time.

reduction factor of exactly $\sqrt{S}$, which also matches the trade-off considered by NIST (which we will discuss in Section 2.6.2). Compared to many classical algorithms, this is an inefficient parallelisation, since we must increase the circuit width by a factor of S to reduce the depth by a factor of $\sqrt{S}$.

There are different ways to parallelise Grover's algorithm. Kim, Han, and Jeong [KHJ18] describe two, which they denote as *inner* and *outer* parallelisation. Outer parallelisation runs multiple instances of the full algorithm in parallel. Only one instance must succeed, allowing us to reduce the necessary success probability, and hence number of iterations, for all. Inner parallelisation divides the search space into disjoint subsets and assigns each subset to a parallel machine. Each machine's search space is smaller, so the number of necessary iterations shrinks. Both methods avoid any communication, quantum or classical, during the Grover iterations. They require communication at the beginning, to distribute the plaintext-ciphertext pairs to each machine and to delegate the search space for inner parallelisation, and communication at the end to collect the measured keys and decide which one, if any, is the true key. The next section discusses why our setting favours inner parallelisation.

Advantages of inner parallelisation. We assume the notation introduced in Section 1.5.1. Consider S parallel machines that run for j iterations, and let us assume for the moment that no spurious keys are present in the search space ($M = 1$). For a single machine, the success probability is $p(j) = \sin^2((2j+1)\theta)$, where $\theta \approx \sin(\theta) = \sqrt{M/N} = \sqrt{N}^{-1}$. Using outer parallelisation, the probability that at least one machine recovers the correct key is $p_S(j) = 1 - (1 - p(j))^S$. We hope to gain a factor $\sqrt{S}$ in the number of iterations, so instead of iterating $\left\lfloor \frac{\pi}{4\theta} \right\rfloor$ times, we run each machine for $j_S = \left\lfloor \frac{\pi}{4\theta\sqrt{S}} \right\rfloor$ iterations.

Considering some small values of S, we get $S = 1 :\ p_1(j_1) \approx 1$, $S = 2 :\ p_2(j_2) \approx 0.961$ and $S = 3 :\ p_3(j_3) \approx 0.945$. As S gets larger, we use a series expansion to find that

$$p_S(j_S) \approx 1 - \left(1 - \frac{\pi^2}{4S} + O\left(\frac{1}{S^2}\right)\right)^S \xrightarrow{S\to\infty} 1 - e^{-\frac{\pi^2}{4}} \approx 0.915. \tag{2.3}$$

This means that by simply increasing S, it is not possible to gain a factor $\sqrt{S}$ in the number of iterations if one aims for a success probability close to 1. In contrast, with inner parallelisation, the correct key lies in the search space of exactly one machine. We are running j_S iterations, which is exactly the required number of iterations to find the key in a search space of size N/S. Therefore, this machine has near certainty of measuring the correct key, while other machines are guaranteed not to measure the correct key. Overall, we have near-certainty of finding the correct key. Inner parallelisation thus achieves a higher success probability than outer parallelisation, while using the same number S of parallel instances and the same number of iterations.

Another advantage of inner parallelisation is that dividing the search space separates any spurious keys into different subsets and reduces the search problem to finding a unique key. This allows us to reduce the number r of message blocks in the Grover oracle and was already observed by Kim, Han, and Jeong [KHJ18] in the context of measure-and-repeat methods. In fact, the correct key lies in exactly one subset of the search space. If the spurious keys fall into different subsets, the respective machines measure spurious keys, which can be discarded classically after measurement with access to the appropriate number of plaintext-ciphertext pairs. The only relevant question is whether there is a spurious key in the correct key's subset of size $2^k/S$. The probability for this is

$$\mathrm{SKP}(k,n,r,S) = \sum_{t=1}^{\infty} \Pr(X=t) \approx 1 - e^{-\frac{2^{k-rn}}{S}}, \tag{2.4}$$

using Equation (2.2) with 2^k replaced by $2^k/S$. If $k = rn$, this probability is $1/S + O(1/S^2)$ as $S \to \infty$. In general, high parallelisation makes spurious keys irrelevant, and the Grover oracle can simply use the smallest r such that $\mathrm{SKP}(k,n,r,S)$ is less than a desired bound.

2.3 Quantum circuit design

In Section 1.5 we introduced quantum computation in terms of the quantum circuit model. In this section we will discuss methods and criteria for quantum

circuit design, and cost models to estimate quantum resources, as relevant to our application.

2.3.1 Fault-tolerant gate set and architecture assumptions

We adopt the computational model presented in [JS19]. The quantum circuits we are concerned with in this chapter operate on qubits. They are composed of so-called Clifford+T gates, which form a commonly used universal fault-tolerant gate set exposed by several families of quantum error-correcting codes [Ter15]. The primitive gates consist of single-qubit Clifford gates, controlled-NOT (CNOT) gates, T gates, and measurements. We make the standard assumption of *full parallelism*, meaning that a quantum circuit can apply any number of gates simultaneously so long as these gates act on disjoint sets of qubits [GR04, BBG+13].

All quantum circuits for AES and LowMC described in this chapter were designed, tested, and costed in the Q# programming language [SGT+18], which supports all assumptions discussed here. Q# allows to describe circuits in terms of single qubit gates (the Pauli gates X, Y, Z, the Hadamard gate H, the phase gate S, the T gate, general rotation gates), and controlled gates. Furthermore, it makes classical control logic around quantum operations transparent, so that loops and conditional statements based on measurement output can be easily expressed. The Q# compiler allows us to compute circuit depth automatically by moving gates around through a circuit if the qubits it acts on were previously idle. In particular, this means that the depth of two circuits applied in series may be less than the sum of the individual depths of each circuit. The Q# language allows the circuit to *allocate* auxiliary qubits as needed, which adds new qubits initialized to $|0\rangle$. If an auxiliary qubit is returned to the state $|0\rangle$ after it has been operated on, the circuit can *release* it. Such a qubit is no longer entangled with the state used for computation and the circuit can now maintain or measure it.

Grover's algorithm is a far-future quantum algorithm, making it difficult to decide on the right cost for each gate. Previous work assumed that T gates

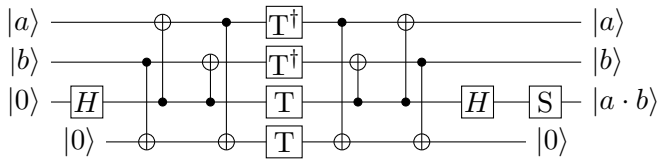
constitute the main cost [GLRS16, ASAM18, LPS20]. They are exceptionally expensive for a surface code [FMMC12]; however, for a future error-correcting code, T gates may be transversal and cheap while a different gate may be expensive. Thus, we present costs when both costing depth in terms of T gates only, and when costing all gates equally.

We ignore all concerns of layout and communication costs for the Grover oracle circuit. Though making this assumption is unrealistic for a surface code, where qubits can only interact with neighbouring ones, other codes may not have these issues. A single oracle circuit uses relatively few logical qubits ($< 2^{20}$), so these costs are unlikely to dominate. This allows us to compare our work with previous proposals, which also ignore these costs. This also implies that uncontrolled swaps are free, since the classical controller can simply track such swaps and rearrange where it applies subsequent gates.

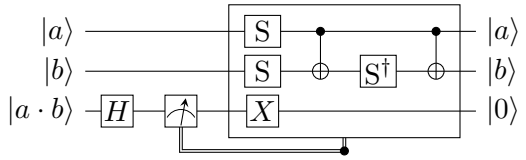
2.3.2 Realising the AND gate.

Previous work on quantum circuits for AES such as [GLRS16, ASAM18, LPS20] mainly uses Toffoli gates to realise the functionality of the classical AND gate, which is used as part of the AES S-box. We instead opt for a different approach, and implement a "quantum AND" gate instead. A quantum AND gate has the same functionality as a Toffoli gate, except the target qubit is assumed to be in the state $|0\rangle$ on input, rather than in an arbitrary state. We use a combination[4] of Selinger's [Sel13] and Jones' [Jon13] circuits to express the AND gate in terms of Clifford and T gates. This circuit uses 4 T gates and 11 Clifford gates in T-depth 1 and total depth 8. It uses one auxiliary qubit which it immediately releases, while its adjoint circuit is slightly smaller. Of particular interest is that the adjoint operator AND† has T-depth 0, at the cost of requiring the use of one measurement and of executing some gates conditioned on the output of such measurement. Diagrams of quantum AND and its adjoint are shown in Figure 2.1. The result of the measurement in AND† is uniformly distributed when uncomputing AND, independently of the inputs to the AND gate. This means that for large circuits one can estimate the

[4] We thank Mathias Soeken for providing the implementation of the AND gate circuit.



(a) AND gate.



(b) AND† gate.

Figure 2.1: AND gate design used in our circuit. We notice that in Figure 2.1b, the measurement returns a classical bit b and leaves the original qubit in the state $|b\rangle$. The value of b is used to conditionally apply the gates inside the box.

average cost of uncomputing AND by taking the average circuit size of AND† assuming half of the uncomputations execute the gates in box in Figure 2.1b, and half do not.

2.3.3 Automated resource estimation and unit tests

One incentive for producing full implementations of the Grover oracle and its components is to obtain precise[5] resource estimates automatically and directly from the circuit descriptions. Another incentive is to test the circuits for correctness and to compare results on classical inputs against existing classical software implementations that are known (or believed) to be correct. Yet quantum circuits are in general not testable, since they rely on hardware yet to be constructed. To partially address this issue the Q# compiler and runtime can classically simulate a subset of quantum circuits, enabling partial test coverage. We thus designed our circuits such that this tool can fully classically simulate them, by using X, CNOT, CCNOT, SWAP, and AND gates only, together with measurements (denoted throughout as M "gates"). This approach limits the design space since we cannot use true quantum methods within the oracle. Yet,

[5]Since the publication of the paper that led to this chapter, a problem with the ResourcesEstimator functionality in Q# was reported in `https://github.com/microsoft/qsharp-runtime/issues/192` and solved in `https://github.com/microsoft/qsharp-runtime/pull/404`. Results reported in this chapter describe achievable circuits.

it is worthwhile to implement components that are testable and can be fully simulated to increase confidence in the validity of resource estimates deduced from such implementations.

As part of the development process, we first implemented AES (resp. LowMC) in Python 3, and tested the resulting code against the AES implementation in PyCryptodome 3.8.2 [PyC19] (resp. the C++ reference implementation in [Low19]). Then, we proceeded to write our Q# implementations (running on the .NET Core version 3.1, using the Microsoft Quantum Development Kit (QDK) version 0.15.2101125897[6]), and tested these against our Python 3 implementations, by making use of the IQ# interface [Mic19a, Mic19b]. For the Q# simulator to run, we are required to use the Microsoft QDK standard library's Toffoli gate for evaluating both Toffoli and AND gates, which results in deeper than necessary circuits. We also have to explicitly SWAP values across wires, which costs 3 CNOT gates, rather than simply keeping track of the necessary free rewiring. Hence, to mitigate these effects, our functions admit a Boolean flag indicating whether the code is being run as part of a unit test by the simulator, or as part of a cost estimate. In the latter case, Toffoli and AND gate designs are automatically replaced by shallower ones, and SWAP instructions are disregarded as free (after manually checking that this does not allow for incompatible circuit optimizations that could result in lower costs than expected). All numbers reporting the total width of a circuit include the initial number of qubits plus the maximal number of temporarily allocated auxiliary qubits within the Q# function. For numbers describing the total depth, all gates such as Clifford gates, CNOT and T gates as well as measurements are assigned a depth of 1.

2.3.4 Current limitations of the Q# resource estimator

The prospect of automating resource estimation makes Q# a very interesting tool. However, being a tool in a relatively early stage of development, issues are to be expected. In this section we explain the issues we faced during the

[6]The published version of this chapter reports previous versions of the .NET Core and Q# SDK. We report the versions used to compute the updated numbers provided in this chapter.

development of this chapter, how they affected previous versions of this work, and how we work around those still present. We stress that to the best of our knowledge the numbers output by the current QDK, using our workarounds, faithfully represent the cost of our circuits under the assumptions previously listed in Section 2.3.1.

Attainable depth and width. At the time of publication of this chapter as [JNRV20], the available version of the Microsoft QDK was 0.7.1905.310. The resource estimator at the time had an undocumented feature that would result in the reported depth and width of a circuit not always being attainable at the same time. As an example[7], the Q# code in Figure 2.2 would be estimated as describing a circuit of depth *and* width 1. A circuit of depth 1 is possible (Figure 2.3a) and similarly a circuit of width 1 is possible (Figure 2.3b). In order to obtain a circuit of depth *and* width 1 the compiler would need to start from the circuit of width 1, ignore the release and re-allocation of the wire between `using` statements and simplify $T^2 = S$. However, the Q# compiler is currently unable to perform the latter simplification, therefore obtaining a circuit of width *and* depth 1 starting from the given code is not possible as far as the compiler could "see". We stress that while for this simple example a circuit of depth and width 1 happens to be achievable, this is not true in general, nor what the compiler shipped in version 0.7.1905.310 of the QDK meant by reporting such lower bound on the circuit size.

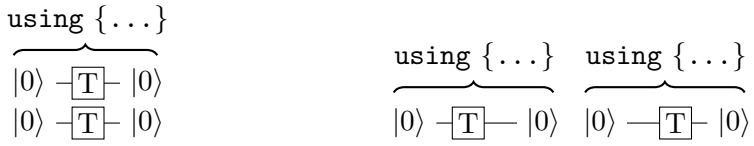
```
1 using(q = Qubit()) { T(q); }
2 using(q = Qubit()) { T(q); }
```

Figure 2.2: Example circuit reporting incompatible depth and width when using version 0.7.1905.310 of the QDK.

As of version 0.15.2101125897, the resource estimator can be requested to return the size of a circuit of minimal[8] depth or width, with the guarantee that the returned values will be attainable at the same time, as long as measurement output is not used to conditionally execute other gates (see next item).

[7] The example used was taken from `https://github.com/microsoft/qsharp-runtime/pull/404`.

[8] The circuits are *minimal* in the sense that the compiler can not find a smaller (in the selected metric) circuit, not necessarily that a smaller one does not exist.



(a) Depth = 1 (b) Width = 1

Figure 2.3: Attainable circuits of depth 1 *or* width 1 corresponding to the Q# code in Figure 2.2.

Gates conditioned on measurements. As mentioned in Section 2.3.2, in order to optimise the depth of our AES circuits we use an implementation of the quantum AND gate that has T-depth 1, and that can be uncomputed without using any T gates at the cost of introducing measurements. As can be see in Figure 2.1b, part of the uncomputation circuit is executed conditionally on the output of a measurement. As of version 0.15.2101125897, the resource estimator simulates the output of measurements, and is able to add the conditional gates to the quantum circuit depending on the simulated output [Vas21, p. 5], which will result in correct gate counts given the measurement outcomes. However, the estimator is not able to account for the conditional gates not being executable in parallel to the measurement. In our case, this only impacts the uncomputation of the AND gate, since this is the only component in our oracles using measurements to evaluate gates conditionally. In particular, this means that whenever the measurement in Figure 2.1b returns 1, the circuit depth is estimated as 4 rather than 6 (see Figure 2.4 for a diagram of the incorrect circuit produced).

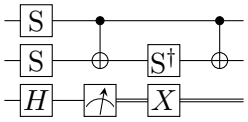


Figure 2.4: Incorrect AND† circuit, as compiled by the QDK version 0.15.2101125897 when the output of the measurement is $|1\rangle$. The gate count is correct, but the depth is not.

To work around this fact, when running cost estimates we add on the middle wire a sequence of two rotation gates R and $R^\dagger$, to be executed (conditionally) after the second CNOT. This results in the depth being correctly estimated as 6 by the current compiler, and since the number of rotation gates is reported separately, it allows us to discard such gates from the total gate count.

Probabilistic estimates. As mentioned above, the resource estimator in the current QDK assigns measurement outputs randomly. Therefore, when estimating the size of oracle circuits containing AND† gates, the CNOT, 1-qubit Clifford, measurement and depth counts are probabilistic. The Q# simulator does not currently support PRNG seeding for de-randomizing the measurements,[9] which means that re-estimating the same circuit multiple times may result in slightly different numbers. In our case, across multiple runs of resource estimation the final attack costs are essentially not impacted by these lower order fluctuations.

Circuit optimization. The Q# compiler's optimization capabilities are currently limited [Vas21]. For example, the sequential R and $R^\dagger$ evaluations we artificially added to the adjoint AND gate (which essentially amount to a no-op of depth 2), are not automatically simplified and eliminated. Reruns of the same circuit with future improved versions of the compiler may result in smaller estimated costs.

Large circuit compilation. The current version of the QDK struggles compiling very large circuits[10]. While this is not an issue for most useful applications of quantum computing, it can hinder compiling very large cryptanalytic circuits.

How our work is affected. The issues highlighted in this section can luckily be worked around, making this chapter possible. We re-estimated all the circuits in this chapter (except for the largest LowMC parameters, see below) with the current version of the compiler in order to report attainable depth and width values. Furthermore, we tweaked the AND† circuit as described above to report correct depths. To address the probabilistic nature of the estimates, we fixed the probability of values output by the measurement in AND† to be $P[|0\rangle] = P[|1\rangle] = 1/2$, matching the circuit description in Section 2.3.2. The cost estimates of the key search attacks stayed essentially the same across

[9] `https://github.com/microsoft/qsharp-runtime/issues/30`.
[10] `https://github.com/microsoft/qsharp-compiler/issues/875`.

versions of the QDK: for our circuits the lower bound width was attainable even when generating minimal depth circuits, and having slightly deeper (on average) $\text{AND}^\dagger$ gates did not impact the overall results. The only exception was the block cipher that we call LowMC L5 (which will be described in Section 2.5). While its circuit used to compile under version 0.7 of the QDK, it does fail under version 0.15.2101125897. However, comparing the cost of our smaller LowMC circuits (L1, L3) in this chapter to those originally generated with version 0.7, they stayed essentially identical. Since the structure of LowMC L5 is identical (only, larger) to that of its smaller parametrisations, only for LowMC L5 we will keep reporting the numbers from our original publication [JNRV20], since we don't believe they would be affected by the changes made since.

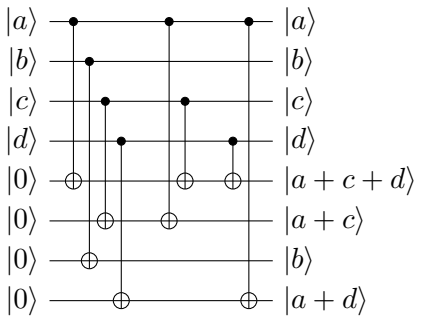
2.3.5 Reversible circuits for linear maps

Linear maps $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ for varying dimensions n and m are essential building blocks of AES and LowMC. In general, such a map f, expressed as multiplication of an input column vector in $\mathbb{F}_2^n$ by a constant matrix $M_f \in \mathbb{F}_2^{m\times n}$, can be implemented as a reversible circuit on n input wires and m additional output wires (initialized to $|0\rangle$), by using an adequate sequence of CNOT gates: if the (i,j)-th coefficient of M_f is 1, we set a CNOT gate targeting the i-th output wire, controlled on the j-th input wire.

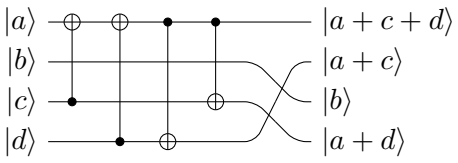
Yet, if a linear map $g\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ is invertible, one can reversibly compute it in-place on the input wires via a PLU decomposition of M_g, $M_g = P \cdot L \cdot U$. The lower- and upper-triangular components L and U of the decomposition can be implemented by using the appropriate CNOT gates in a similar fashion to backward and forward substitution, while the final permutation P does not require any quantum gates and instead is realized by appropriately keeping track of the necessary rewiring. An example of a linear map decomposed in both ways is shown in Figure 2.5. While rewiring is not easily supported in Q#, the same effect can be obtained by defining a custom `REWIRE` operation that computes an in-place swap of any two wires when testing an implementation, and that can be disabled when costing it. We note that PLU decompositions are not generally unique, but it is not clear whether sparser decompositions can be

$$M = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = P \cdot L \cdot U$$

(a) Invertible linear transformation M and its PLU decomposition.



(b) Naive circuit computing M.



(c) In-place implementation of M.

Figure 2.5: Alternative circuits implementing the same linear transformation $M\colon \mathbb{F}_2^4 \to \mathbb{F}_2^4$, by using the two strategies described in § 2.3.5. Both are direct implementations, and could potentially be reduced in size by automatic means as in [MSR+19, MSC+19, GKMR14, ZC19], or manually. Figure 2.5b is wider and has a larger gate count, but is shallower, than Figure 2.5c.

consistently obtained with any particular technique. For our implementations, we perform PLU decompositions using SageMath 8.1 [S+17], which internally relies on the M4RI [AB19] library.

2.3.6 Cost metrics for quantum circuits

For a meaningful cost analysis, we assume that an adversary has fixed constraints on their total available resources, and a specific cost metric they wish to minimize. Without such limits, we might conclude that AES-128 could be broken in under a second using 2^{128} machines, or broken using only a few thousand qubits but a billion-year runtime. Most importantly, we assume a total depth limit $D_{\max}$ as explained in Section 2.2.2.

In this chapter, we use the two cost metrics that are considered by Jaques and Schanck in [JS19]. The first is the total number of gates, the *G-cost*. It assumes non-volatile ("passive") quantum memory, and therefore models circuits that incur some cost with every gate, but where no cost is incurred in time units during which a qubit is not operated on.

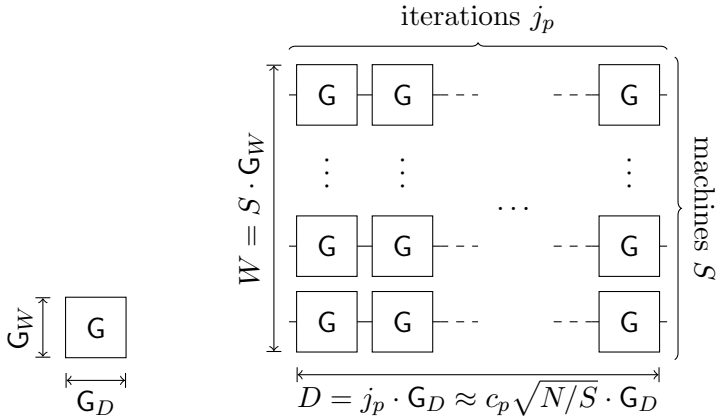
The second cost metric is the product of circuit depth and width, the *DW-cost*. This is a more realistic cost model when quantum error correction is necessary. It assumes a volatile ("active") quantum memory, which incurs some cost to correct errors on every qubit in each time step, i.e. each layer of the total circuit depth. In this cost model, a released auxiliary qubit would not require error correction, and the cost to correct it could be omitted. But we assume an efficient strategy for qubit allocation that avoids long idle periods for released qubits and thus choose to ignore this subtlety. Instead, we simply cost the maximum width at any point in the oracle, times its total depth. For both cost metrics, we can choose to count only T-gates towards gate count and depth, or count all gates equally.

2.3.7 The cost of Grover's algorithm

We will now reason on the cost of (parallelised) Grover's algorithm, how this behaves under a maximum depth constraint and how it compares to classical search. We will also give a lower bound for the depth of key search. All our results assume a straightforward[11] parallel Grover strategy. We will use the notation laid out in Section 1.5.1 for the runtime of Grover's algorithm.

Parallel Grover circuits. Let the search space have size $N = 2^k$, the target be unique ($M = 1$) and $\theta \approx \sin(\theta) = \sqrt{M/N} = \sqrt{N}^{-1}$. Suppose we use an oracle G such that a single Grover iteration costs G_G gates, has depth G_D, and uses G_W qubits. Let $S = 2^s$ be the number of parallel machines that are used with the inner parallelisation method by dividing the search space in S disjoint parts (see Section 2.2.3). In order to achieve a certain success probability p, the

[11] Our results do not cover any gains from the Search With Two Oracles technique from [DP19, DP21].



(a) Grover oracle (b) Parallel Grover circuit

Figure 2.6: Size of a circuit for running parallel Grover's algorithm (right), given the size of the function oracle used (G, left). In the case of key search, the size of G will depend on the spurious key probability (SKP, see § 2.2.3), which itself depends on the number of parallel computers S.

required number of iterations j_p can be deduced from $p = \sin^2((2j+1)\theta)$ which yields $j_p = \lceil(\arcsin(\sqrt{p})/\theta - 1)/2\rceil \approx (\arcsin(\sqrt{p})/2) \cdot \sqrt{N/S}$ (see Figure 2.6).

Let $c_p \coloneqq \arcsin(\sqrt{p})/2$, then the total depth of a j_p-fold Grover iteration is

$$D = j_p\mathsf{G}_D \approx c_p\sqrt{N/S} \cdot \mathsf{G}_D = c_p 2^{\frac{k-s}{2}}\mathsf{G}_D \text{ cycles.} \tag{2.5}$$

Note that for $p \approx 1$ we have $c_p \approx c_1 = \frac{\pi}{4}$. Each machine uses $j_p\mathsf{G}_G \approx c_p\sqrt{N/S} \cdot \mathsf{G}_G = c_p 2^{\frac{k-s}{2}}\mathsf{G}_G$ gates, i.e. the total G-cost over all S machines is

$$G = S \cdot j_p\mathsf{G}_G \approx c_p\sqrt{N \cdot S} \cdot \mathsf{G}_G = c_p 2^{\frac{k+s}{2}}\mathsf{G}_G \text{ gates.} \tag{2.6}$$

Finally, the total width is $W = S \cdot \mathsf{G}_W = 2^s\mathsf{G}_W$ qubits, which leads to a DW-cost

$$DW \approx c_p\sqrt{N \cdot S} \cdot \mathsf{G}_D\mathsf{G}_W = c_p 2^{\frac{k+s}{2}}\mathsf{G}_D\mathsf{G}_W \text{ qubit-cycles.} \tag{2.7}$$

These cost expressions show that minimizing the number $S = 2^s$ of parallel machines minimizes both G-cost and DW-cost. Thus, under fixed limits on depth, width, and the number of gates, an adversary's best course of action is to use the entire depth budget and parallelise as little as possible. Under this premise, the depth limit fully determines the optimal attack strategy for a given Grover oracle. Limits on width or the number of gates simply become

binary feasibility criteria and are either too tight and the adversary cannot finish the attack with the desired success probability, or one of the limits is loose. If one resource limit is loose, we may be able to modify the oracle to use this resource to reduce depth, lowering the overall cost.

Optimizing the oracle under a depth limit. Grover's algorithm parallelises so badly that it is generally preferable to aggressively optimise the oracle's depth. This allows more iterations within the depth limit, thus reducing the necessary parallelisation.

In order to deduce the number S of parallel machines required, let $D_{\max}$ be a fixed depth limit. Given the depth G_D of the oracle, we are able to run $j_{\max} = \lfloor D_{\max}/\mathsf{G}_D \rfloor$ Grover iterations of the oracle G. For a target success probability p, we obtain the number S of parallel instances to achieve this probability in the instance whose key space partition contains the key from $p = \sin^2((2j_{\max}+1)\sqrt{S/N})$ as

$$S = \left\lceil \frac{N \cdot \arcsin^2\left(\sqrt{p}\right)}{(2 \cdot \lfloor D_{\max}/G_D \rfloor + 1)^2} \right\rceil \approx c_p^2 2^k \frac{\mathsf{G}_D^2}{D_{\max}^2}. \tag{2.8}$$

Using this in Equation (2.6) gives a total gate count of

$$G = c_p^2 2^k \frac{\mathsf{G}_D \mathsf{G}_G}{D_{\max}} \text{ gates.} \tag{2.9}$$

It follows that for two oracle circuits G and F, the total G-cost is lower for G if and only if $\mathsf{G}_D\mathsf{G}_G < \mathsf{F}_D\mathsf{F}_G$. That is, we wish to minimize the product $\mathsf{G}_D\mathsf{G}_G$. Similarly, the total DW-cost under the depth constraint is

$$DW = c_p^2 2^k \frac{\mathsf{G}_D^2 \mathsf{G}_W}{D_{\max}} \text{ qubit-cycles.} \tag{2.10}$$

Here, we wish to minimize $\mathsf{G}_D^2\mathsf{G}_W$ of the oracle circuit to minimize total DW-cost, with the higher power on G_D suggesting again that minimizing depth should be prioritized over minimizing G_W.

Comparing parallel Grover search to classical search. In the computational model of [JS19], each quantum gate is interpreted as some computation done by a classical controller. For certain parameter settings, these controllers

may find the key more efficiently through a classical search. Assume, this is done with a brute force algorithm, which simply iterates through all potential keys and checks if they are correct. Let C be the classical gate cost to test a single key. Then for a search space of size $N = 2^k$, the total cost in terms of classical gate evaluations for the brute force attack to achieve success probability p is $p2^k\mathsf{C}$ (where the attack is trivially parallelisable). Comparing this cost to the gate cost for Grover's algorithm in Equation (2.6), we conclude that if we use $S \geq (p\mathsf{C}/(c_p\mathsf{G}_G))^2 2^k$ parallel machines, Grover's algorithm will require more quantum gates than classical gate evaluations are required to run exhaustive search on the same hardware.

Since the Grover oracle G includes a reversible evaluation of the block cipher and since quantum computation of a function is likely more costly than its classical counterpart, we may assume that the classical gate cost C is smaller than the quantum gate cost G_G of the Grover oracle, i.e. $\mathsf{C} \leq \mathsf{G}_G$. From the definition of $c_p = \arcsin\big(\sqrt{p}\big)/2$ above, it holds that $p/c_p < 1.45$, so that $(p\mathsf{C}/(c_p\mathsf{G}_G))^2 < 2.11$. In particular, for $p = 1$ we have $(p\mathsf{C}/(c_p\mathsf{G}_G))^2 = 16/\pi^2 \cdot \mathsf{C}^2/\mathsf{G}_G^2 \approx 1.62 \cdot \mathsf{C}^2/\mathsf{G}_G^2 \leq 1.62$. Depending on the actual cost ratio C/G_G, the resulting bound on the number of parallel quantum machines after which classical search is cheaper may affect the viability of Grover's algorithm for key search. This may especially be the case when taking energy or monetary costs per gate evaluation into consideration.

Communication cost to assemble the results. We briefly discuss the communication cost incurred by communicating a found solution from one of the machines in a large network of parallel computers to a central processor. Here, each machine measures a candidate key after a specified number of Grover iterations. The classical controller then checks this key against a small number of given plaintext-ciphertext pairs in order to determine whether it is a valid solution. If the key is correct, it is communicated to a central processor.

If the number of machines is small, the central processor simply queries each machine sequentially for the correct key. For a large number of machines, we instead assume they are connected in a binary tree structure with one machine

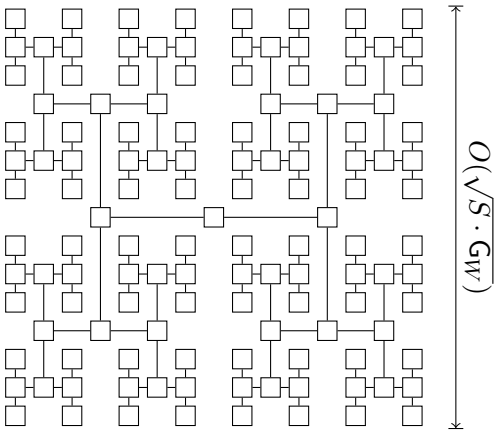


Figure 2.7: H-tree arrangement of parallel machines. Given S machines, it has area $O(S \cdot \mathsf{G}_W)$ and edge $O(\sqrt{S \cdot \mathsf{G}_W})$.

designated as the root. The central processor queries this one for the final result. If it has measured a correct key, it is returned, otherwise it asynchronously queries two other machines which form the roots of equally-sized sub-trees, in which the same process is repeated. For S machines this requires S requests, but only $\log S$ must be sequential.

We assume that the spatial arrangement of the S machines is in a two-dimensional plane in the form of an H-tree (see Figure 2.7). Assuming that in each machine qubits are arranged on a square board, this results in a squared arrangement with area $O(S \cdot \mathsf{G}_W)$ [Lei80]. Furthermore, it can be assumed that communication between machines is via classical channels with very small signal propagation times. The total distance any signal must travel is proportional to the square root of the size of this tree, i.e. $\sqrt{S}$. Thus, the total time to recover the final key is $O(\log S) + c_S\sqrt{S\mathsf{G}_W}$ cycles, where $c_S \sim \frac{\text{depth}}{\sqrt{G_W}} \sim \frac{\text{depth/time}}{\text{length/time}}$ is a constant to account for the ratio between quantum gate application times and classical signal propagation speed, as to keep our unit of measure of computation time in terms of quantum gate cycles. For large S, the $O(\log S)$ term is negligible, and therefore we ignore it.

We assume that $c_S \ll 1$, meaning that these classical channels can propagate a signal across a qubit-sized distance much faster than we can apply a gate to that qubit. This means the depth of each Grover search will dwarf the

communication costs so long as

$$c_S\sqrt{S\mathsf{G}_W} \leq D = c_p\sqrt{\frac{N}{S}}\cdot\mathsf{G}_D \iff S \leq 2^{k/2}\frac{c_p\mathsf{G}_D}{c_S\sqrt{\mathsf{G}_W}}.$$

Assuming equalities in place of $\leq$ and letting $x = D = c_s\sqrt{S\mathsf{G}_W}$, the total cost of the attack in terms of depth when including communication in our estimate would become $2\,x$. Replacing S with $\lambda^2\,S$ for some factor λ^2 would then result in a total depth of $f(\lambda) = \left(\lambda + \frac{1}{\lambda}\right)x$. It is easy to verify that the minimum of f is at $\lambda = 1$, meaning that $S = 2^{k/2}\frac{c_p\mathsf{G}_D}{c_S\sqrt{\mathsf{G}_W}}$ gives the minimum possible depth for a quantum search attack as

$$D_{\min} = c_p\sqrt{\frac{N}{S}}\cdot\mathsf{G}_D = 2^{\frac{k}{4}}\sqrt{c_pc_S\mathsf{G}_D\sqrt{\mathsf{G}_W}} \text{ cycles,} \tag{2.11}$$

Unless we can construct a three-dimensional layout[12], we cannot solve the search problem with Grover's algorithm in depth less than (2.11). Furthermore, this requires using $\Theta(2^{k/2}/c_S)$ quantum computers. Using the values for G_D and G_W listed in Tables 2.9 and 2.10 for the oracles we will design in the next sections, (2.11) implies that for AES-128, 192, and 256 the minimum depths are $2^{40.2}\,c_S^{1/2}$, $2^{56.2}\,c_S^{1/2}$ and $2^{72.3}\,c_S^{1/2}$, respectively. For LowMC-128, 192, and 256 the minimum depths are respectively $2^{42.8}\,c_S^{1/2}$, $2^{59.8}\,c_S^{1/2}$ and $2^{76.4}\,c_S^{1/2}$.[13]

2.4 A quantum circuit for AES

The Advanced Encryption Standard (AES) [DR99, DR01] is a block cipher standardized by NIST in 2001. Using the notation from the original submission [DR99], AES is composed of an S-box, a Round function (with subroutines ByteSub, ShiftRow, MixColumn, AddRoundKey; with the last round slightly differing from the others), and a KeyExpansion function (with subroutines SubByte, RotByte). The pseudo-code from [DR99, §4.4] is reported in simplified fashion in Algorithm 4.

We note that the FIPS specification for AES [DR01] renamed some of the subroutines from [DR99]. For reference, these are ByteSub to SubBytes,

[12] A truly three-dimensional layout seems unlikely, though an adversary with the resources to build 2^{64} quantum computers may also be able to launch them into orbit and assemble them into a sphere.

[13] In order to provide a strict lower bound, we use numbers from the shallowest oracles, and assume $r = 1$.

ShiftRow to ShiftRows, and MixColumn to MixColumns in the Round function, and SubByte to SubWord and RotByte to RotWord in the KeyExpansion function. In this chapter, we keep using the names from [DR99] as we used at the time of publication of this chapter as [JNRV20], in order to keep our circuits' descriptions compatible with the Q# implementation available at `github.com/microsoft/grover-blocks`.

Input: message m
Input: key k
1 $s \leftarrow m$
2 $ek \leftarrow \text{KeyExpansion}(k)$
3 $s \leftarrow \text{AddRoundKey}(s, k)$
4 **for** $i \leftarrow 1$ **to** total rounds $- 1$ **do**
5 $\quad s \leftarrow \text{Round}(s, ek)$
6 $c \leftarrow \text{FinalRound}(s, ek)$
7 **return** c

Algorithm 4: AES.

Three different instances of AES have been standardized, for key lengths of 128, 192 and 256 bits. Grassl *et al.* [GLRS16] describe a quantum circuit implementation of the S-box and other components, resulting in a full description of all three instances of AES (but no testable code has been released). They take care to reduce the number of auxiliary qubits required, i.e. reducing the circuit *width* as much as possible. The recent improvements by Langenberg *et al.* [LPS20] build on the work by Grassl *et al.* with similar objectives.

In this section, we describe our implementation of AES in the quantum programming language Q# [SGT+18]. Some of the components are taken from the description in [GLRS16], while others are implemented independently, or ported from other sources. We take the circuit description from [GLRS16] as the basis for our work and compare to the results in [LPS20]. In general, we aim at reducing the *depth* of the AES circuit, while limitations on width are less important. Width restrictions are not explicitly considered by the NIST call for proposals [Nat16, § 4.A.5].

The internal state of AES contains 128 bits, arranged in four 32-bit (or 4-byte) words. In the rest of this section, when referring to a "word", we intend a 4-byte word. In all tables below, we denote by #CNOT, the number of CNOT

gates, by #1qCliff the number of 1-qubit Clifford gates, by $\#T$ the number of T gates, by $\#M$ the number of measurement operations, by D the circuit depth when considering all gates, by T-D the circuit depth when considering only T gates and by W the maximum number of qubits used.

2.4.1 S-box, ByteSub and SubByte

The AES S-box is an invertible non-linear transformation on a byte, that interprets the input as an element of $\mathbb{F}_{256}$, inverts it (mapping 0 to 0), and applies an affine transformation over $\mathbb{F}_2$ to the inverted output. The S-box is the only source of T gates in a quantum circuit for AES. On classical hardware, it can be implemented easily using a lookup-table. Yet, on a quantum computer, this is not efficient [BGB+18, LKS18, Gid19]. Alternatively, the inversion can be computed either by using some variant of Euclid's algorithm (taking care of the special case of 0), or by applying Lagrange's theorem and raising the input to the $(|\mathbb{F}_{256}^{\times}|-1)$-th power (i.e. the 254-th power), which incidentally also takes care of the 0 input. Grassl *et al.* [GLRS16] suggest an Itoh-Tsujii inversion algorithm [IT88], following [ARS13], and compute all required multiplications over $\mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$. This idea had already been extensively explored in the vast[14] literature on hardware design for AES, and requires a different construction of $\mathbb{F}_{256}$ to be most effective. Following this lead, we ported the S-box circuit by Boyar and Peralta from [BP12] to Q#. The specified linear program combining AND and XOR operations can be easily expressed as a sequence of equivalent quantum AND and CNOT operations (see Section 2.3.2). We present cost estimates for the AES S-box in Table 2.1. We compare our port of [BP12] to our own Q# implementation of the S-box circuits from [GLRS16] and [LPS20]. ByteSub is a state-wide parallel application of the S-box, requiring new output auxiliary qubits to store the result, while SubByte is a similar word-wide application of the S-box.

Remark 14. *Langenberg* et al. *[LPS20] independently introduced a new AES quantum circuit design using the S-box circuit proposed in [BP10]. They also present a ProjectQ [SHT18] implementation of the S-box, albeit without unit*

[14] E.g. see [Rij00, SMTM01, BP10, BP+19, JKL10, NNT+10, UHS+15, RTA18, RMTA18, WSH+19].

operation	#CNOT	#1qCliff	#T	#M	T-D	D	W
[GLRS16] S-box	8683	1028	3584	0	217	1692	44
[BP10] S-box	810	248	164	41	35	511	41
[BP12] S-box	660	196	136	34	6	100	137

Table 2.1: Comparison of our reconstruction of the original [GLRS16] S-box circuit with the one from [BP10] as used in [LPS20] and the one in this work based on [BP12]. In our implementation of [BP10] from [LPS20], we replace CCNOT gates with AND gates to allow a fairer comparison.

tests. We ported their source code to Q#, tested and costed it. For a fairer comparison, we replaced their CCNOT gates with the AND gate design that our circuits use. Cost estimates can be found in Table 2.1. Overall, the [BP12] S-box leads to a more cost effective circuit for our purposes in both the G-cost and DW-cost metrics, and hence we did not proceed further in our analysis of costs using the [BP10] design. Note that the results obtained here differ from the ones presented in [LPS20, §3.2]. This is due to the difference in counting gates and depth. While [LPS20] counts Toffoli gates, the Q# resource estimator costs at a lower level of T gates and also counts all gates needed to implement a Toffoli gate.

2.4.2 ShiftRow and RotByte

ShiftRow is a permutation on the full 128-bit AES state, happening across its four words [DR99, §4.2.2]. As a permutation of qubits, it can be entirely encoded as rewiring. As in [GLRS16], we consider rewiring as free and do not include it in our cost estimates. Similarly, RotByte is a circular left shift of a word by 8 bits, and can be implemented by appropriate rewiring.

2.4.3 MixColumn

The operation MixColumn interprets each word in the state as a polynomial in $\mathbb{F}_{256}[x]/(x^4+1)$. Each word is multiplied by a fixed polynomial $c(x)$ [DR99, § 4.2.3]. Since the latter is coprime to x^4+1, this operation can be seen as an

operation	#CNOT	#1qCliff	$\#T$	$\#M$	T-D	D	W
In-place MixColumn	1108	0	0	0	0	111	128
[Max19] MixColumn	1248	0	0	0	0	22	318

Table 2.2: Comparison of an in-place implementation of MixColumn (via PLU decomposition) versus the recent shallow out-of-place design in [Max19].

invertible linear transformation, and hence can be implemented in place by a PLU decomposition of a matrix in $\mathbb{F}_2^{32\times32}$. To simplify this tedious operation, we use SageMath [S+17] code that performs the PLU decomposition, and outputs equivalent Q# code. Note that Grassl *et al.* [GLRS16] describe the same technique, while achieving a significantly smaller design than the one we obtain (ref. Table 2.2), but we were not able to reproduce their results. However, highly optimized, shallower circuits have been proposed in the hardware design literature, such as [JMPS17, KLSW17, BFI19, EJMY19, TP19]. Hence, we also chose to use a recent design by Maximov [Max19]. Both circuits are costed independently in Table 2.2. Maximov's circuit has a much lower depth, but it only reduces the total depth, does not reduce the T-depth (which is already 0) and comes at the cost of an increased width. Our estimates show that without a depth restriction, it seems advantageous to use the in-place version to minimize both G-cost and DW-cost metrics, while for a depth restricted setting, Maximov's circuit seems better due to the square in the depth term in Equation (2.10).

2.4.4 AddRoundKey

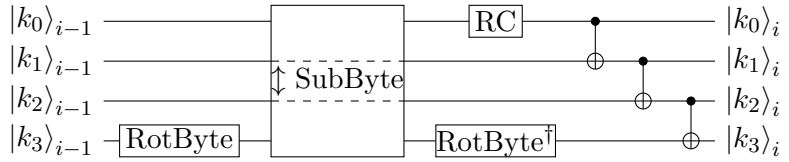
AddRoundKey performs a bitwise XOR of a round key to the internal AES state and can be realized with a parallel application of 128 CNOT gates, controlled on the round key qubits and targeted on the state qubits. Grassl *et al.* [GLRS16] and Langenberg *et al.* [LPS20] use the same approach.

2.4.5 KeyExpansion

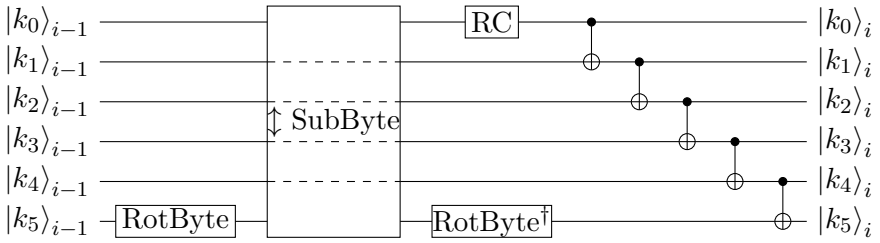
Key expansion is one of the two functions in AES using S-boxes, and hence is a source of T gates. Therefore, it might have a strong impact on the overall efficiency of the circuit. A simple implementation of KeyExpansion would allocate enough auxiliary qubits to store the full expanded key, including all round keys. We call this strategy *naive unrolling*. This is easy to implement with relatively low depth, but uses more qubits than necessary. The authors of [GLRS16] amortize this width cost by caching only those key bytes that require S-box evaluations. Instead, we minimize width by not requiring auxiliary qubits at all (other than those used internally by the S-box). At the same time, we reduce the depth in comparison with naive unrolling.

We now describe the design we opt for, which we call *in-place* KeyExpansion. Let $|k\rangle_0$ denote the AES key consisting of $N_k \in \{4, 6, 8\}$ words and let $|k\rangle_i$ be the i-th set of N_k consecutive round key words. The first such block $|k\rangle_1$ can be computed in-place as shown in Figure 2.8. The depicted circuits produce the i-th set of N_k round key words from the $(i-1)$-th set. Note that for AES-128 these sets correspond to the actual round keys as the key size is equal to the block size, while for AES-192 and AES-256 each $|k\rangle_i$ contains more words than needed for a round key. Let KE denote the operation mapping $|k\rangle_{i-1} \mapsto |k\rangle_i$. We write KE_j^l to denote the part of the operation KE that produces the words $j, \ldots, l$ of the new set. KE_j^l can be used as part of the round strategy that we will describe in Section 2.4.6, to only compute as many words of the round key as necessary at any given time, resulting in an overall narrower and shallower circuit.

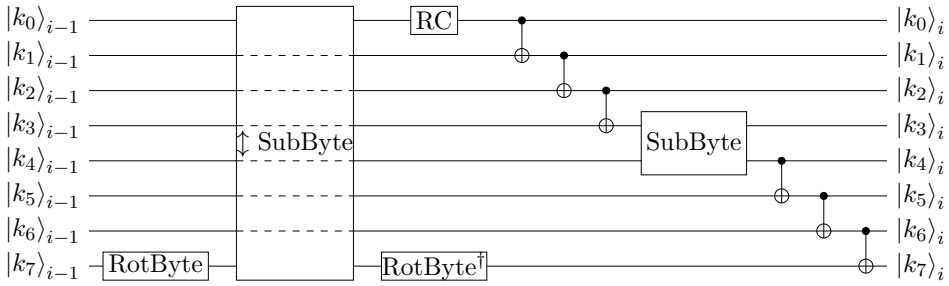
Remark 15. *In addition to improving the S-box circuit over [GLRS16], Langenberg* et al. *[LPS20, §4] demonstrate significant savings by reducing the number of qubits and the depth of key expansion. This is achieved by an improved scheduling of key expansion during AES encryption, namely by computing round key words only at the time they are required and uncomputing them early. While their method is based on the one in [GLRS16] using auxiliary qubits for the round keys, the approach we use works in place and reduces width and depth at the same time.*



(a) AES-128 in-place key expansion step producing the i-th round key.



(b) AES-192 in-place key expansion step producing the i-th set of 6 round key words.



(c) AES-256 in-place key expansion step producing the i-th set of 8 round key words.

Figure 2.8: In-place AES key expansion for AES-128, AES-192, and AES-256, deriving the i-th set of N_k round key words from the $(i-1)$-th set. Each $|k_j\rangle_i$ represents the j-th word of $|k\rangle_i$. SubByte takes the input state on the top wire, and returns the output on the bottom wire, while ↕ SubByte takes inputs on the bottom wire, and returns outputs on the top. Dashed lines indicate wires that are not used in the ↕ SubByte operation. RC is the round constant addition (labelled `Rcon` in [DR99, DR01]), implemented by applying X gates as appropriate.

operation	KE	#CNOT	#1qCliff	$\#T$	$\#M$	T-D	D	W
AES-128	in-place	291420	83656	54400	13600	120	2827	1785
AES-192	in-place	328490	92916	60928	15232	120	2976	2105
AES-256	in-place	403040	115102	75072	18768	126	3356	2425
AES-128	naive	293694	83084	54400	13600	132	2986	3065
AES-192	naive	331448	92944	60928	15232	132	3133	3577
AES-256	naive	406304	114974	75072	18768	138	3384	4089

Table 2.3: Size comparison for AES quantum circuits using in-place KeyExpansion vs naive unrolling. In both cases, an "in-place" MixColumn circuit is used. We notice that the difference in width between equivalent circuits corresponds to $4 \cdot 32 \cdot (Nr + 1) - 32 \cdot Nk$ qubits, where Nr (resp. Nk) is the number of AES rounds (resp. words in the AES key), see [DR99].

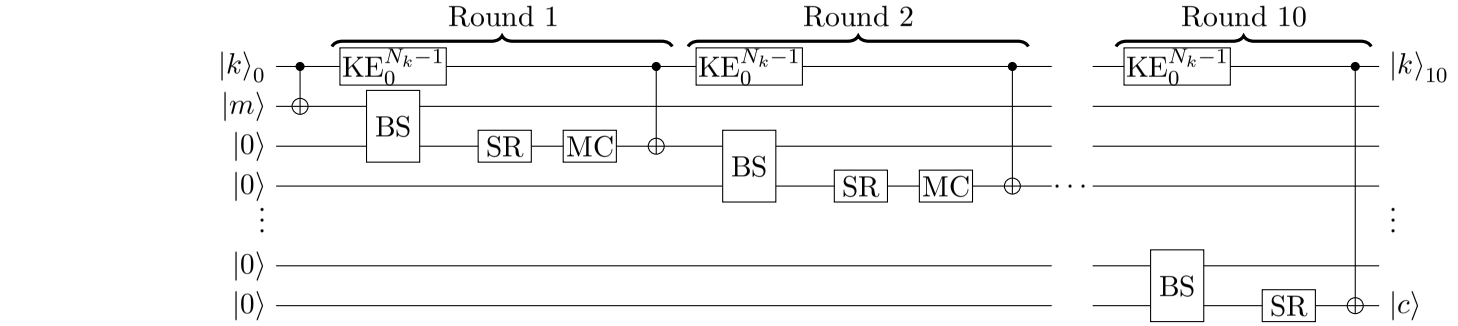
In-place KeyExpansion vs. naive unrolling. While in-place KeyExpansion clearly saves width by not requiring auxiliary qubits for the expansion, it may look as going against our design choice of minimizing depth. In particular, one may think that a naive design where a register of enough auxiliary qubits is allocated such that the whole key expansion can be performed before any rounds are run could save in depth, given that it does not need to handle any particular previous state on the qubits. In Table 2.3, we report numbers comparing the sizes of our AES circuits, with the only difference being the naive vs the in-place designs for KeyExpansion, showing that the latter is shallower (and of course narrower). The reason for this is that when using in-place Key expansion, we are able to perform the gates for the KE operations in parallel to the gates used in the round circuit. In particular, the S-box computations required to expand the key can be run in parallel to those executed on the state by ByteSub.

2.4.6 Round, FinalRound and full AES

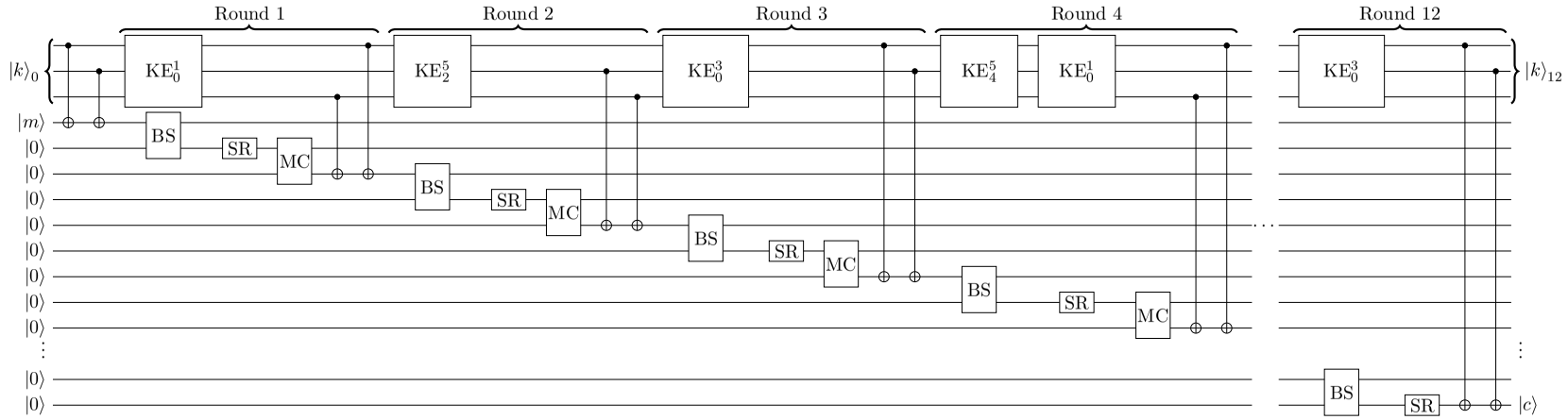
To encrypt a message block using AES-128 (resp. -192, -256), we initially XOR the input message with the first 4 words of the key, and then execute 10 (resp. 12, 14) rounds consisting of ByteSub, ShiftRow, MixColumn (except in the final round) and AddRoundKey.

The quantum circuits for AES we propose follow the same blueprint with the exception that key expansion is interleaved with the algorithm in such a way that the operations KE_j^l only produce the key words that are immediately required. The resulting circuits are shown in Figures 2.9 and 2.10. For formatting reasons, we omit the repeating round pattern, and only represent a subset of the full set of qubits used. In AES-128, each round is identical until round 9. In AES-192 rounds 5, 8 and 11 use the same KE call and order as round 2; rounds 6 and 9 do as round 3; rounds 7 and 10 do as round 4. In AES-256, rounds 4, 6, 8, 10, 12 (resp. 5, 7, 9, 11, 13) use the same KE call and order as round 2 (resp. 3). Cost estimates for the resulting AES encryption circuits are in Table 2.4. In contrast to [GLRS16] and [LPS20], we aim to reduce circuit depth, hence uncomputing of rounds is delayed until the output ciphertext is produced. For easier testability and modularity, the Round circuit is divided into two parts: a ForwardRound operator that computes the output state but does not clean auxiliary qubits, and its adjoint. For unit-testing Round in isolation, we compose ForwardRound with its adjoint operator. For testing AES, we first run all ForwardRound instances without auxiliary qubit cleaning, resulting in a similar ForwardAES operator, copy out the ciphertext, and then undo the ForwardAES operation.

Table 2.4 presents circuit size results for the AES circuit for both versions of MixColumn, the in-place implementation using a PLU decomposition as well as Maximov's out-of-place but lower depth circuit. We use both because each has advantages for different applications. The full depth corresponds to G_D as in Section 2.3.6 and Section 2.2.3, while width corresponds to G_W. While for AES-128 and AES-192, $\mathsf{G}_D\mathsf{G}_W$ is smaller for the in-place implementation, $\mathsf{G}_D^2\mathsf{G}_W$ is smaller for Maximov's circuit. Hence, Section 2.2.3 indicates Maximov's circuit gives a lower DW-cost under a depth restriction. If there is no depth restriction or if we only consider T-depth rather than depth, the in-place design has a lower DW-cost.

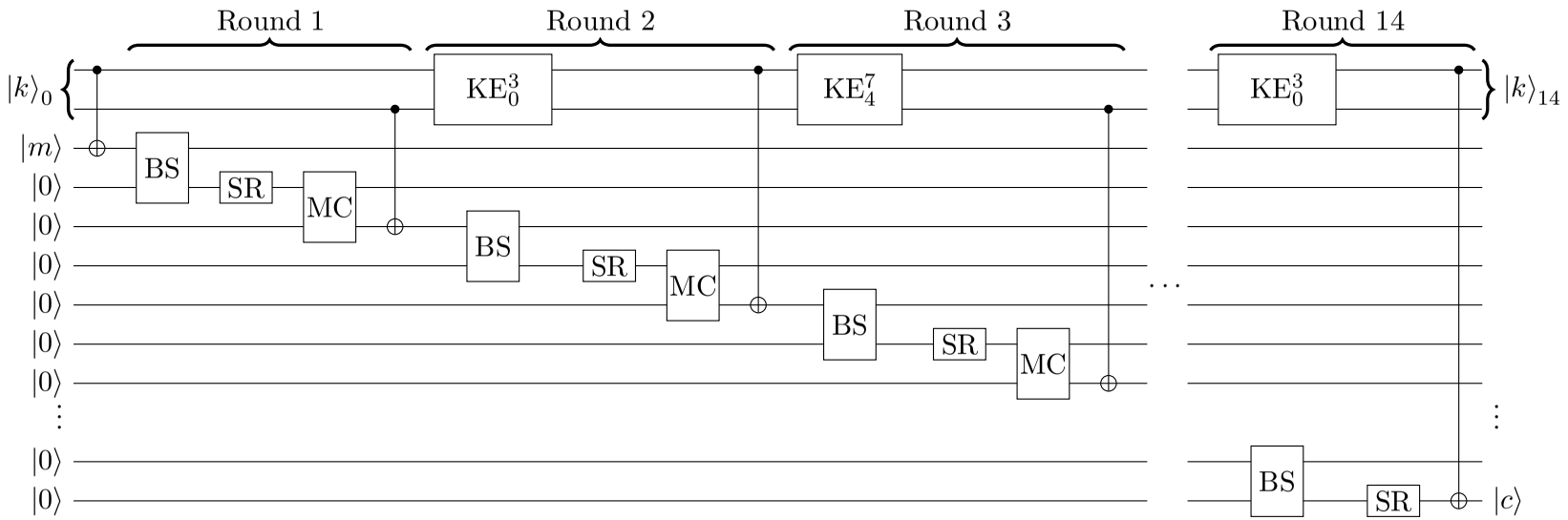


(a) AES-128 operation.



(b) AES-192 operation.

Figure 2.9: Circuit sketches for the AES-128 and -192 operation. Each wire under the $|k\rangle_0$ label represents 4 words for AES-128 and 2 words for -192. Other wires represent 4 words. CNOT gates represent parallel CNOT gates applied bitwise. BS stands for ByteSub, SR for ShiftRow and MC for MixColumn. In Figure 2.9a, MixColumn acts in-place, in Figure 2.9b it acts out-of-place.



(a) AES-256 operation.

Figure 2.10: Circuit sketch for the AES-256 operation. Each wire under the $|k\rangle_0$ label represents 4 words of the key. Each subsequent wire (initially labelled $|m\rangle$ and $|0\rangle$) represents 4 words. CNOT gates between word-sized wires should be read as multiple parallel CNOT gates applied bitwise. BS stands for ByteSub, SR for ShiftRow and MC for MixColumn. MixColumn uses an out-of-place implementation, such as Maximov's MixColumn linear program [Max19].

operation	MC	#CNOT	#1qCliff	$\#T$	$\#M$	T-D	D	W
AES-128	in-place	291420	83656	54400	13600	120	2827	1785
AES-192	in-place	328490	92916	60928	15232	120	2976	2105
AES-256	in-place	403040	115102	75072	18768	126	3356	2425
AES-128	[Max19]	293716	83208	54400	13600	120	2085	2937
AES-192	[Max19]	331462	92700	60928	15232	120	1878	3513
AES-256	[Max19]	406638	115018	75072	18768	126	1954	4089

Table 2.4: Circuit size estimates for the AES operator using the [BP12] S-box, for MixColumn design ("MC") either in-place or out-of-place [Max19]. The apparently inconsistent T-depth is discussed in § 2.4.7.

2.4.7 T-depth

Every round of AES (as implemented in Figures 2.9 and 2.10) computes at least one layer of S-boxes as part of ByteSub, which must later be uncomputed. We would thus expect the T-depth of n rounds of AES to be $2n$ times the T-depth of the S-box. Instead, Table 2.4 shows smaller depths. We find this effect when using either the AND circuit or the unit-testable CCNOT implementation. To test if this is a bug, we used a placeholder S-box circuit which has an arbitrary T-depth d for which the compiler cannot parallelise uses of the T gate (see Figure 2.11 for the design). This "dummy" AES design shows the expected T-depth of $2n \cdot d$. Thus we believe the Q# compiler found a non-trivial parallelisation between components of the S-box and the surrounding circuit. This provides a strong case for full explicit implementations of quantum cryptanalytic algorithms in Q# or other languages that allow automatic resource estimates and optimizations; in our case the T-depth of AES-256 is 25% less than naively expected. Unfortunately, Q# cannot yet generate full circuit diagrams for depth-optimized circuits, so we do not know exactly where the parallelisation takes place[15].

[15] `https://github.com/microsoft/Quantum/issues/462`.

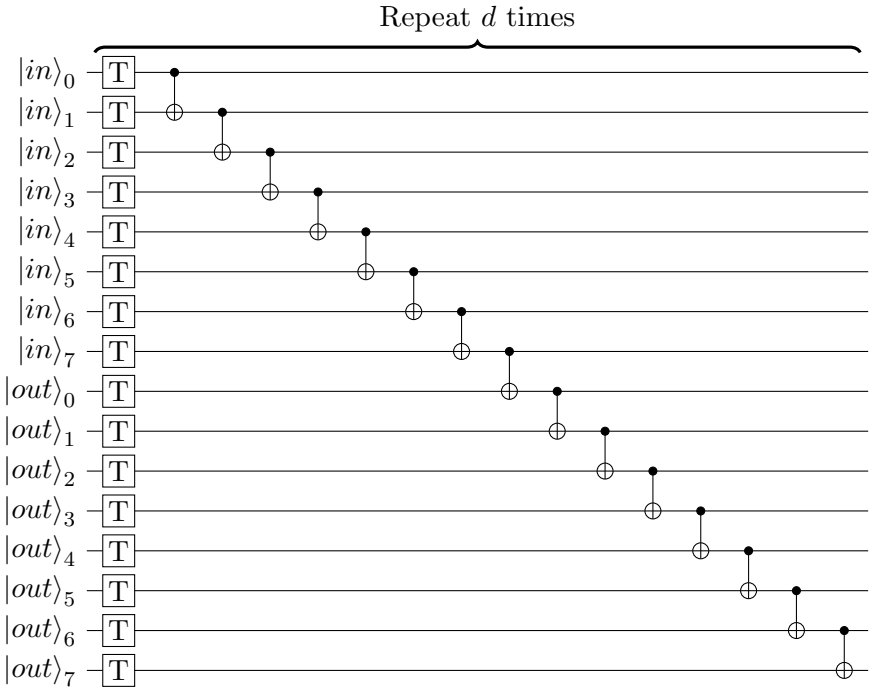


Figure 2.11: Dummy S-box design, that tries to forcefully avoid non-parallel calls to the S-box to be partially executed at the same time.

2.5 A quantum circuit for LowMC

LowMC [ARS+15, ARS+16] is a family of block ciphers aiming for having circuits with low multiplicative complexity. Originally designed to reduce the high cost of binary multiplication in the MPC and FHE scenarios, it has been adopted as a fundamental component by the Picnic signature scheme [CDG+17, ZCD+17]) proposed for standardisation as part of the NIST process for standardizing post-quantum cryptography.

To achieve low multiplicative complexity, LowMC uses an S-box layer of AND-depth 1, which contains a user-defined number of parallel 3-bit S-box computations. In general, any instantiation of LowMC comprises a specific number of rounds. Each round calls an S-box layer, an affine transformation, and a round key addition. Key-scheduling can either be precomputed or computed on the fly. We give in Algorithm 5 a pseudo-code description of the LowMC encryption algorithm.

Input: message m
Input: key k
$s \leftarrow m$
$rk_0 \leftarrow \text{KeyExpansion}(k, 0)$
$s \leftarrow \text{KeyAddition}(s, rk_0)$
for $i \leftarrow 1$ **to** total rounds **do**
  $s \leftarrow \text{S-boxLayer}(s)$
  $s \leftarrow \text{AffineLayer}(s)$
  $rk_i \leftarrow \text{KeyExpansion}(k, i)$
  $s \leftarrow \text{KeyAddition}(s, rk_i)$
$c \leftarrow s$
return c

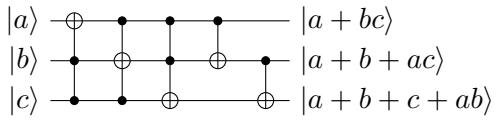
Algorithm 5: LowMC.

In this work, we study the original LowMC design. This results in a sub-optimal circuit, which can clearly be improved by porting the more recent design from [DKP+19] instead. Even for the original LowMC description, our work shows that the overhead from the cost of the Grover oracle is very small, in particular under the T-depth metric. Since LowMC could be standardized as a component of Picnic, we deem it appropriate to point out the differences in Grover oracle cost between different block ciphers and that generalization from AES requires caution.

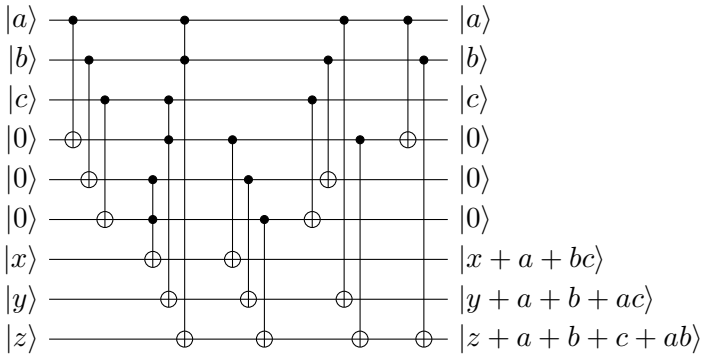
In this section we describe our Q# implementation of the LowMC instances used as part of Picnic. In particular, Picnic proposes three parameter sets, with $(\text{key size}, \text{block size}, \text{rounds}) \in \{(128, 128, 20), (192, 192, 30), (256, 256, 38)\}$, all with 10 parallel S-boxes per substitution layer.

2.5.1 S-box and S-boxLayer

The LowMC S-box can be naturally implemented using Toffoli (CCNOT) gates. In particular, a simple in-place implementation with depth 5 (T-depth 3) is shown in Figure 2.12, alongside a T-depth 1 out-of-place circuit, both of which were produced manually. Costs for both circuits can be found in Table 2.5. We use the CCNOT implementation with no measurements from [Sel13]. For LowMC inside of Picnic, the full S-boxLayer consists of 10 parallel S-boxes run on the 30 low order bits of the state.



(a) LowMC in-place S-box.



(b) LowMC T-depth 1 S-box.

Figure 2.12: Alternative quantum circuit designs for the LowMC S-box. The in-place design requires auxiliary qubits as part of the concrete CCNOT implementation.

operation	#CNOT	#1qCliff	$\#T$	$\#M$	T-D	D	W
In-place S-box	50	6	21	0	3	23	7
Shallow S-box	60	6	21	0	1	11	13

Table 2.5: Cost estimates for a single LowMC S-box circuit, following the two designs proposed in Figure 2.12. We note that Figure 2.12 does not display the concrete implementation of CCNOT.

operation	#CNOT	#1qCliff	$\#T$	$\#M$	T-D	D	W
AffineLayer L1 R1	8093	60	0	0	0	2365	128
AffineLayer L3 R1	18080	90	0	0	0	5301	192
AffineLayer L5 R1	32714	137	0	0	0	8603	256

Table 2.6: Costs for in-place circuits implementing the first round (R1) AffineLayer transformation for the three instantiations of LowMC used in Picnic.

2.5.2 LinearLayer, ConstantAddition and AffineLayer

AffineLayer is an affine transformation applied to the state at every round. It consists of a matrix multiplication (LinearLayer) and the addition of a constant vector (ConstantAddition). Both matrix and vector are different for every round and are predefined constants that are populated pseudo-randomly. ConstantAddition is implemented by applying X gates for entries of the vector equal to 1. In Picnic, for every round and every parameter set, all LinearLayer matrices are invertible (due to LowMC's specification requirements), and hence we use a PLU decomposition for matrix multiplication (Section 2.3.5). Cost estimates for the first round affine transformation in LowMC as used in Picnic are shown in Table 2.6.

2.5.3 KeyExpansion and KeyAddition

To generate the round keys rk_i, in each round i the LowMC key k is multiplied by a different key derivation pseudo-random matrix KM_i. For Picnic, each KM_i is invertible, so we compute rk_i from rk_{i-1} as $rk_i = KM_i \cdot KM_{i-1}^{-1} \cdot rk_{i-1}$. We compute this in-place using a PLU decomposition of $KM_i \cdot KM_{i-1}^{-1}$. This saves matrix multiplications and qubits compared to computing rk_i directly from k. We call this operation KeyExpansion. KeyAddition is equivalent to AddRoundKey in AES, and is implemented the same way. Cost estimates for the first round key expansion in LowMC as used in Picnic can be found in Table 2.7.

operation	#CNOT	#1qCliff	$\#T$	$\#M$	T-D	D	W
KeyExp. L1 R1	8104	0	0	0	0	2438	128
KeyExp. L3 R1	18242	0	0	0	0	4896	192
KeyExp. L5 R1	32525	0	0	0	0	9358	256

Table 2.7: Costs for in-place circuits implementing the first round (R1) Key-Expansion (KeyExp.) operation for the three instantiations of LowMC used in Picnic.

operation	#CNOT	#1qCliff	$\#T$	$\#M$	T-D	D	W
LowMC L1	689944	4932	8400	0	40	98699	991
LowMC L3	2271870	9398	12600	0	60	319317	1483
LowMC L5	5070324	14274	15960	0	76	693471	1915

Table 2.8: Costs for the full encryption circuit for LowMC as used in Picnic.

2.5.4 Round function and full LowMC

The LowMC round function sequentially applies S-boxLayer, AffineLayer and KeyAddition to the state. Our implementation also runs KeyExpansion before AffineLayer. For a full LowMC encryption, we first add the LowMC key k to the message to produce the initial state, then run the specified number of rounds on it. Costs of the resulting encryption circuits are reported in Table 2.8.

Remark 16. *Contrary to what was noticed in the case of AES in Section 2.4.7, the T-depth reported in Table 2.8 is proportional to the number of rounds used in the LowMC instances implemented. This may be due to the simpler S-box and round structure in LowMC being less amenable to compiler optimizations.*

2.6 Grover oracles and resource estimates for key search

Equipped with Q# implementations of the AES and LowMC encryption circuits, this section describes the implementation of full Grover oracles for both block

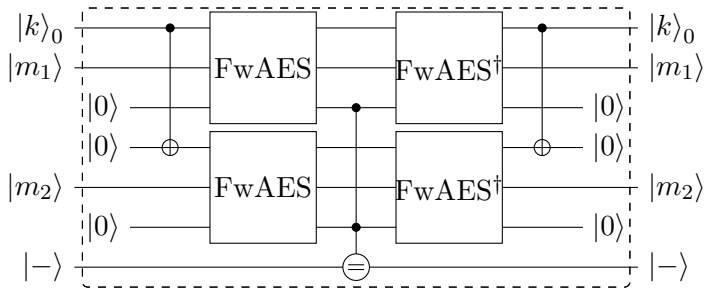


Figure 2.13: Grover oracle construction from AES using two message-ciphertext pairs. FwAES represents the ForwardAES operator described in § 2.4.6. The middle operator "=" compares the output of AES with the provided ciphertexts and flips the target qubit if they are equal.

ciphers. Eventually, based on the cost estimates obtained automatically from the Q# implementations of the oracles, we provide quantum resource estimates for full key search attacks via Grover's algorithm. Beyond comparing to previous work, our emphasis is on evaluating algorithms that respect a total depth limit, for which we consider NIST's values for `MAXDEPTH` from [Nat16]. This means we *must* parallelise. We use inner parallelisation via splitting up the search space, as described in Section 2.2.3.

2.6.1 Grover oracles

As discussed in Section 2.2.2 and Section 2.2.3, we must determine the parameter r, the number of known plaintext-ciphertext pairs that are required for a successful key-recovery attack. The Grover oracle encrypts r plaintext blocks under the same candidate key and computes a Boolean value that encodes whether all r resulting ciphertext blocks match the given values. A circuit for the block cipher allows us to build an oracle for any r by simply fanning out the key qubits to the r instances and running the r block cipher circuits in parallel. Then a comparison operation with the classical ciphertexts conditionally flips the result qubit and the r encryptions are uncomputed. Figure 2.13 shows the construction for AES and $r = 2$, using the ForwardAES operation from Section 2.4.6.

The required number of plaintext-ciphertext blocks. In the case of Grover key search with no constraint on circuit depth, the explicit computation of the probabilities in Equation (2.1) shows that using $r = 2$ (resp. 2, 3) for AES-128 (resp. -192, -256) guarantees a unique key with overwhelming probability. The probabilities that there are no spurious keys are $1 - \epsilon$, where $\epsilon < 2^{-128}$, 2^{-64}, and 2^{-128}, respectively. Grassl *et al.* [GLRS16, § 3.1] used $r = 3$, $r = 4$ and $r = 5$, respectively. Hence, their values are too large and the Grover oracle can work correctly with fewer full AES evaluations.

If one is content with a success probability lower than 1, it suffices to use $r = \lceil k/n \rceil$ blocks of plaintext-ciphertext pairs. In this case, it is enough to use $r = 1$, 2, and 3 for AES-128, -192, -256, respectively. Langenberg *et al.* [LPS20] also propose these values. As an example, if we use $r = 1$ for AES-128, the probability of not having spurious keys is $1/e \approx 0.368$, which could be a high enough chance for a successful attack in certain scenarios, e.g., when there is a strict limit on the width of the attack circuit. Furthermore, when a large number of parallel machines are used in an instance of the attack, as discussed in Section 2.2.3, even the value $r = 1$ can be enough in order to guarantee with high probability that the relevant subset of the key space contains the correct key as a unique solution.

The LowMC parameter sets we consider here all have $k = n$. Therefore, $r = 2$ plaintext-ciphertext pairs are enough for all three sets ($k \in \{128, 192, 256\}$). Then, the probability that the key is unique is $1 - \epsilon$, where $\epsilon < 2^{-k}$, i.e. this probability is negligibly close to 1. With high parallelisation, $r = 1$ is sufficient for a success probability very close to 1.

Grover oracle cost for AES. Table 2.9 shows the resources needed for the full AES Grover oracle for the relevant values of $r \in \{1, 2, 3\}$. Even without parallelisation, more than 2 pairs are never required for AES-128 and AES-192. The same holds for 3 or more pairs for AES-256.

Grover oracle cost for LowMC. The resources required for our implementation of the full LowMC Grover oracle for the relevant values of $r \in \{1, 2\}$

operation	MC	r	#CNOT	#1qCliff	$\#T$	$\#M$	T-D	D	W
AES-128	IP	1	292213	84228	54908	13727	121	2830	1665
AES-192	IP	1	329779	94480	61436	15359	120	2986	1985
AES-256	IP	1	403895	115798	75580	18895	126	3346	2305
AES-128	IP	2	584567	168216	109820	27455	121	2830	3329
AES-192	IP	2	659623	188312	122876	30719	120	2994	3969
AES-256	IP	2	808371	231724	151164	37791	126	3343	4609
AES-256	IP	3	1212773	347502	226748	56687	126	3348	6913
AES-128	M	1	294453	83668	54908	13727	121	2074	2817
AES-192	M	1	332765	94292	61436	15359	120	1884	3393
AES-256	M	1	407401	115530	75580	18895	126	1943	3969
AES-128	M	2	589879	168760	109820	27455	121	2093	5633
AES-192	M	2	665843	188432	122876	30719	120	1886	6785
AES-256	M	2	815639	231700	151164	37791	126	1953	7937
AES-256	M	3	1223521	347158	226748	56687	126	1957	11905

Table 2.9: Costs for the AES Grover oracle operator for $r = 1$, 2 and 3 plaintext-ciphertext pairs. "MC" is the MixColumn design, either in-place ("IP") or Maximov's [Max19] ("M").

are shown in Table 2.10. No setting needs more than $r = 2$ plaintext-ciphertext pairs. Note that although the circuits in Section 2.5.4 for computing the LowMC cipher do not use AND gates, the comparison operator used to construct the Grover oracle does. This explains why in Table 2.10 a small number of measurements is reported.

2.6.2 Cost estimates for block cipher key search

Using the cost estimates for the AES and LowMC Grover oracles from Section 2.6.1, this section provides cost estimates for full key search attacks on both block ciphers. For the sake of a direct comparison to the previous results in [GLRS16] and [LPS20], we first ignore any limit on the depth and present the same setting as in these works. Then, we provide cost estimates with imposed depth limits and the consequential parallelisation requirements.

operation	r	#CNOT	#1qCliff	$\#T$	$\#M$	T-D	D	W
LowMC L1	1	690959	5948	8908	127	41	98705	1585
LowMC L3	1	2273401	10934	13364	191	61	319323	2377
LowMC L5	1	5072343	16209	16980	372	77	693477	3049
LowMC L1	2	1382179	11896	17820	255	41	98711	3169
LowMC L3	2	4547147	21780	26732	383	61	319329	4753
LowMC L5	2	10145281	32567	33964	783	77	693483	6097

Table 2.10: Cost estimates for the LowMC Grover oracle operator for $r = 1$ and 2 plaintext-ciphertext pairs. LowMC parameter sets are as used in Picnic.

Comparison to previous work. Table 2.11 shows cost estimates for a full run of Grover's algorithm when using $\lfloor \frac{\pi}{4} 2^{k/2} \rfloor$ iterations of the AES Grover operator without parallelisation. We only take into account the costs imposed by the oracle operator U_f (in the notation of Section 1.5.1) and ignore the costs of the operator $U_\psi = 2\,|\psi\rangle\langle\psi| - I$. If the number of plaintext-ciphertext pairs ensures a unique key, this number of operations maximizes the success probability p_{succ} to be negligibly close to 1. For smaller values of r such as those proposed in [LPS20], the success probability is given by the probability that the key is unique[16].

The G-cost is the total number of gates, which is the sum of the first three columns in the table, corresponding to the numbers of 1-qubit Clifford and CNOT gates (both under "Clifford" to more easily compare with other works which similarly collect both numbers), T gates and measurements. Table 2.11 shows that the G-cost is always better in our work when comparing values for the same AES instance and the same value for r and the same holds for the DW-cost.

Table 2.12 shows cost estimates for LowMC in the same setting. Despite LowMC's lower multiplicative complexity and a relatively lower number of T gates, the large number of CNOT gates leads to overall higher G-cost and DW-cost than AES, as we count all gates.

[16]Technically, this results in an approximate lower bound to the success probability.

Grassl *et al.* [GLRS16]										
		$\log_2$						$\log_2$		
scheme	r	#Clifford	$\#T$	$\#M$	T-D	D	W	G-cost	DW-cost	$\approx p_s$
AES-128	3	86.63	86.25	—	80.08	81.21	2 953	87.45	92.74	1
AES-192	4	119.23	118.86	—	112.28	113.41	4 449	120.06	125.53	1
AES-256	5	151.87	151.50	—	144.53	145.65	6 681	152.70	158.36	1
extrapolation of Grassl *et al.* [GLRS16] to lower r										
AES-128	1	85.04	84.67	—	80.08	81.21	984	85.87	91.15	$1/e$
AES-192	2	118.23	117.86	—	112.28	113.41	2 224	119.06	124.53	1
AES-256	2	150.55	150.18	—	144.53	145.65	2 672	151.38	157.03	$1/e$
Langenberg *et al.* [LPS20]										
AES-128	1	82.55	81.56	—	77.53	79.48	865	83.14	89.23	$1/e$
AES-192	2	115.77	114.75	—	109.33	111.30	1 793	116.34	122.11	1
AES-256	2	148.04	147.03	—	141.73	143.69	2 465	148.62	154.96	$1/e$
this work (with "in-place" MixColumn)										
AES-128	1	82.17	79.40	77.40	70.57	75.12	1665	82.42	85.82	$1/e$
AES-128	2	83.17	80.40	78.40	70.57	75.12	3329	83.42	86.82	1
AES-192	2	115.35	112.56	110.56	102.56	107.20	3969	115.59	119.15	1
AES-256	2	147.64	144.86	142.86	134.63	139.36	4609	147.88	151.53	$1/e$
AES-256	3	148.22	145.44	143.44	134.63	139.36	6913	148.47	152.12	1
this work (with "in-place" MixColumn), using Grassl *et al.* [GLRS16] values for r										
AES-128	3	83.76	80.98	78.98	70.57	75.12	4993	84.00	87.40	1
AES-192	4	116.35	113.56	111.56	102.56	107.20	7937	116.59	120.15	1
AES-256	5	148.96	146.18	144.18	134.63	139.36	11521	149.20	152.85	1

Table 2.11: Comparison of cost estimates for Grover's algorithm with $\lfloor \frac{\pi}{4} 2^{k/2} \rfloor$ AES oracle iterations for attacks with high success probability, disregarding `MAXDEPTH`. CNOT and 1-qubit Clifford gate counts are added to allow easier comparison to the previous work from [GLRS16, LPS20], who report both kinds of gates under "Clifford". [LPS20] uses the S-box design from [BP10]. In this table we only use the in-place MixColumn design (see § 2.4.3). The circuit sizes for AES-128 (resp. -192, -256) in the second block have been extrapolated from Grassl *et al.* by multiplying gate counts and circuit width by 1/3 (resp. 1/2, 2/5), while keeping depth values intact. p_s reports the approximate success probability.

		$\log_2$							$\log_2$		
scheme	r	# CNOT	#1qCliff	$\#T$	$\#M$	T-D	D	W	G-cost	DW-cost	$\approx p_s$
LowMC L1	1	83.05	76.19	76.77	70.64	69.01	80.24	1585	83.08	90.87	$1/e$
LowMC L3	1	116.77	109.07	109.36	103.23	101.58	113.94	2377	116.78	125.15	$1/e$
LowMC L5	1	149.93	141.63	141.70	136.19	133.92	147.06	3049	149.93	158.63	$1/e$
LowMC L1	2	84.05	77.19	77.77	71.65	69.01	80.24	3169	84.08	91.87	1
LowMC L3	2	117.77	110.06	110.36	104.23	101.58	113.94	4753	117.78	126.15	1
LowMC L5	2	150.93	142.64	142.70	137.26	133.92	147.06	6097	150.93	159.63	1

Table 2.12: Cost estimates for Grover's algorithm with $\lfloor \frac{\pi}{4} 2^{k/2} \rfloor$ LowMC oracle iterations for attacks with high success probability, without a depth restriction.

Cost estimates under a depth limit. Tables 2.14 and 2.15 show cost estimates for running Grover's algorithm against AES and LowMC under a given depth limit. This restriction is proposed in the NIST call for proposals for standardisation of post-quantum cryptography [Nat16]. We use the notation and example values for `MAXDEPTH` from the call. Imposing a depth limit forces the parallelisation of Grover's algorithm, which we assume uses inner parallelisation, see Section 2.2.3.

The values in the table follow Section 2.3.6. Given cost estimates G_G, G_D and G_W for the oracle circuit, we determine the maximal number of Grover iterations that can be carried out within the `MAXDEPTH` limit. Then the required number S of parallel instances is computed via Equation (2.8) and the G-cost and DW-cost follow from Equations (2.9) and (2.10). The number r of plaintext-ciphertext pairs is the minimal value such that the probability SKP for having spurious keys in the subset of the key space that holds the target key is less than 2^{-20}.

The impact of imposing a depth limit on the key search algorithm can directly be seen by comparing, for example Table 2.14 with Table 2.11 in the case of AES. Key search against AES-128 without depth limit has a G-cost of $2^{83.42}$ gates and a DW-cost of $2^{86.82}$ qubit-cycles. Now, setting $\texttt{MAXDEPTH} = 2^{40}$ increases both the G-cost and the DW-cost by a factor of roughly 2^{34} to $2^{117.09}$ gates and $2^{120.80}$ qubit-cycles. For $\texttt{MAXDEPTH} = 2^{64}$, the increase is by a factor of roughly 2^{10}. We note that for $\texttt{MAXDEPTH} = 2^{96}$, key search on AES-128 does not require any parallelisation.

Implications for post-quantum security categories. The security strength categories 1, 3 and 5 in the NIST call for proposals [Nat16] are defined by the resources needed for key search on AES-128, AES-192 and AES-256, respectively. For a cryptographic scheme to satisfy the security requirement at a given level, the best known attack must take at least as many resources as key search against the corresponding AES instance.

As guidance, NIST provides a table with gate cost estimates via a formula depending on the depth bound `MAXDEPTH`. This formula is deduced as follows: assume that non-parallel Grover search requires a depth of $D = x \cdot \texttt{MAXDEPTH}$ for some $x \geq 1$ and that the circuit has G gates. Then, about x^2 machines are needed, each running for a fraction $1/x$ of the non-parallel runtime and using roughly G/x gates, in order for the quantum attack to fit within the depth budget given by `MAXDEPTH` while attaining the same attack success probability. Hence, the total gate count for a parallelised Grover search is roughly $(G/x) \cdot x^2 = G \cdot D/\texttt{MAXDEPTH}$. The cost formula reported in the NIST table (also provided in Table 2.13 for reference) is deduced by using the values for G-cost and depth D from Grassl *et al.* [GLRS16].

However, the above formula does not take into account that parallelisation often allows us to reduce the number of required plaintext-ciphertext pairs, resulting in a G-cost reduction for search in each parallel Grover instance by a factor larger than x. Note also that [Nat16, Footnote 5] mentions that using the formula for very small values of x (corresponding to very large values of `MAXDEPTH` such that $x = D/\texttt{MAXDEPTH} < 1$ and that no parallelisation is required) underestimates the quantum security of AES. This is the case for AES-128 with $\texttt{MAXDEPTH} = 2^{96}$.

In Table 2.13, we compare NIST's numbers with our gate counts for parallel Grover search. Our results for each specific setting incorporate the reduction of plaintext-ciphertext pairs through parallelisation, provide the correct cost if parallelisation is not necessary and use improved circuit designs. The table shows that for most situations, AES is less quantum secure than the NIST estimates predict. For each category, we provide a very rough approximation

formula that could be used to replace NIST's formula. We observe a consistent reduction in G-cost for quantum key search by 11-13 bits.

Since NIST clearly defines its security categories 1, 3 and 5 based on the computational resources required for key search on AES, the explicit required gate counts should be lowered to account for any cheaper attacks (in the same computational model), such as our. This would mean that it is now easier for submitters to claim equivalent security, with the exception of category 1 with $\texttt{MAXDEPTH} = 2^{96}$. A possible consequence of our work is that some of the NIST submissions might profit from slightly tweaking certain parameter sets to allow more efficient implementations, while at the same time satisfying the (now weaker) requirements for their intended security category.

Remark 17. *The G-cost results in Table 2.15 show that key recovery against the LowMC instances we implemented requires at least as many gates as key recovery against AES with the same key size. If NIST replaces its explicit gate cost estimates for AES with the ones in this work, these LowMC instances meet the post-quantum security requirements as defined in the NIST call [Nat16]. On the other hand, the same results show that they do not meet the explicit gate count requirements for the original NIST security categories. For example, LowMC L1 can be broken with an attack having G-cost* $2^{123.32}$ *when* $\texttt{MAXDEPTH} = 2^{40}$*, while the original bound in category 1 requires a scheme to not be broken by an attack using less than* 2^{130} *gates. In all settings considered here, a LowMC key can be found with a slightly smaller G-cost than NIST's original estimates for AES.*[17] *The margin is relatively small. However, we cannot finalize conclusions about the relative security of LowMC and AES until quantum circuits for LowMC are optimized as much as the ones for AES.*

2.7 Conclusions

This chapter's main focus was on exploring the setting proposed by NIST where quantum attacks are limited by a total bound on the depth of quantum circuits.

[17] Except for the case of LowMC L1 in $\texttt{MAXDEPTH} = 2^{96}$, where the criterium for being in Category 1 appears to be too relaxed due to NIST underestimating the security of AES-128.

NIST Security Category	source	G-cost for MAXDEPTH ($\log_2$) 2^{40}	2^{64}	2^{96}	approximation
1 AES-128	[Nat16]	130.0	106.0	74.0	2^{170}/MAXDEPTH
	this work	117.1	93.1	*83.4	$\approx 2^{157}$/MAXDEPTH
3 AES-192	[Nat16]	193.0	169.0	137.0	2^{233}/MAXDEPTH
	this work	181.1	157.1	126.1	$\approx 2^{221}$/MAXDEPTH
5 AES-256	[Nat16]	258.0	234.0	202.0	2^{298}/MAXDEPTH
	this work	245.5	221.5	190.5	$\approx 2^{285}$/MAXDEPTH

Table 2.13: Comparison of our cost estimate results with NIST's approximations based on Grassl *et al.* [GLRS16]. The *approximation* column displays NIST's formula from [Nat16] and a rough approximation to replace the NIST formula based on our results. Under MAXDEPTH $= 2^{96}$, AES-128 is a special case as the attack does not require any parallelisation and the approximation underestimates its cost.

Previous works [GLRS16, ASAM18, LPS20] aim to minimize cost under a trade-off between circuit depth and a limit on the total number of qubits needed, say a hypothetical bound MAXWIDTH. Depth limits are not discussed when choosing a Grover strategy. Since it is somewhat unclear what exact characteristics and features a future scalable quantum computer might have, quantum circuit and Grover strategy optimization with the goal of minimizing different cost metrics under different constraints than MAXDEPTH could be an interesting avenue for future research.

We have studied key search problems for a single target. In classical cryptanalysis, multi-target attacks have to be taken into account for assessing the security of cryptographic systems. We leave the exploration of estimating the cost of quantum multi-target attacks, for example using the algorithm by Banegas and Bernstein [BB17] under MAXDEPTH (or alternative regimes), as future work.

Further, implementing quantum circuits for cryptanalysis in Q# or another quantum programming language for concrete cost estimation could be worthwhile to increase confidence in the security of proposed post-quantum schemes. For example, quantum lattice sieving and enumeration appear to be prime candidates.

			$\log_2$						
scheme	MC	r	`MD`	S	SKP	D	W	G-cost	DW-cost
AES-128	M	1	40	69.34	−69.34	40.00	80.80	117.09	120.80
AES-192	M	1	40	133.06	−69.06	40.00	144.79	181.13	184.79
AES-256	M	1	40	197.15	−69.15	40.00	209.11	245.46	249.11
AES-128	M	1	64	21.34	−21.34	64.00	32.80	93.09	96.80
AES-192	M	1	64	85.06	−21.06	64.00	96.79	157.13	160.79
AES-256	M	1	64	149.15	−21.15	64.00	161.11	221.46	225.11
AES-128	IP	2	96	—	—	75.12	11.70	83.42	86.82
AES-192	M	2	96	21.07	—	96.00	33.79	126.13	129.79
AES-256	M	2	96	85.17	−85.17	96.00	98.12	190.47	194.12

(a) The depth cost metric is the full depth D.

			$\log_2$						
scheme	MC	r	`MD`	S	SKP	T-D	W	G-cost	T-DW-cost
AES-128	IP	1	40	61.14	−61.14	40.00	71.84	112.99	111.84
AES-192	IP	1	40	125.12	−61.12	40.00	136.07	177.14	176.07
AES-256	IP	1	40	189.26	−61.26	40.00	200.43	241.51	240.43
AES-128	IP	2	64	13.14	—	64.00	24.84	89.99	88.84
AES-192	IP	2	64	77.12	—	64.00	89.07	154.14	153.07
AES-256	IP	2	64	141.26	−141.26	64.00	153.43	218.51	217.43
AES-128	IP	2	96	—	—	70.57	11.70	83.42	82.27
AES-192	IP	2	96	13.12	—	96.00	25.07	122.14	121.07
AES-256	IP	2	96	77.26	−77.26	96.00	89.43	186.51	185.43

(b) The depth cost metric is the T-depth T-D only.

Table 2.14: Cost estimates for parallel Grover key search against AES under a depth limit `MAXDEPTH` with *inner* parallelisation (see § 2.2.3). MC is the MixColumn circuit used (in-place or Maximov's [Max19]), r is the number of plaintext-ciphertext pairs used in the Grover oracle, `MD` is `MAXDEPTH`, S is the number of subsets into which the key space is divided, SKP is the probability that spurious keys are present in the subset holding the target key, W is the qubit width of the full circuit, D the full depth, T-D the T-depth, DW-cost uses the full depth and T-DW-cost the T-depth. After the Grover search is completed, each of the S measured candidate keys is classically checked against 2 (resp. 2, 3) plaintext-ciphertext pairs for AES-128 (resp. -192, -256).

		$\log_2$						
scheme	r	MD	S	SKP	D	W	G-cost	DW-cost
LowMC L1	1	40	80.48	−80.48	40.00	91.11	123.32	131.11
LowMC L3	1	40	147.87	−147.87	40.00	159.09	190.72	199.09
LowMC L5	1	40	214.11	−214.11	40.00	225.68	256.99	265.68
LowMC L1	1	64	32.48	−32.48	64.00	43.11	99.32	107.11
LowMC L3	1	64	99.87	−99.87	64.00	111.09	166.72	175.09
LowMC L5	1	64	166.11	−166.11	64.00	177.68	232.99	241.68
LowMC L1	2	96	—	—	80.24	11.63	84.08	91.87
LowMC L3	1	96	35.87	−35.87	96.00	47.09	134.72	143.09
LowMC L5	1	96	102.11	−102.11	96.00	113.68	200.99	209.68

(a) The depth cost metric is the full depth D.

		$\log_2$						
scheme	r	MD	S	SKP	T-D	W	G-cost	T-DW-cost
LowMC L1	1	40	58.02	−58.02	40.00	68.65	112.09	108.65
LowMC L3	1	40	123.16	−123.16	40.00	134.38	178.37	174.38
LowMC L5	1	40	187.84	−187.84	40.00	199.41	243.85	239.41
LowMC L1	2	64	10.02	—	64.00	21.65	89.09	85.65
LowMC L3	1	64	75.16	−75.16	64.00	86.38	154.37	150.38
LowMC L5	1	64	139.84	−139.84	64.00	151.41	219.85	215.41
LowMC L1	2	96	—	—	69.01	11.63	84.08	80.64
LowMC L3	2	96	11.16	—	96.00	23.38	123.37	119.38
LowMC L5	1	96	75.84	−75.84	96.00	87.41	187.85	183.41

(b) The depth cost metric is the T-depth T-D only.

Table 2.15: Cost estimates for parallel Grover key search against LowMC under a depth limit MAXDEPTH with *inner* parallelisation (see § 2.2.3). r is the number of plaintext-ciphertext pairs used in the Grover oracle, MD is MAXDEPTH, S is the number of subsets into which the key space is divided, SKP is the probability that spurious keys are present in the subset holding the target key, W is the qubit width of the full circuit, D the full depth, T-D the T-depth, DW-cost uses the full depth and T-DW-cost the T-depth. After the Grover search is completed, each of the S measured candidate keys is classically checked against 2 plaintext-ciphertext pairs.

2.7.1 Developments since publication

Since the publication of this chapter as [JNRV20], a few papers covering related topics have been made available that we would like to mention here. In the space of Grover cost estimates, some works have used a similar methodology to ours to estimate attacks against other symmetric primitives. In particular, [AMM20b, CS20, Sch20, AMM+20c] looked at the SIMON, ARIA, Gimli and FSR-based constructions respectively, providing implementations and cost estimates in Qiskit [AAA+19], while [JCKS20, JCK+20] looked at SPECK and at various Korean block ciphers respectively, with implementations and estimates using ProjectQ [SHT18]. [LY20, AMM20a] combined a similar methodology to compare Grover key search on SIMON and SPECK to quantum-aided differential cryptanalysis (with the latter paper providing an implementation in Qiskit). In the space of Grover search against AES specifically, [ZWS+20] introduced a new S-box design requiring fewer qubits, [CLLc20] investigates depth–width trade-offs for the S-box circuit, [DP21] implements the Search With Two Oracles technique from [DP19] in Q#. [HJN+20, BJ20] focused on Shor's and Simon's algorithms against elliptic curves and other symmetric primitives respectively, while still providing Q# implementations of the attacked primitives. Finally, [AGPS20] designed Nearest-Neighbour Search quantum circuits to investigate the crossover point between classical and quantum lattice sieving in the depth-unbounded setting.

Acknowledgements. We thank Chris Granade and Bettina Heim for their help with the Q# language and compiler, Mathias Soeken and Thomas Häner for general discussions on optimizing quantum circuits and Q#, Mathias Soeken for providing the quantum AND gate circuit we use, and Daniel Kales and Greg Zaverucha for their input on Picnic and LowMC.

Chapter 3

On the Expected Cost of Solving uSVP

Contents

Reducing the Learning with Errors problem (LWE) to the unique Shortest Vector Problem (uSVP) and then applying lattice reduction is a commonly relied-upon strategy for estimating the cost of solving LWE-based constructions. In the literature, two different conditions are formulated under which this strategy is successful. One going back to Gama & Nguyen's work on predicting lattice reduction (Eurocrypt 2008) and the other outlined by Alkim et al. (USENIX 2016). Since these two estimates predict significantly different costs for solving LWE parameter sets from the literature, we revisit the uSVP strategy. We present empirical evidence from lattice reduction experiments exhibiting a behaviour in line with the latter estimate. However, we also observe that in some situations lattice reduction behaves somewhat better than expected from

Alkim et al.'s work and explain this behaviour under standard assumptions. Finally, we show that the security estimates of some LWE-based constructions from the literature need to be revised and give refined expected solving costs.

3.1 Motivation

The *Learning with Errors* problem (LWE) has attained a central role in cryptography as a key hard problem for building quantum-safe cryptographic constructions, from public key encryption [Reg05, LP11, ADPS16] to obfuscation of some families of circuits [BVWW16].

Recalling from Section 1.4.2, LWE asks to recover a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$, given a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and a vector $\mathbf{b} \in \mathbb{Z}_q^m$ such that $\mathbf{As} + \mathbf{e} = \mathbf{b} \bmod q$ for a short error vector $\mathbf{e} \in \mathbb{Z}_q^m$ sampled coordinate-wise from an error distribution χ. The decision variant of LWE asks to distinguish between an LWE instance $(\mathbf{A}, \mathbf{b})$ and uniformly random $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$. To assess the security provided by a given set of parameters n, χ, q, two strategies are typically considered: the *dual* strategy finds short vectors in the lattice

$$\left\{ \mathbf{x} \in \mathbb{Z}_q^m \mid \mathbf{x}\,\mathbf{A} \equiv \mathbf{0} \bmod q \right\},$$

i.e. it solves the *Short Integer Solutions* problem (SIS). Given such a short vector $\mathbf{v}$, we can decide if an instance is LWE by computing $\langle \mathbf{v}, \mathbf{b} \rangle = \langle \mathbf{v}, \mathbf{e} \rangle \bmod q$ which is short whenever $\mathbf{v}$ and $\mathbf{e}$ are sufficiently short [MR09]. This strategy was revisited for small, sparse secret instances of LWE [Alb17]. The *primal* strategy finds the closest vector to $\mathbf{b}$ in the integral column span of $\mathbf{A}$ mod q [LP11], i.e. it solves the corresponding *Bounded Distance Decoding* problem (BDD) directly. Writing $[\mathbf{I}_n | \mathbf{A}']$ for the reduced row echelon form of $\mathbf{A}^t \in \mathbb{Z}_q^{n \times m}$ (with high probability and after appropriate permutation of columns), this task can be reformulated as solving the *unique Shortest Vector Problem* (uSVP) in the $m+1$ dimensional q-ary lattice

$$\Lambda = \mathbb{Z}^{m+1} \cdot \begin{pmatrix} \mathbf{0} & q\,\mathbf{I}_{m-n} & 0 \\ \mathbf{I}_n & \mathbf{A}' & 0 \\ \text{—}\mathbf{b}^t\text{—} & & c \end{pmatrix} \tag{3.1}$$

by Kannan's embedding [Kan87] with embedding coefficient c.[1] Indeed, BDD and uSVP are polynomial-time equivalent for small approximation factors [LM09]. The lattice Λ has volume $c \cdot q^{m-n}$ and contains a vector of norm $\sqrt{\|\mathbf{e}\|^2 + c^2}$ which is unusually short, i.e. the gap between the first and second lattice minimum $\lambda_2(\Lambda)/\lambda_1(\Lambda)$ is large.

Alternatively, if the secret vector $\mathbf{s}$ is also short, there is a second established embedding reducing LWE to uSVP (cf. (3.7)). When the LWE instance under consideration is in *normal form*, i.e. the secret $\mathbf{s}$ follows the noise distribution, the geometries of the lattices in (3.1) and (3.7) are the same, which is why without loss of generality we only consider (3.1) in this work save for Section 3.4.

To find short vectors, lattice reduction [LLL82, Sch87, GN08a, HPS11, CN11, MW16] can be applied. Thus, to establish the cost of solving an LWE instance, we may consider the cost of lattice reduction for solving uSVP.

Two conflicting estimates for the success of lattice reduction in solving uSVP are available in the literature. The first is going back to [GN08b] and was developed in [AFG14, APS15, Gö16, HKM17] for LWE. This estimate is commonly relied upon by designers in the literature, e.g. [BG14a, CHK+17, CKLS18, CLP17, ABB+17]. The second estimate was outlined in [ADPS16] and is relied upon in [BCD+16, BDK+18]. We will use the shorthand *2008 estimate* for the former and *2016 estimate* for the latter. As illustrated in Figure 3.1, the predicted costs under these two estimates differ greatly. For example, considering $n = 1024$, $q \approx 2^{15}$ and χ a discrete Gaussian with standard deviation $\sigma = 3.2$, the former predicts a cost of $\approx 2^{355}$ operations, whereas the latter predicts a cost of $\approx 2^{287}$ operations in the same cost model for lattice reduction.[2]

[1] Alternatively, we can perform lattice reduction on the q-ary lattice spanned by $\mathbf{A}^t$, i.e. the lattice spanned by the first m rows of (3.1), followed by an enumeration to find the closest (projected) lattice point to (the projection of) $\mathbf{c}$ [LP11, LN13].

[2] Assuming that an SVP oracle call in dimension β costs $2^{0.292\,\beta+16.4}$ [BDGL16, APS15], where $+16.4$ takes the place of $o(\beta)$ from the asymptotic formula and is based on experiments in [Laa15].

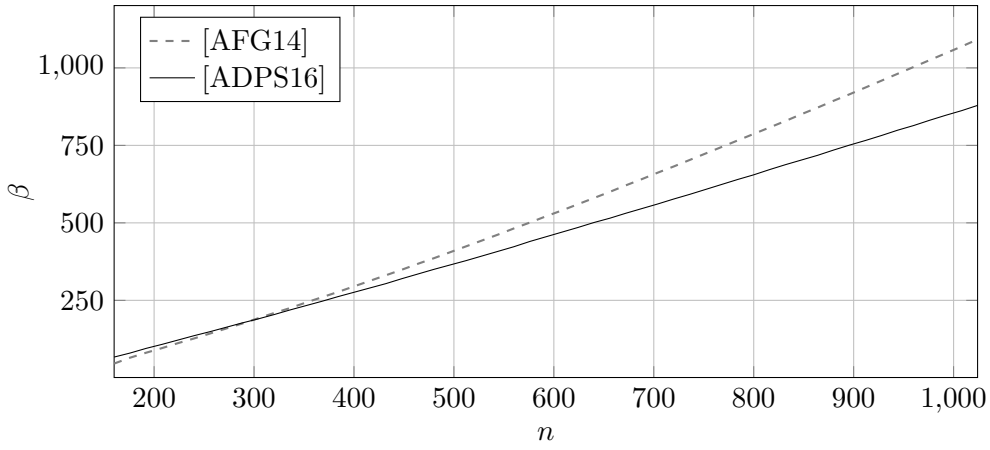


Figure 3.1: Required block size β according to the estimates given in [AFG14] and [ADPS16] for modulus $q = 2^{15}$, standard deviation $\sigma = 3.2$ and increasing n; for [AFG14] we set $\tau = 0.3$ and $c = 1$. Lattice reduction runs in time $2^{\Omega(\beta)}$.

Contribution and chapter roadmap. Relying on progress made in publicly available lattice reduction libraries [DT17, FPY17], we revisit the embedding approach for solving LWE resp. BDD under some reasonable assumptions about the LWE error distribution. In Section 3.2 we recall the two competing estimates for the cost of such approach from the literature. Then, in Section 3.3, we expand on the exposition from [ADPS16] followed by presenting the results of running 23,000 core hours worth of lattice reduction experiments in medium to larger block sizes β. Our results confirm that lattice reduction largely follows the behaviour expected from the 2016 estimate [ADPS16]. However, we also find that in our experiments the attack behaves somewhat better than expected.[3] In Section 3.3.3, we then explain the observed behaviour of the BKZ algorithm under the *Geometric Series Assumption* (GSA) and under the assumption that the unique shortest vector is distributed in a random direction relative to the rest of the basis. Finally, using the 2016 estimate, in Section 3.4 we show that some proposed parameters from the literature need to be updated to maintain the originally claimed level of security. In particular, we give reduced costs for solving the LWE instances underlying TESLA [ABB+17] and the somewhat homomorphic encryption scheme in [BCIV17]. We also show

[3] We note that this deviation from the expectation has a negligible impact on security estimates for cryptographic parameters.

that under the revised, corrected estimate, the primal attack performs about as well on SEAL v2.1 parameter sets as the dual attack from [Alb17].

All of our code and data can be found at `github.com/fvirdia/agvw17-code-data`.

3.2 Choosing BKZ block sizes

In this section we illustrate the two approaches used in the lattice literature to estimate the cost of using BKZ (cf. Algorithm 3 in Section 1.4.1) to solve uSVP over full-rank lattices of dimension d. The runtime of BKZ-β is dominated by that of the SVP oracle subroutine O_{SVP}, that repeatedly solves the Shortest Vector Problem in β-dimensional projective sublattices. The SVP oracle is often implemented using lattice point enumeration with preprocessing, which has time complexity $2^{\Theta(\beta \log \beta)}$ [HS07, GNR10, ABF+20], or lattice sieving, which has time and memory complexity $2^{\Theta(\beta)}$ [AKS01, NV08, BDGL16]. Therefore, to estimate the complexity of solving uSVP using lattice reduction, it is crucial to estimate the smallest block size β sufficient to recover the unique shortest vector $\mathbf{v} \in \Lambda$.

3.2.1 2008 Estimate

In [GN08b], Gama and Nguyen present a systematic experimental investigation into the behaviour of lattice reduction algorithms LLL, DEEP[4] and BKZ. In particular, they investigate the behaviour of these algorithms for solving Hermite-SVP, approx-SVP and unique-SVP for various families of lattices.

For unique-SVP, the authors performed experiments in small block sizes on two classes of semi-orthogonal lattices and on Lagarias-Odlyzko lattices [LO83], which permit to estimate the gap $\lambda_2(\Lambda)/\lambda_1(\Lambda)$ between the first and second minimum of the lattice. For all three families, [GN08b] observed that LLL and BKZ seem to recover a unique shortest vector with high probability whenever $\lambda_2(\Lambda)/\lambda_1(\Lambda) \geq \tau\delta^{d-1}$, where $\tau < 1$ is an empirically determined constant that

[4] That is, LLL with deep insertions [SE94].

depends on the lattice family and algorithm used, and δ is the root-Hermite factor of the algorithm over the lattice family.

In [AFG14] an experimental analysis of solving LWE based on the same estimate was carried out for lattices of the form (3.1). As mentioned above, this lattice contains an unusually short vector $\mathbf{v} = (\mathbf{e} \mid c)$ of squared norm $\lambda_1(\Lambda)^2 = \|\mathbf{v}\|^2 = \|\mathbf{e}\|^2 + c^2$. Thus, when $c = \|\mathbf{e}\|$ resp. $c = 1$ this implies $\lambda_1(\Lambda) \approx \sqrt{2m}\sigma$ resp. $\lambda_1(\Lambda) \approx \sqrt{m}\sigma$, with σ the standard deviation of $e_i \xleftarrow{\$} \chi$. The second minimum $\lambda_2(\Lambda)$ is assumed to correspond to the Gaussian Heuristic for the lattice. Experiments in [AFG14] using LLL and BKZ (with block sizes 5 and 10) confirmed the 2008 estimate, providing constant values for τ for lattices of the form (3.1), depending on the chosen algorithm, for a 10% success rate. Overall, τ was found to lie between 0.3 and 0.4 when using BKZ.

Still focusing on LWE, in [APS15] a closed formula for the root-Hermite factor δ required by an algorithm to solve LWE is given in function of n, σ, q, τ, which implicitly assumes $c = \|\mathbf{e}\|$. In [Gö16] a bound for δ in the [GN08b] model for the case of $c = 1$, which is usually used in practice, is given. In [HKM17], a related closed formula is given, directly expressing the asymptotic running time for solving LWE using this approach.

3.2.2 2016 Estimate

In [ADPS16], Alkim *et al.* outline an alternative estimate when using BKZ-β, which predicts that $\mathbf{e}$ can be found if[5]

$$\sqrt{\beta}\sigma \leq \delta^{2\beta-d-1} \operatorname{Vol}(\Lambda(\mathbf{B}))^{1/d}, \tag{3.2}$$

under the assumption that the Geometric Series Assumption holds (until a projection of the unusually short vector is found). The brief justification for this estimate given in [ADPS16] notes that this condition ensures that the projection of $\mathbf{e}$ orthogonally to the first $d - \beta$ basis vectors is shorter than

[5] In the published version of this chapter [AGVW17], the exponent of δ in (3.2) is reported as $2\beta - d$. This was obtained by assuming the "δ^n" definition of the root-Hermite factor from [LP11]. Changing the definition to "δ^{n-1}" as argued in § 1.4.1, the $2\beta - d - 1$ exponent present in [ADPS16] follows. We have therefore amended this chapter to incorporate the change.

the expectation for $\|\mathbf{b}^*_{d-\beta+1}\|$ under the GSA and thus would be found by the SVP oracle when called on the last block of size β. Hence, for any β satisfying (3.2), the actual behaviour would deviate from that predicted by the GSA. The argument can be completed by appealing to the intuition that in principle detecting this deviation would suffice for solving Decision-LWE. We will see in Section 3.3.1 that this also results in a solution to uSVP and hence Search-LWE.

To derive (3.2), we express their argument in general terms. Their approach consists of finding the smallest β such that in the final full sized block starting at index $d-\beta+1$,

$$\|\pi_{d-\beta+1}(\mathbf{v})\| \leq \left\|\mathbf{b}^*_{d-\beta+1}\right\|, \tag{3.3}$$

resulting in O_{SVP} recovering the projection of $\mathbf{v}$ at index $d-\beta+1$. In [ADPS16], the authors consider normal form LWE, and assume the secret distribution χ to be centred around 0. Using Kannan's embedding (3.1), the uSVP solution will be an embedded vector $\mathbf{v} = (\mathbf{e} \mid c)$ of dimension $d = m+1$ for which each entry is drawn i.i.d. from a distribution of standard deviation σ and mean $\mu = 0$, with the addition of one final constant entry c, usually set to 1. The squared norm $\|\mathbf{v}\|^2$ may be modelled as a random variable following a scaled chi-squared distribution $\sigma^2 \cdot \chi^2_{d-1}$ with $d-1$ degrees of freedom, plus a fixed contribution from c, resulting in $\mathbb{E}(\|\mathbf{v}\|^2) = (d-1)\sigma^2 + c^2$, since χ is close to a discrete Gaussian distribution (in their case, it is a centered binomial distribution).

In [ADPS16], the authors approximate the left hand side of (3.3) as

$$\|\pi_{d-\beta+1}(\mathbf{v})\| \approx \mathbb{E}(\|\mathbf{v}\|)\sqrt{\beta/d} \approx \sigma\sqrt{\beta},$$

by using[6] $\mathbb{E}(\|\mathbf{v}\|) \approx \mathbb{E}(\|\mathbf{v}\|^2)^{1/2} \approx \sigma\sqrt{d}$ and rescaling by $\sqrt{\beta/d}$ to account for the orthogonal projection from a d-dimensional vector space to a β-dimensional subspace. To approximate the right hand side of (3.3), Alkim *et al.* make use of the GSA. Assuming that BKZ-β returns a first basis vector of length $\ell_1(\beta)$ and that it outputs a basis with GSA factor $\alpha(\beta)$, this becomes

$$\left\|\mathbf{b}^*_{d-\beta+1}\right\| \approx \alpha(\beta)^{d-\beta} \cdot \ell_1(\beta),$$

[6]The error in this assumption tends to 0 as $d \to \infty$, so we ignore it. The $\mathbb{E}(\|\mathbf{v}\|) \approx \mathbb{E}(\|\mathbf{v}\|^2)^{1/2}$ approximation can be avoided altogether by working with squared norms.

which using Lemma 7 and the definition of the root-Hermite factor δ, can be estimated as

$$\alpha(\beta)^{d-\beta} \cdot \ell_1(\beta) = \delta^{2\beta-2d} \cdot \delta^{d-1} \operatorname{Vol}(\Lambda(\mathbf{B}))^{1/d} = \delta^{2\beta-d-1} \operatorname{Vol}(\Lambda(\mathbf{B}))^{1/d}.$$

Putting both approximations together, this results in condition (3.2), that β must satisfy for solving uSVP using BKZ-β. We note that technically Alkim *et al.* define δ directly using the formula that results from Lemma 9, rather than as the $(d-1)$-th root of the Hermite factor. This does not affect our analysis, since our definition of δ together with assuming that the GSA holds, implies their definition of δ by using Lemma 9.

3.3 Solving uSVP

Given the significant differences in expected solving time under the two estimates, cf. Figure 3.1, and the progress made in publicly available lattice reduction libraries enabling experiments in larger block sizes [DT17, FPY17], we conduct a more detailed examination of BKZ's behaviour on uSVP instances. For this, we establish the behaviour we would expect during lattice reduction if the intuition from [ADPS16] is correct, which we then experimentally investigate in Section 3.3.2. Overall, our experiments generally confirm the correctness of the approach from [ADPS16]. However, BKZ behaves somewhat better than expected, as we will explain in Section 3.3.3.

For the rest of this section, let $\mathbf{v}$ be a unique shortest vector in some lattice $\Lambda \subset \mathbb{R}^d$, i.e. in case of (3.1) we have $\mathbf{v} = (\mathbf{e} \mid c)$ where we pick $c = 1$.

3.3.1 Prediction

Projected norm. In what follows, we assume the unique shortest vector $\mathbf{v}$ is drawn from a spherical distribution or is at least "not too skewed" with respect to the current basis. As a consequence, following [ADPS16], we assume that all orthogonal projections of $\mathbf{v}$ onto a k-dimensional subspace of $\mathbb{R}^d$ have expected

norm $\sqrt{k/d}\,\|\mathbf{v}\|$. Note that this assumption can be dropped by adapting (3.2) to $\|\mathbf{v}\| \leq \delta^{2\beta-d-1}\,\mathrm{Vol}(\Lambda)^{\frac{1}{d}}$ since $\|\pi_{d-\beta+1}(\mathbf{v})\| \leq \|\mathbf{v}\|$.

Finding a projection of the short vector. Assume that β is chosen minimally such that (3.2) holds. When running BKZ-β the length of the Gram-Schmidt basis vectors of the current basis converge to the lengths predicted by the GSA. Therefore, at some point BKZ will find a basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of Λ for which we can assume that the GSA holds with root Hermite factor δ, that is $\|\mathbf{b}_i^*\| = \alpha^{i-1}\|\mathbf{b}_1^*\|$ where $\delta = \alpha^{-1/2}$ by Remark 8. Now, consider the stage of BKZ where the SVP oracle is called on the last full projected block of size β with respect to $\mathbf{B}$. Note that the projection $\pi_{d-\beta+1}(\mathbf{v})$ of the shortest vector is contained in the lattice

$$\Lambda_{d-\beta+1}^{\perp} := \Lambda\left(\pi_{d-\beta+1}(\mathbf{b}_{d-\beta+1}), \ldots, \pi_{d-\beta+1}(\mathbf{b}_d)\right),$$

since

$$\pi_{d-\beta+1}(\mathbf{v}) = \sum_{i=d-\beta+1}^{d} \nu_i\, \pi_{d-\beta+1}(\mathbf{b}_i), \text{ where } \nu_i \in \mathbb{Z} \text{ with } \mathbf{v} = \sum_{i=1}^{d} \nu_i\, \mathbf{b}_i.$$

By (3.2), the projection $\pi_{d-\beta+1}(\mathbf{v})$ is in fact expected to be the shortest non-zero vector in $\Lambda_{d-\beta+1}^{\perp}$, since it is shorter than the GSA's estimate for $\lambda_1(\Lambda_{d-\beta+1}^{\perp})$, i.e.

$$\|\pi_{d-\beta+1}(\mathbf{v})\| \approx \sqrt{\frac{\beta}{d}}\,\|\mathbf{v}\| \leq \alpha^{d-\beta} \cdot \|\mathbf{b}_1^*\| = \delta^{-2(d-\beta)} \cdot \delta^{d-1}\mathrm{Vol}(\Lambda)^{\frac{1}{d}}.$$

Hence the SVP oracle will find $\pm\pi_{d-\beta+1}(\mathbf{v})$ and BKZ inserts

$$\mathbf{b}_{d-\beta+1}^{\mathsf{new}} = \pm \sum_{i=d-\beta+1}^{d} \nu_i \mathbf{b}_i$$

into the basis $\mathbf{B}$ at position $d-\beta+1$, as already outlined in [ADPS16]. In other words, by finding $\pm\pi_{d-\beta+1}(\mathbf{v})$, BKZ recovers the last β coefficients $\nu_{d-\beta+1}, \ldots, \nu_d$ of $\mathbf{v}$ with respect to the basis $\mathbf{B}$.

Finding the short vector. As hinted in Section 3.2.2, the above argument can be extended to an argument for the full recovery of $\mathbf{v}$. Consider the case

that in some tour of BKZ-β, a projection of $\mathbf{v}$ was found at index $d-\beta+1$. Then in the following tour, by arguments analogous to the ones above, a projection of $\mathbf{v}$ will likely be found at index $d-2\beta+2$, since now it holds that

$$\pi_{d-2\beta+2}(\mathbf{v}) \in \Lambda\left(\pi_{d-2\beta+2}(\mathbf{b}_{d-2\beta+2}), \ldots, \pi_{d-2\beta+2}(\mathbf{b}^{\mathsf{new}}_{d-\beta+1})\right).$$

Repeating this argument for smaller indices shows that after a few tours $\mathbf{v}$ will be recovered. Potentially, once $\pi_{d-\beta+1}(\mathbf{v})$ is recovered by BKZ-β at index $d-\beta+1$, recovery of $\mathbf{v}$ could be obtained using BKZ-β' with a smaller block size $\beta' \ll \beta$ using the same argument as above (since the basis is already BKZ-β reduced), as to make this step into a significantly cheaper post-processing phase. Furthermore, noting that BKZ calls LLL which in turn calls size-reduction, i.e. Babai's nearest plane [Bab86], at some index $i > 1$ size-reduction alone will recover $\mathbf{v}$ from $\pi_i(\mathbf{v})$. In particular, it is well-known (eg. [DD18]) that size-reduction (Algorithm 1) will succeed in recovering $\mathbf{v}$ whenever

$$\mathbf{v} \in \mathbf{b}^{\mathsf{new}}_{d-\beta+1} + \left\{ \sum_{i=1}^{d-\beta} c_i \cdot \mathbf{b}^*_i : c_i \in \left[-\frac{1}{2}, \frac{1}{2}\right] \right\}. \tag{3.4}$$

3.3.2 Observation

The above discussion naturally suggests a strategy to verify the expected behaviour. We have to verify that the projected norms $\|\pi_i(\mathbf{v})\| = \|\pi_i(\mathbf{e} \mid 1)\|$ do indeed behave as expected and that $\pi_{d-\beta+1}(\mathbf{v})$ is recovered by BKZ-β for the minimal $\beta \in$ satisfying (3.2). Finally, we have to measure when and how $\mathbf{v} = (\mathbf{e} \mid 1)$ is eventually recovered.

Thus, we ran lattice reduction on many lattices constructed from LWE instances using Kannan's embedding. In particular, we picked the entries of $\mathbf{s}$ and $\mathbf{A}$ uniformly at random from $\mathbb{Z}_q$, the entries of $\mathbf{e}$ from a discrete Gaussian distribution with standard deviation $\sigma = 8/\sqrt{2\pi}$, and we constructed our basis as in (3.1) with embedding coefficient $c = 1$. For parameters (n, q, σ), we then estimated the minimal pair (in lexicographical order) (β, m) to satisfy (3.2).

Implementation. To perform our experiments, we used SageMath 7.5.1 [S+17] in combination with the `fplll` 5.1.0 [DT17] and `fpylll` 0.2.4dev [FPY17] li-

braries. All experiments were run on a machine with Intel(R) Xeon(R) CPU E5-2667 v2 @ 3.30GHz cores ("strombenzin") resp. Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz ("atomkohle"). Each instance was reduced on a single core, with no parallelisation.

Our BKZ implementation inherits from the implementation in `fplll` and `fpylll` of BKZ 2.0 [CN11, Che13] algorithm. As in BKZ 2.0, we restricted the enumeration radius to be approximately the size of the Gaussian Heuristic for the projected sublattice, apply recursive BKZ-β' preprocessing with a block size $\beta' < \beta$, make use of extreme pruning [GNR10] and terminate the algorithm when it stops making significant progress. We give simplified pseudo-code of our implementation in Algorithm 6. We ran BKZ for at most 20 tours using `fplll`'s default pruning and preprocessing strategies and, using `fplll`'s default auto-abort strategy, terminated the algorithm whenever the slope of the Gram-Schmidt vectors did not improve for five consecutive tours. Additionally, we aborted if a vector of length $\approx \|\mathbf{v}\|$ was found in the basis (after line 14 of Algorithm 6).

Implementations of block-wise lattice reduction algorithms such as BKZ make heavy use of LLL [LLL82] and size-reduction. This is to remove linear dependencies introduced during the algorithm, to avoid numerical stability issues [PS08] and to improve the performance of the algorithm by moving short vectors to the front earlier. The main modification in our implementation is that calls to LLL during preprocessing and post-processing are restricted to the current block, not touching any other vector, to aid analysis. That is, in Algorithm 6, LLL is called in lines 7 and 12 and we modified these LLL calls not to touch any row with index smaller than κ, not even to perform size-reduction.

As a consequence, we only make use of vectors with index smaller than κ in lines 3 and 14. Following the implementations in [DT17, FPY17], we call size-reduction from index 1 to κ before (line 3) and after (line 14) the innermost loop with calls to the SVP oracle. These calls do not appear in the original description of BKZ. However, since the innermost loop re-randomises the basis when using extreme pruning, the success condition of the original BKZ algorithm needs to be altered. That is, the algorithm cannot break the

Data: LLL-reduced lattice basis $\mathbf{B}$
Data: block size β, preprocessing block size β'

1 **repeat** // tour
2 **for** $\kappa \leftarrow 1$ **to** d **do** // step$_\kappa$
3 size-reduction from index 1 to κ (inclusive)
4 $\ell \leftarrow \|b_\kappa^*\|$
  // extreme pruning + recursive preprocessing
5 **repeat** *until termination condition met*
6 rerandomise $\pi_\kappa(\mathbf{b}_{\kappa+1}, \ldots, \mathbf{b}_{\kappa+\beta-1})$
7 LLL on $\pi_\kappa(\mathbf{b}_\kappa, \ldots, \mathbf{b}_{\kappa+\beta-1})$
8 BKZ-β' on $\pi_\kappa(\mathbf{b}_\kappa, \ldots, \mathbf{b}_{\kappa+\beta-1})$
9 $\mathbf{v} \leftarrow$ SVP on $\pi_\kappa(\mathbf{b}_\kappa, \ldots, \mathbf{b}_{\kappa+\beta-1})$
10 **if** $\mathbf{v} \neq \perp$ **then**
11 extend $\mathbf{B}$ by inserting $\mathbf{v}$ into $\mathbf{B}$ at index $\kappa + \beta$
12 LLL on $\pi_\kappa(\mathbf{b}_\kappa, \ldots, \mathbf{b}_{\kappa+\beta})$ to remove linear dependencies
13 drop row with all zero entries
14 size-reduction from index 1 to κ (inclusive)
15 **if** $\ell = \|b_\kappa^*\|$ **then**
16 **yield** $\top$
17 **else**
18 **yield** $\perp$
19 **if** $\top$ *for all* κ **then**
20 **return**;

Algorithm 6: Simplified BKZ 2.0 Algorithm as used in this chapter's experiments, see § 3.3.2.

outer loop once it makes no more changes as originally specified. Instead, the algorithm terminates if it does not find a shorter vector at any index κ. Now, the calls to size-reduction ensure that the comparison at the beginning and end of each step κ is meaningful even when the Gram-Schmidt vectors are only updated lazily in the underlying implementation. That is, the calls to size-reduction trigger an internal update of the underlying Gram-Schmidt vectors and are hence implementation artefacts. The reader may think of these size-reduction calls as explicating calls otherwise hidden behind calls to LLL and we stress that our analysis applies to BKZ as commonly implemented, our changes merely enable us to more easily predict and experimentally verify the behaviour.

We note that the break condition for the innermost loop at line 5 depends on the pruning parameters chosen, which control the success probability of

enumeration. Since it does not play a material role in our analysis, we simply state that some condition will lead to a termination of the innermost loop.

Finally, we recorded the following information. At the end of each step κ during lattice reduction, we recorded the minimal index i such that $\pi_i(\mathbf{v})$ is in $\mathrm{span}_{\mathbb{R}}(\mathbf{b}_1, \ldots, \mathbf{b}_i)$ and whether $\pm\mathbf{v}$ itself is in the basis. In particular, to find the index i in the orthogonalised basis $\mathbf{B}^*$ of $\pi_i(\mathbf{v})$ given $\mathbf{v}$, we compute the coefficients of $\mathbf{v}$ in basis $\mathbf{B}^*$ (at the current step) and pick the first index i such that all coefficients with larger indices are zero. Then, we have $\pi_i(\mathbf{b}_i) = \lambda \cdot \pi_i(\mathbf{v})$ for some $\lambda \in \mathbb{R}$. From the algorithm, we expect to have found $\pm\pi_i(\mathbf{b}_i) = \pi_i(\mathbf{v})$ and call i the index of the projection of $\mathbf{v}$.

Results. In Figure 3.2, we plot the average norms of $\pi_i(\mathbf{v})$ against the expectation $\sqrt{d-i+1}\,\sigma \approx \sqrt{\frac{d-i+1}{d}}\sqrt{m \cdot \sigma^2 + 1}$, indicating that $\sqrt{d-i+1}\,\sigma$ is a close approximation of the expected lengths except perhaps for the last few indices.

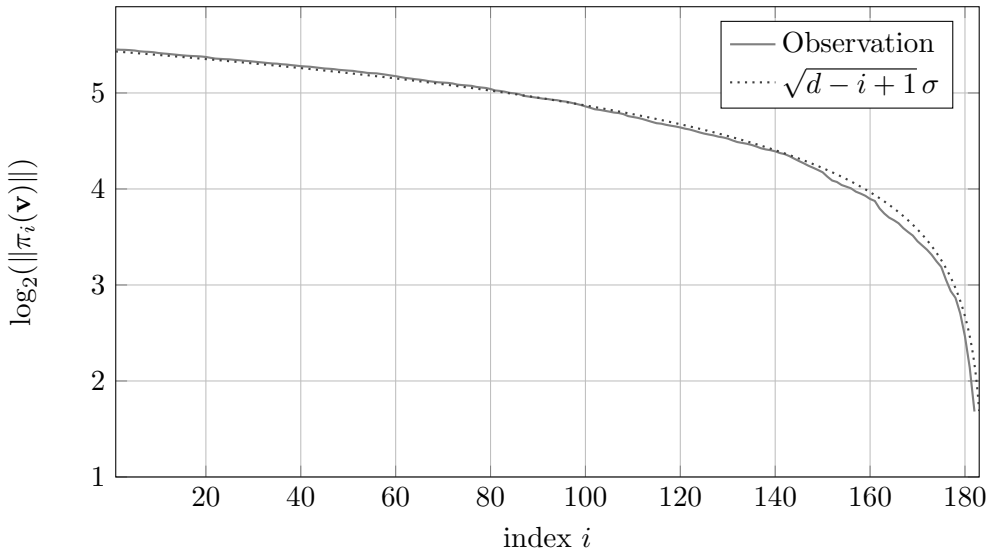


Figure 3.2: Expected and average observed norms $\|\pi_i(\mathbf{v})\|$ for 16 bases (LLL-reduced) and vectors $\mathbf{v}$ of dimension $d = m + 1$ and volume q^{m-n} with LWE parameters $n = 65, m = 182, q = 521$ and standard deviation $\sigma = 8/\sqrt{2\pi}$.

Recall that, as illustrated in Figure 3.3, we expect to find the projection $\pi_{d-\beta+1}(\mathbf{v})$ when (β, d) satisfy (3.2), eventually leading to a recovery of $\mathbf{v}$

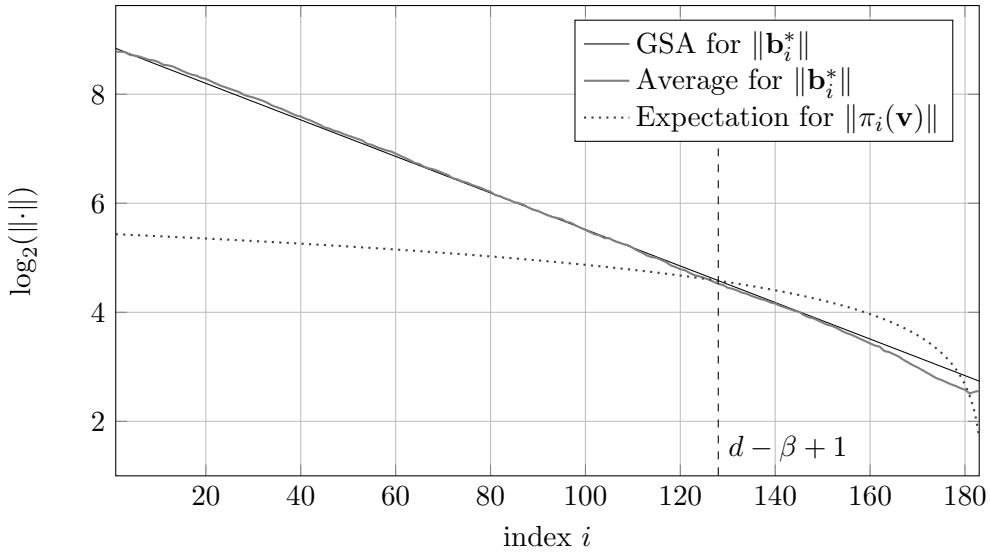


Figure 3.3: Expected and observed norms for lattices of dimension $d = m+1 = 183$ and volume q^{m-n} after BKZ-β reduction for LWE parameters $n = 65, m = 182, q = 521$ and standard deviation $\sigma = 8/\sqrt{2\pi}$ and $\beta = 56$ (minimal (β, m) such that (3.2) holds). Average of Gram-Schmidt lengths is taken over 16 BKZ-β reduced bases of random q-ary lattices, i.e. *without* an unusually short vector.

by, say, an extension of the argument for the recovery of $\pi_{d-\beta+1}(\mathbf{v})$. Our experiments, summarised in Table 3.1, show a related, albeit not identical behaviour. Defining a cut-off index $c_{\text{off}} = d - 0.9\beta + 1$ and considering $\pi_\kappa(\mathbf{v})$ for $\kappa < c_{\text{off}}$, we observe that the BKZ algorithm typically first recovers $\pi_\kappa(\mathbf{v})$ which, unlike what we predicted in Section 3.3.1, is immediately followed by the recovery of $\mathbf{v}$ in the same step, as in without need to further tours or calls to O_{SVP}. In more detail, in Figure 3.4 we show the measured probability distribution of the index κ such that $\mathbf{v}$ is recovered from $\pi_\kappa(\mathbf{v})$ in the same step. Note that the mode of this distribution is smaller than $d - \beta + 1$. We explain this bias in Section 3.3.3.

The recovery of $\mathbf{v}$ from $\pi_\kappa(\mathbf{v})$ can be effected by one of three subroutines: either by a call to LLL, by a call to size-reduction, or by a call to enumeration that recovers $\mathbf{v}$ directly (due to hypothetically having $\mathbf{v} = \pi_\kappa(\mathbf{v})$). Since LLL itself contains many calls to size-reduction, and enumeration being lucky is rather unlikely, size-reduction is a good place to start the investigation. Indeed, restricting the LLL calls in Algorithm 6 as outlined in Section 3.3.2,

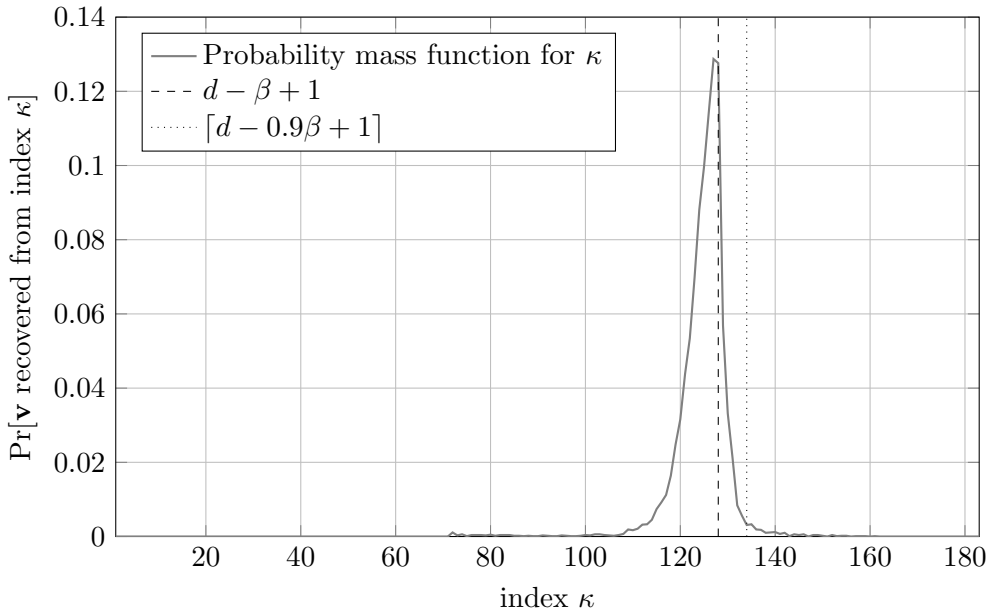


Figure 3.4: Probability mass function of the index κ from which size-reduction recovers $\mathbf{v}$, calculated over 10,000 lattice instances with LWE parameters $n = 65, m = 182, q = 521$ and standard deviation $\sigma = 8/\sqrt{2\pi}$, reduced using $\beta = 56$. The mean of the distribution is ≈ 124.76 while $d - \beta + 1 = 128$.

identifies that size-reduction suffices. That is, to measure the success rate of size-reduction recovering $\mathbf{v}$ from $\pi_\kappa(\mathbf{v})$, we observe size-reduction acting on $\pi_\kappa(\mathbf{v})$. Here, we consider size-reduction to fail in recovering $\mathbf{v}$ if it does not recover $\mathbf{v}$ given $\pi_\kappa(\mathbf{v})$ for $\kappa < c_{\text{off}}$ with $c_{\text{off}} = d - 0.9\beta + 1$, regardless of whether $\mathbf{v}$ is finally recovered at a later point either by size-reduction on a new projection, or by some other call in the algorithm such as an SVP oracle call at a smaller index. As shown in Table 3.1, size-reduction's success rate is close to 1. Note that the cut-off index c serves to limit underestimating the success rate: intuitively we do not expect size-reduction to succeed when starting from a projection with larger index, such as $\pi_{d-\gamma+1}(\mathbf{v})$ with $\gamma < 10$. We discuss this in Section 3.3.3.

Overall, Table 3.1 confirms the prediction from [ADPS16]: picking $\beta = \beta_{2016}$ to be the block size predicted by the 2016 estimate leads to a successful recovery of $\mathbf{v}$ with high probability.

n	q	β_{2016}	m_{2016}	β	#	$\mathbf{v}$	same step		time
							$\kappa < c_{\text{off}}$	$\kappa = d-\beta+1$	
65	521	56	182	56	10^4	93.3%	99.7%	99.7%	1,131.4
				51		52.8%	98.8%	97.3%	1,359.3
				46		4.8%	96.4%	85.7%	1,541.2
80	1031	60	204	60	10^3	94.2%	99.6%	100.0%	2,929.0
				55		60.6%	99.3%	96.5%	2,458.5
				50		8.9%	97.6%	100.0%	1,955.0
				45		0.2%	100.0%	—	1,568.1
100	2053	67	243	67	500	88.8%	99.8%	100.0%	28,803.7
				62		39.6%	99.5%	100.0%	19,341.9
				57		5.8%	100.0%	100.0%	7,882.2
				52		0.2%	0.0%	—	3,227.0
108	2053	77	261	77	5	100.0%	100.0%	100.0%	351,094.2
110	2053	78	272	78	5	100.0%	100.0%	100.0%	1,012,634.8

Table 3.1: Overall success rate ("$\mathbf{v}$") and success rate of size-reduction ("same step") for solving LWE instances characterised by n, σ, q with m samples, standard deviation $\sigma = 8/\sqrt{2\pi}$, minimal (β_{2016}, m_{2016}) such that $\sqrt{\beta_{2016}}\,\sigma \leq \delta^{2\beta_{2016}-(m_{2016}+1)-1} q^{(m_{2016}-n)/(m_{2016}+1)}$ with δ in function of β_{2016} following [Che13], see § 1.4.1.2. The column "β" gives the actual block size used in experiments. The "same step" rate is calculated over all successful instances where $\mathbf{v}$ is found before the cut-off point c_{off} and for the instances where exactly $\pi_{d-b+1}(\mathbf{v})$ is found (if no such instance is found, we do not report a value). In the second case, the sample size is smaller, since not all instances recover $\mathbf{v}$ from exactly $\kappa = d - \beta + 1$. The column "time" lists average solving CPU time for one instance, in seconds. Note that our changes to the algorithm and our extensive record keeping lead to an increased running time of the BKZ algorithm compared to [DT17, FPY17]. Furthermore, the occasional longer running time for smaller block sizes is explained by the absence of early termination due to $\mathbf{v}$ not being found.

3.3.3 Explaining observation

As noted above, our experiments indicate that the algorithm behaves better than expected by (3.2). Firstly, the BKZ algorithm does not necessarily recover a projection of $\mathbf{v}$ at index $d - \beta + 1$. Instead, the index κ at which we recover a projection $\pi_\kappa(\mathbf{v})$ follows a distribution with a centre below $d - \beta + 1$, cf. Figure 3.4. Secondly, size-reduction usually immediately recovers $\mathbf{v}$ from $\pi_\kappa(\mathbf{v})$. This is somewhat unexpected, since we do not have the guarantee that $|c_i| \leq 1/2$ as required in the success condition of size-reduction given in (3.4). Thirdly, as it can be seen in Table 3.1, picking β as suggested by the 2016 estimate results in recovery of the target vector about 90% of the time. However, somewhat smaller block sizes also present some relatively high success probability.

Finding the projection. To explain the bias towards a recovery of $\pi_\kappa(\mathbf{v})$ for some $\kappa < d - \beta + 1$, note that if (3.2) holds then for the parameter sets *in our experiments* the lines for $\|\pi_i(\mathbf{v})\|$ and $\|\mathbf{b}_i^*\|$ intersect twice (cf. Figure 3.3). Let $d - \gamma + 1$ be the index of the second intersection. Thus, there is a good chance that $\|\pi_{d-\gamma+1}(\mathbf{v})\|$ is a shortest vector in the lattice spanned by the last projected block of some small rank γ and will be placed at index $d - \gamma + 1$. As a consequence, all projections $\pi_i(\mathbf{v})$ with $i > d - \gamma + 1$ will be zero and $\pi_{d-\beta-\gamma+1}(\mathbf{v})$ will be contained in the β-dimensional lattice

$$\Lambda\left(\pi_{d-\beta-\gamma+1}(\mathbf{b}_{d-\beta-\gamma+1}), \ldots, \pi_{d-\beta-\gamma+1}(\mathbf{b}_{d-\gamma+1})\right),$$

enabling it to be recovered by BKZ-β at an index $d - \beta - \gamma + 1 < d - \beta + 1$ during the successive tour. Thus, BKZ in our experiments behaves better than predicted by (3.2). We note that another effect of this second intersection is that, for very few instances, it directly leads to a recovery of $\mathbf{v}$ from $\pi_{d-\beta-\gamma+1}(\mathbf{v})$.

Giving a closed formula incorporating this effect akin to (3.2) would entail to predict the index γ and then replace β with $\beta + \gamma$ in (3.2), while keeping δ the root-Hermite factor of BKZ-β. However, as illustrated in Figures 3.2 and 3.3, neither the prediction $\sqrt{d - i + 1}\,\sigma$ for $\|\pi_{d-1+1}(\mathbf{v})\|$ nor the GSA hold for the last 50 or so indices of the basis [CN11, Che13]. Furthermore, we stress that

while the second intersection often occurs for parameter sets within reach of practical experiments, it does not always occur for all parameter sets. That is, for many large parameter sets (n, σ, q), e.g. those in [ADPS16], a choice of β satisfying (3.2) does *not* lead to a predicted second intersection at some larger index. Thus, this effect may highlight the pitfalls of extrapolating experimental lattice reduction data from small instances to large instances, and not an inherent property of the primal attack using BKZ.

Finding the short vector. As noticed before, from our experiments it seems that size-reduction is able to recover $\mathbf{v}$ from $\pi_{d-\beta+1}(\mathbf{v})$ with high probability. From the point of view of costing the primal attack, a conservative choice would be that of assuming that this happens with 100% probability, ignoring the cost of any hypothetical post-processing tours needed to recover $\mathbf{v}$. In addition, it is possible to give a heuristic argument justifying the high success probability of size-reduction under a certain independence assumption, similar to those already used in the study of decoding [LP11, §4] and hybrid attacks [BGPW16, Heuristic 4] and compared in [Wun18, §5.3.2]. In what follows, we assume that the GSA exactly holds and that the projected norms $\|\pi_i(\mathbf{v})\|$ are equal to their expected value (cf. Figure 3.2). Under these assumptions, we show that size-reduction recovers the short vector $\mathbf{v}$ with high probability. More precisely, we show:

Claim 1. *Let $\mathbf{v} \in \Lambda \subset \mathbb{R}^d$ be a unique (up to sign) shortest vector and $\beta > 2$ be an integer. Assume that (3.2) holds, that the current basis is $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ such that $\mathbf{b}_k^* = \pi_k(\mathbf{v})$ for $k = d - \beta + 1$ and*

$$\mathbf{v} = \mathbf{b}_k + \sum_{i=1}^{k-1} \nu_i \mathbf{b}_i = \mathbf{b}_k^* + \sum_{i=1}^{k-1} \nu_i^* \mathbf{b}_i^* / \|\mathbf{b}_i^*\| \tag{3.5}$$

for some $\nu_i \in \mathbb{Z}$ and $\nu_i^* \in \mathbb{R}$, that $\|\pi_i(\mathbf{v})\| = \sqrt{d - i + 1}\,\sigma$ for $i \leq k$, and that the GSA holds for $\mathbf{B}$ until index k. If the size-reduction step of BKZ-β is called on $\mathbf{b}_k$, it recovers $\mathbf{v}$ with high probability over the randomness of the basis.

Note that if BKZ has just found a projection of $\mathbf{v}$ at index k, the current basis is as required by Claim 1. Now, let $\nu_i \in \mathbb{Z}$ denote the coefficients of $\mathbf{v}$ with

respect to the basis $\mathbf{B}$, i.e.

$$\mathbf{v} = \mathbf{b}_{d-\beta+1} + \sum_{i=1}^{d-\beta} \nu_i \mathbf{b}_i.$$

We proceed to size-reduce $\mathbf{b}_{d-\beta+1}$ with the basis vectors $\mathbf{b}_i$ for $i = d-\beta, d-\beta-1, \ldots, 1$. Let $\mathbf{b}_{d-\beta+1}^{(d-\beta+1)} = \mathbf{b}_{d-\beta+1}$, where the superscript denotes a step during size-reduction. Before step i we have

$$\begin{aligned}
\mathbf{b}_k^{(i+1)} &= \mu_{k,1}^{(i+1)}\mathbf{b}_1^* + \cdots + \mu_{k,i}^{(i+1)}\mathbf{b}_i^* + \cdots + \mu_{k,k-1}^{(i+1)}\mathbf{b}_{k-1}^* + \mathbf{b}_k^* \\
\pi_i(\mathbf{b}_k^{(i+1)}) &= \mu_{k,i}^{(i+1)}\mathbf{b}_i^* + \cdots + \mu_{k,k-1}^{(i+1)}\mathbf{b}_{k-1}^* + \mathbf{b}_k^* \\
\mathbf{b}_i &= \mu_{i,1}\mathbf{b}_1^* + \cdots + \mu_{i,i-1}\mathbf{b}_{i-1}^* + \mathbf{b}_i^*,
\end{aligned}$$

where no changes are being made to the basis vectors $\mathbf{b}_i$ for $i < k$, such that the basis with respect to which projections π_i are made can be left implicit. After step i, size-reduction produces $\mathbf{b}_k^{(i)} \leftarrow \mathbf{b}_k^{(i+1)} + \lambda \mathbf{b}_i$ for $\lambda \in \mathbb{Z}$ such that $|\mu_{k,i}^{(i)}| = |\langle \mathbf{b}_k^{(i)}, \mathbf{b}_i^* \rangle| / \|\mathbf{b}_i^*\|^2 = |\mu_{k,i}^{(i+1)} + \lambda| \leq 1/2$. Projecting orthogonally to $\mathrm{span}_{\mathbb{R}}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$, we can see that, equivalently, during step i size-reduction is finding the shortest vector in the coset $L_i \coloneqq \pi_i(\mathbf{b}_k^{(i+1)}) + \mathbb{Z}\mathbf{b}_i^*$.

Let C_i be the condition that after step i of size-reduction, $\mathbf{b}_k^{(i)} = \mathbf{b}_k^{(k)} + \sum_{j=i}^{k-1} \nu_j \mathbf{b}_j$. We know that C_i is trivially true for $i = k$. Assuming C_{i+1} is true,

$$\begin{aligned}
\pi_i(\mathbf{v}) &= \pi_i(\mathbf{b}_k^{(k)} + \sum_{j=1}^{k-1} \nu_j \mathbf{b}_j) \quad \text{by (3.5)} \\
&= \pi_i(\mathbf{b}_k^{(k)}) + \sum_{j=i+1}^{k-1} \nu_j \pi_i(\mathbf{b}_j) + \nu_i \pi_i(\mathbf{b}_i) \\
&= \pi_i(\mathbf{b}_k^{(i+1)}) + \nu_i \mathbf{b}_i^* \qquad \text{by } C_{i+1} \text{ and } \mathbf{b}_i^* = \pi_i(\mathbf{b}_i) \\
&\in \pi_i(\mathbf{b}_k^{(i+1)}) + \mathbb{Z}\mathbf{b}_i^* = L_i.
\end{aligned}$$

Let E_i be the event that $\pi_i(\mathbf{v})$ is the shortest element in the L_i coset. Then, the i-th step of size-reduction will recover the ν_i coefficient and set $\mathbf{b}_k^{(i)}$ so that condition C_i is satisfied. By induction, if $\pi_i(\mathbf{v})$ is the shortest element in L_i for all i (which happens with probability $P[E_1 \wedge \cdots \wedge E_{k-1}]$), size-reduction finds the shortest vector $\mathbf{v} = \mathbf{b}_{d-\beta+1}^{(1)}$ and inserts it into the basis at position $d-\beta+1$, replacing $\mathbf{b}_{d-\beta+1}$. If the adversary checks the status of the basis after each call to size-reduction, they can easily detect this event and solve Search-LWE.

It remains to argue that the probability $p = P[E_1 \wedge \cdots \wedge E_{k-1}]$ is high. Let

$$p_i = P[E_i] = \Pr\left[\pi_i(\mathbf{v}) \text{ is the shortest element in } L_i\right].$$

Following the analysis of the success probability of the nearest plane algorithm in [LP11, BGPW16], we assume that the events E_i are independent when $\mathbf{v}$ is (nearly) spherically distributed, as in our case. Hence, we can proceed to compute $p = \prod_{i=1}^{d-\beta} p_i$. For each i the probability p_i is equal to the probability that

$$\|\pi_i(\mathbf{v})\| < \min\{\|\pi_i(\mathbf{v}) + \mathbf{b}_i^*\|, \|\pi_i(\mathbf{v}) - \mathbf{b}_i^*\|\}$$

as illustrated in Figure 3.5.

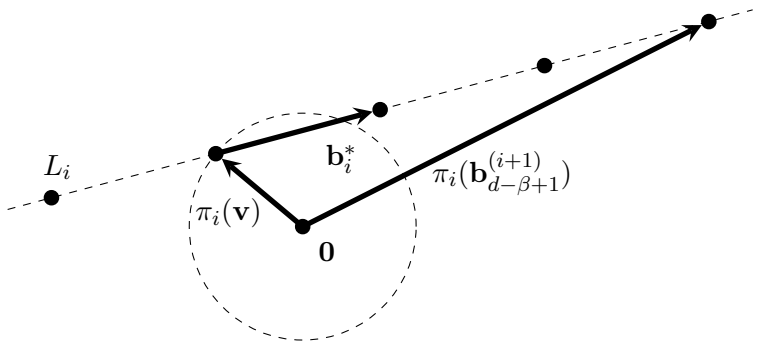


Figure 3.5: Illustration of a case such that $\pi_i(\mathbf{v})$ is the shortest element on L_i.

To approximate the probabilities p_i, we model them as follows. By assumption, we have

$$r_i := \|\pi_i(\mathbf{v})\| = (\sqrt{d-i+1}/\sqrt{d})\,\|\mathbf{v}\| \text{ and } R_i := \|\mathbf{b}_i^*\| = \delta^{-2(i-1)+d-1}\mathrm{Vol}(\Lambda)^{\frac{1}{d}},$$

and that $\pi_i(\mathbf{v})$ is uniformly distributed with norm r_i. We can therefore model p_i as described in the following and illustrated in Figure 3.6.

Pick a point $\mathbf{w}$ with norm r_i uniformly at random. Then the probability p_i is approximately the probability that $\mathbf{w}$ is closer to $\mathbf{0}$ than it is to $\mathbf{b}_i^*$ and to $-\mathbf{b}_i^*$, i.e.

$$r_i < \min\{\|\mathbf{w} - \mathbf{b}_i^*\|, \|\mathbf{w} + \mathbf{b}_i^*\|\}.$$

Calculating this probability leads to the following approximation of p_i

$$p_i \approx \begin{cases} 1 - \frac{2A_{d-i+1}(r_i,h_i)}{A_{d-i+1}(r_i)} & \text{if } R_i < 2r_i \\ 1 & \text{if } R_i \geq 2r_i \end{cases},$$

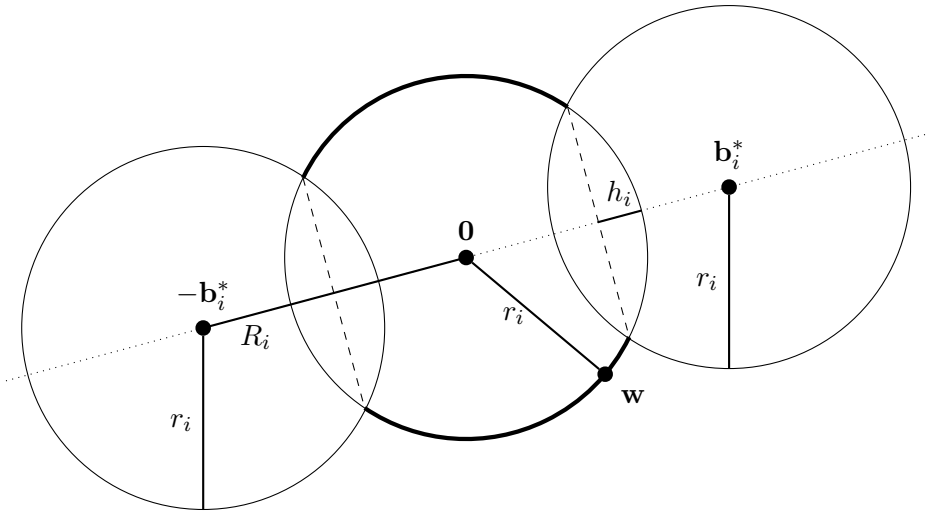


Figure 3.6: Illustration of the success probability p_i in $\mathbb{R}^2$. If $\mathbf{w}$ is on the thick part of the circle, step i of size-reduction is successful.

where $A_{d-i+1}(r_i)$ is the surface area of the sphere in $\mathbb{R}^{d-i+1}$ with radius r_i and $A_{d-i+1}(r_i, h_i)$ is the surface area of the hyperspherical cap of the sphere in $\mathbb{R}^{d-i+1}$ with radius r_i of height h_i with $h_i = r_i - R_i/2$. Using the formulas provided in [Li11], an easy calculation leads to

$$p_i \approx \begin{cases} 1 - \frac{\int_0^{2\frac{h_i}{r_i} - \left(\frac{h_i}{r_i}\right)^2} t^{((d-i)/2)-1}(1-t)^{-1/2}dt}{B(\frac{d-i}{2}, \frac{1}{2})} & \text{if } R_i < 2r_i \\ 1 & \text{if } R_i \geq 2r_i \end{cases}, \tag{3.6}$$

where $B(\cdot, \cdot)$ denotes the Euler beta function. Note that if we assume equality holds in (3.2), the success probability p only depends on the block size β and not on the specific lattice dimension, volume of the lattice, or the length of the unique short vector. Indeed, assuming equality in (3.2), the ratios between the predicted norms $\|\pi_{d-\beta+1-j}(\mathbf{v})\|$ and $\|\mathbf{b}^*_{d-\beta+1-j}\|$ only depend on β for all $j = 1, 2, \ldots$, since

$$\begin{aligned} \frac{r_{d-\beta+1-j}}{R_{d-\beta+1-j}} &= \frac{\|\pi_{d-\beta+1-j}(\mathbf{v})\|}{\|\mathbf{b}^*_{d-\beta+1-j}\|} \\ &= \frac{\frac{\sqrt{\beta}\sqrt{\beta+j}}{\sqrt{\beta}\sqrt{d}}\|\mathbf{v}\|}{\delta^{2(\beta+j)-d-1}\operatorname{Vol}(\Lambda)^{\frac{1}{d}}} \\ &= \frac{\frac{\sqrt{\beta+j}}{\sqrt{\beta}}\delta^{2\beta-d-1}\operatorname{Vol}(\Lambda)^{\frac{1}{d}}}{\delta^{2(\beta+j)-d-1}\operatorname{Vol}(\Lambda)^{\frac{1}{d}}} \\ &= \frac{\sqrt{\beta+j}}{\sqrt{\beta}}\,\delta^{-2j}. \end{aligned}$$

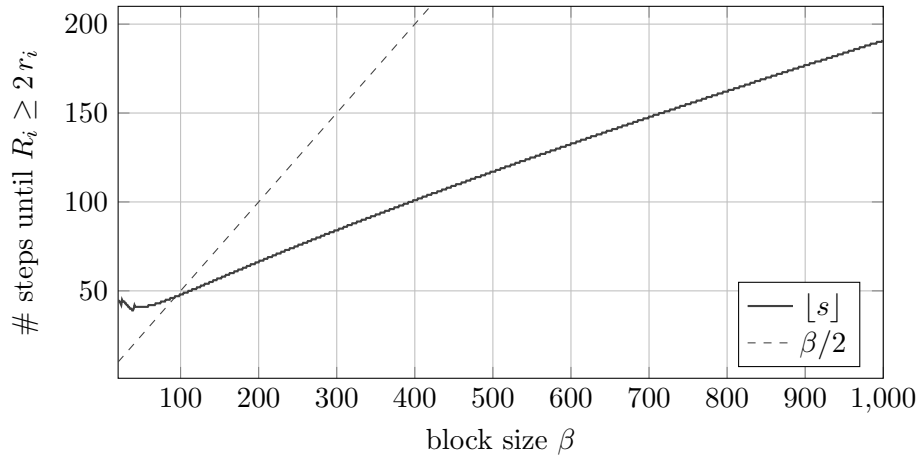


Figure 3.7: Number of size-reduction steps where $P[E_i] < 1$, as a function of β.

By (3.6), the estimated success probability only depends on the $\frac{h_i}{r_i} = 1 - \frac{1}{2}\frac{R_i}{r_i}$ and on $d - i$, for $i \leq d - \beta$. The size-reduction steps are indexed by $i = d - \beta, \ldots, 1$. This means that p_i depends on $d - i = \beta, \beta + 1, \ldots$, until index i such that $R_i \geq 2\,r_i$, from which point onward $P[E_i] = 1$. The number of steps of size-reduction where $P[E_i] < 1$ is then $\lfloor s \rfloor$ such that $R_{d-\beta+1-s} = 2\,r_{d-\beta+1-s}$, and depends only on β since $R_{d-\beta+1-s} = 2\,r_{d-\beta+1-s} \Leftrightarrow \sqrt{\beta} = 2\sqrt{\beta + s}\,\delta^{-2s}$. As long as $d - \beta + 1 - \lfloor s \rfloor > 0$ (that is, index i such that $P[E_i] = 1$ is reached during size-reduction), p is therefore independent of d. In Figure 3.7 we plot the number of steps $\lfloor s \rfloor$ as a function of the block size, and show that for cryptanalytic block sizes $\lfloor s \rfloor < \beta/2$, meaning that likely for any embedding lattice $d - \beta + 1 - \lfloor s \rfloor < d$ and $P[E_1 \wedge \cdots \wedge E_{k-1}]$ only depends on β.

Estimated success probabilities p for different block sizes β are plotted in Figure 3.8. The prediction given in Figure 3.8 is in line with the measured probability of finding $\mathbf{v}$ in the same step when its projection $\pi_{d-\beta+1}(\mathbf{v})$ is found, as reported in Table 3.1 for $\beta = \beta_{2016}$ and $m = m_{2016}$. Since these probabilities were already very close to 1, we added three more experimental data points by running our code on parameter sets targetting smaller block sizes ($q = 97$, $\sigma = 1$, varying n, m). We can see that experimentally the success probability seems to grow slower than our formula predicts. However, it should be noticed that our formula is plotted also for small block sizes, where lattice heuristics are known not to hold [GN08b, §4.2], [CN11, §6.1].

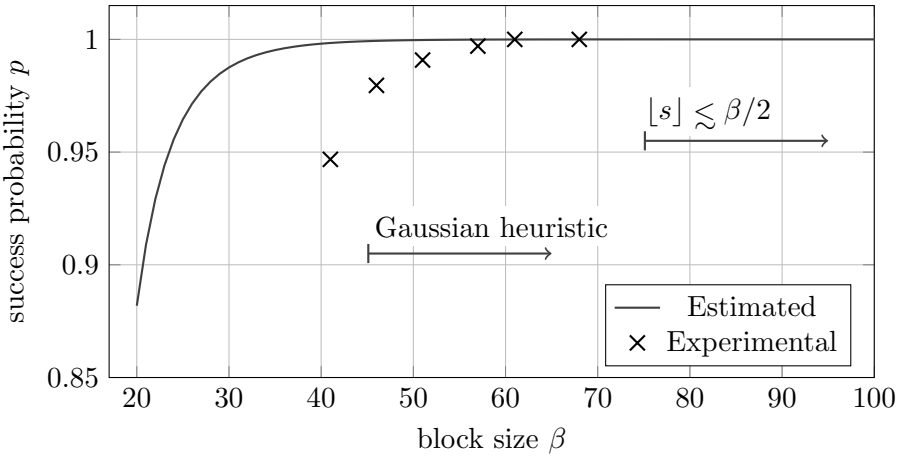


Figure 3.8: Estimated and experimentally measured success probability p for varying block sizes β, assuming β is chosen minimal such that (3.2) holds. The three rightmost data points are from column "same step, $\kappa = d-\beta+1$" of Table 3.1, the three leftmost points were generated using parameters targetting smaller block sizes by picking $q = 97$, $\sigma = 1$ and varying n, m.

Finally, note that by the above analysis we do not expect to recover $\mathbf{v}$ from a projection $\pi_{d-\gamma+1}(\mathbf{v})$ for some small $\gamma \ll \beta$ except with small probability. Indeed, if we were to follow the analysis of Claim 1 starting at index $k = d-\gamma+1$, at step $i = d - \gamma$ of size-reduction we would need $\pi_{d-\gamma}(\mathbf{v})$ being the shortest vector in $L_{d-\gamma} = \mathbb{Z}\mathbf{b}^*_{d-\gamma} + \pi_{d-\gamma}(\mathbf{v})$. However, by definition of the *second* intersection $\|\pi_i(\mathbf{v})\| > \|\mathbf{b}^*_i\|$ for indices $i \in [d - \beta + 2, d - \gamma]$ (c.f. Figure 3.3). This means that likely shorter vectors than $\pi_i(v)$ are present in L_i for some such index i, stopping size-reduction from recovering $\mathbf{v}$.

Smaller block sizes. In Section 3.2.2, we explained the reasoning behind the 2016 estimate. In such model, Condition (3.2) provides a clear cut answer to what is the smallest viable block size to solve uSVP. In practice however, BKZ 2.0 is a randomised algorithm working on random uSVP instances. While our experiments indicate the overall validity of the 2016 estimate, this does not incorporate the probabilistic nature of the computational problem. In Chapter 4 we will investigate this issue, expanding on work initiated by Dachman-Soled *et al.* [DDGR20] for predicting the expected behaviour of Progressive BKZ, and will produce simulator algorithms that allow to explain the observed success probability of smaller block sizes at solving uSVP.

3.4 Applications

Section 3.3 indicates that (3.2) is a reliable indicator for when lattice reduction will succeed in recovering an unusually short vector. Furthermore, as illustrated in Figure 3.1, applying (3.2) lowers the required block sizes compared to the 2008 model which is heavily relied upon in the literature. Thus, in this section we evaluate the impact of applying the revised estimates to various parameter sets from the literature. Indeed, for many schemes we find that their parameters need to be adapted to maintain the level of security claimed at the time of their publication.

Many of the schemes considered below feature an unusually short secret $\mathbf{s}$ where $s_i \leftarrow U(\{-B, \ldots, B\})$ for some small $B \in \mathbb{Z}$. Furthermore, some schemes pick the secret to also be sparse such that most components of $\mathbf{s}$ are zero. Thus, before we apply the revised 2016 estimate, we briefly recall the alternative embedding due to Bai and Galbraith [BG14b] which takes these small (and sparse) secrets into account.

3.4.1 Bai and Galbraith's embedding

Consider an LWE instance in matrix form $(\mathbf{A}, \mathbf{b}) \equiv (\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$. By inspection, it can be seen that the vector $(\nu\,\mathbf{s} \mid \mathbf{e} \mid c)$, for some non-zero ν and c, is contained in the lattice

$$\Lambda = \left\{ \mathbf{x} \in (\nu\mathbb{Z})^n \times \mathbb{Z}^m \times (c\mathbb{Z}) \;\mid \mathbf{x} \cdot \left(\frac{1}{\nu}\mathbf{A} \mid \mathbf{I}_m \mid -\frac{1}{c}\mathbf{b} \right)^t \equiv \mathbf{0} \bmod q \right\}, \quad (3.7)$$

where ν allows to balance the size of the secret and the noise. An $(n+m+1) \times (n+m+1)$ basis $\mathbf{B}$ for Λ can be constructed as

$$\mathbf{B} = \begin{pmatrix} \mathbf{0} & q\mathbf{I}_m & \mathbf{0} \\ \nu\mathbf{I}_n & -\mathbf{A}^t & \mathbf{0} \\ \mathbf{0} & \mathbf{b}^t & c \end{pmatrix}. \quad (3.8)$$

We can see that $\mathbf{B}$'s integer row-span is a sublattice of Λ by noting that

$$\begin{pmatrix} \mathbf{0} & q\mathbf{I}_m & \mathbf{0} \\ \nu\mathbf{I}_n & -\mathbf{A}^t & \mathbf{0} \\ \mathbf{0} & \mathbf{b}^t & c \end{pmatrix} \left(\frac{1}{\nu}\mathbf{A} \mid \mathbf{I}_m \mid -\frac{1}{c}\mathbf{b} \right)^t = \left(q\mathbf{I}_m \mid \mathbf{A}^t - \mathbf{A}^t \mid \mathbf{b} - \mathbf{b} \right)^t \equiv \mathbf{0} \bmod q.$$

Therefore, given any basis $\mathbf{C} \in \mathbb{R}^{(n+m+1)\times(n+m+1)}$ of Λ, we must have $\mathbf{B} = \mathbf{MC}$ for some integer matrix $\mathbf{M}$. The dimensions of $\mathbf{B}$ imply that $\mathbf{M}$ is also $(n+m+1)\times(n+m+1)$. Finally, we can see that $\mathbf{B}$ is exactly a basis for Λ (and not for a strict sublattice), by using the fact that $|\det(\mathbf{B})| = |\det(\mathbf{M})| \cdot |\det(\mathbf{C})|$, and that $|\det(\mathbf{B})| = \mathrm{vol}(\Lambda) = |\det(\mathbf{C})|$ by direct calculation [MR09], implying that $\mathbf{M}$ is unimodular. Using this basis we can find our target vector as $(* \mid \mathbf{s} \mid 1) \cdot \mathbf{B} = (\nu\,\mathbf{s} \mid \mathbf{e} \mid c)$, for suitable values of $*$.

Small secrets. If $\mathbf{s}$ is small and/or sparse such that its coefficients' distribution χ_s is narrower than the error distribution χ_e, by choosing $\nu = c = 1$ the vector $(\mathbf{s} \mid \mathbf{e} \mid 1)$ is unbalanced, i.e. $\frac{\|\mathbf{s}\|}{\sqrt{n}} \ll \frac{\|\mathbf{e}\|}{\sqrt{m}} \approx \sigma$, where σ is the standard deviation of χ_e. We may then want to rebalance it by choosing an appropriate value of ν such that $\|(\nu\,\mathbf{s} \mid \mathbf{e} \mid 1)\| \approx \sigma\sqrt{n+m}$. Rebalancing preserves $(\nu\,\mathbf{s} \mid \mathbf{e} \mid 1)$ as the unique shortest vector in the lattice, while at the same time increasing the volume of the lattice being reduced, hence reducing the block size required by (3.2) when compared to Kannan's embedding (3.1). The same reasoning could also be used in favour of setting $c = \sigma$ as embedding coefficient.

If $\mathbf{s} \xleftarrow{\$} \{-1, 0, 1\}^n$ we expect $\|\nu\,\mathbf{s}\|^2 \approx \frac{2}{3}\nu^2 n$. Therefore, we can chose $\nu = \sqrt{\frac{3}{2}}\sigma$ to obtain $\|\nu\,\mathbf{s}\| \approx \sigma\sqrt{n}$, so that $\|(\mathbf{s} \mid \mathbf{e} \mid 1)\| \approx \sigma\sqrt{n+m}$. Similarly, if exactly $w < n$ entries of $\mathbf{s}$ are non-zero from $\{-1, 1\}$, we have $\|\nu\,\mathbf{s}\|^2 = w\,\nu^2$. Choosing $\nu = \sqrt{\frac{n}{w}}\sigma$, we obtain a vector $\nu\,\mathbf{s}$ of length $\sigma\sqrt{n}$. In general, assuming a secret distribution χ_s with mean $\mathbb{E}(\chi_s) = 0$, we can compute

$$\mathbb{E}(\|\nu\mathbf{s}\|^2) = n \cdot \nu^2 \cdot \mathbb{E}(s_i^2) = n \cdot \nu^2 \cdot \mathbb{V}(\chi_s).$$

In order to balance such a secret vector, the optimal scaling factor can be deduced as $\nu = \sqrt{\mathbb{V}(\chi_e)/\mathbb{V}(\chi_s)}$.

Non-integer scaling factors. In theory, the optimal value of ν could be any real not smaller than 1. In practice however, lattice reduction libraries such as Fplll [DT17] require input bases to have integer coefficients. In the experimental setting, this issue can be avoided by "clearing denominators". The idea is to use a rational approximation $\nu \approx x/y$, with $x, y \in \mathbb{Z}_{\geq 1}$. Then,

one can define a basis $\mathbf{B}_1$ obtained by clearing the denominator of ν

$$\mathbf{B}_1 = \begin{pmatrix} \mathbf{0} & yq\mathbf{I}_m & \mathbf{0} \\ x\mathbf{I}_n & -y\mathbf{A}^t & \mathbf{0} \\ \mathbf{0} & y\mathbf{b}^t & yc \end{pmatrix} \approx y \cdot \mathbf{B}.$$

This has the effect of scaling every lattice vector in $\Lambda(\mathbf{B})$ by $y \geq 1$, keeping the geometry unaltered while resulting in an integer basis.

Centering distributions. In the case of secret distributions with non-zero mean μ_s, two simple approaches can be used to generate an embedding with a target vector containing a recentered version of $\mathbf{s}$. This can be useful since it allows for a more aggressive choice of ν. For example, this is what we assume would be done by an attacker when we investigate the cost of solving uSVP with binary secrets. The first approach is to map any LWE samples $(\mathbf{A}, \mathbf{b})$ into samples $(\mathbf{A}, \mathbf{b} - \mathbf{A}\boldsymbol{\mu_s})$, where $\boldsymbol{\mu_s} = (\mu_s, \ldots, \mu_s)^t$. This works since

$$(* \mid \mathbf{s} - \boldsymbol{\mu_s} \mid 1) \cdot \begin{pmatrix} \mathbf{0} & q\mathbf{I}_m & \mathbf{0} \\ \nu\mathbf{I}_n & -\mathbf{A}^t & \mathbf{0} \\ \mathbf{0} & \mathbf{b}^t - \boldsymbol{\mu_s}^t\mathbf{A}^t & c \end{pmatrix} = (\nu\,(\mathbf{s} - \boldsymbol{\mu_s}) \mid \mathbf{e} \mid c).$$

Recovering the target vector on the right hand side results in solving the original LWE instance, while the first n coefficients in the target vector are now centred around 0 rather than μ_s. For example, applying this method with $\nu = 2$ to a binary secret, i.e. one sampled from $U(\{0, 1\})$, means the first n coefficients of the target vector will be distributed uniformly in the set $\{-1, 1\}$. The second approach for centring the secret distribution is to use the basis

$$(* \mid \mathbf{s} \mid 1) \cdot \begin{pmatrix} \mathbf{0} & q\mathbf{I}_m & \mathbf{0} \\ \nu\mathbf{I}_n & -\mathbf{A}^t & \mathbf{0} \\ -\nu\boldsymbol{\mu_s} & \mathbf{b}^t & c \end{pmatrix} = (\nu\,(\mathbf{s} - \boldsymbol{\mu_s}) \mid \mathbf{e} \mid c).$$

In cases where error distribution has mean $\mu_e \neq 0$, one can center the error distribution by mapping samples $(\mathbf{A}, \mathbf{b})$ into samples $(\mathbf{A}, \mathbf{b} - \boldsymbol{\mu_e})$.

Of course, one can center error and secret distributions at the same time, if needed. For example, if χ_e has mean μ_e and χ_s has mean μ_s, mapping samples $(\mathbf{A}, \mathbf{b})$ into $(\mathbf{A}, \mathbf{b} - \boldsymbol{\mu_e} - \mathbf{A}\boldsymbol{\mu_s})$ will result in a centered shortest embedded vector. An integer basis can be obtained by appropriately clearing the denominators of any rational approximations of ν, μ_e and μ_s.

Sparse secrets. In the case of sparse secrets, combinatorial techniques can also be applied [How07, BGPW16, Alb17]. Given a secret $\mathbf{s}$ with at most $w < n$ non-zero entries, we guess k entries of $\mathbf{s}$ to be 0, therefore decreasing the dimension of the lattice to consider. For each guess, we then apply lattice reduction to recover the remaining components of the vector $(\mathbf{s} \mid \mathbf{e} \mid 1)$. Therefore, when estimating the overall complexity of solving such instances, we find k minimising $C(n-k)/p_k$, where $C(n)$ is the cost of solving uSVP for a lattice of dimension n and p_k is the probability of guessing correctly. We note that after any coefficient guessing is applied, the target vector is a lower dimensional $(\mathbf{s}' \mid \mathbf{e}' \mid 1)$. It may then be optimal to compute the scaling factor ν and the recentering strategy using the distributions $\chi_{s'}$ and $\chi_{e'}$ of the coefficients of $\mathbf{s}'$ and $\mathbf{e}'$ respectively, since these may differ from χ_s and χ_e.

3.4.2 Estimates

In what follows, we assume that the geometry of (3.7) is sufficiently close to that of (3.1) so that we transfer the analysis as is. Furthermore, we will denote applying (3.2) from [ADPS16] for Kannan's embedding as "Kannan" and applying (3.2) for Bai and Galbraith's embedding [BG14b] as "Bai-Gal". Unless stated otherwise, we will assume that calling BKZ with block size β in dimension d costs $8\,d\,2^{0.292\,\beta+16.4}$ operations [BDGL16, Alb17].

Lizard. Proposed in [CKLS16, CKLS18], Lizard is a PKE scheme based on the Learning With Rounding problem using small, sparse secrets. The authors provide a reduction to LWE, and security parameters against classic and quantum adversaries, following their analysis. In particular, they cost BKZ by a single call to sieving on a block of size β. They estimate this call to cost $\beta\,2^{\gamma\,\beta}$ CPU cycles where $\gamma = 0.292$ for classical adversaries, $\gamma = 0.265$ for quantum ones and $\gamma = 0.2075$ as a lower bound for sieving ("paranoid"). Applying the revised 2016 cost estimate for the primal attack to the parameters suggested in [CKLS16] (using their sieving cost model as described above) reduces the expected costs, as shown in Table 3.2. We note that after private communication the authors of Lizard have updated their parameters in [CKLS18].

	Classical			Quantum			Paranoid		
n, $\log_2 q$, σ	386, 11, 2.04			414, 11, 2.09			504, 12, 4.20		
Cost	β	d	λ	β	d	λ	β	d	λ
[CKLS16]	418	—	130.8	456	—	129.7	590	—	131.6
Kannan	372	805	117.2	400	873	114.6	567	1120	126.8
Bai-Gal	270	646	88.5	297	692	86.9	372	833	85.9

Table 3.2: Bit complexity estimates λ for solving Lizard PKE [CKLS16] as given in [CKLS16] and using Kannan's resp. Bai and Galbraith's embedding under the 2016 estimate. The dimension of the LWE secret is n. In all cases, BKZ-β is estimated to cost $\beta\, 2^{\gamma\,\beta}$ operations.

HElib. Introduced in [GHS12], HElib is a Fully-Homomorphic Encryption (FHE) library implementing the BGV scheme [BGH13]. Albrecht [Alb17] provides revised security estimates for HElib by employing a dual attack exploiting the small and sparse secret, using the same cost estimate for BKZ as given at the beginning of this section. In Table 3.3 we provide costs for a primal attack using Kannan's and Bai and Galbraith's embeddings. Primal attacks perform worse than the algorithm described [Alb17], but, as expected, under the 2016 estimate the gap narrows.

SEAL. Introduced in [CLP17], SEAL is an FHE library by Microsoft, based on the FV scheme [FV12]. Up to date (at the time of publication) parameters are given in [CLP17], using the same cost model for BKZ as mentioned at the beginning of this section. In Table 3.4, we provide complexity estimates for Kannan's and Bai and Galbraith's embeddings under the 2016 estimate. Note that the gap in solving time between the dual and primal attack reported in [Alb17] is closed for SEAL v2.1 parameters.

TESLA. Described in [BG14a, ABB+17], TESLA is a signature scheme based on LWE. Post-quantum secure parameters in the quantum random oracle model were proposed in [ABB+17]. In Table 3.5, we show that these parameters need to be increased to maintain the currently claimed level of security under the 2016 estimate. Note that [ABB+17] maintains a gap of

80 bit security															
n	1024			2048			4096			8192			16384		
$\log_2 q$, σ	47, 3.2			87, 3.2			167, 3.2			326, 3.2			638, 3.2		
Cost	β	d	λ	β	d	λ	β	d	λ	β	d	λ	β	d	λ
[Alb17] $\text{SILKE}_{\text{sparse}}$	105	—	61.3	111	—	65.0	112	—	67.0	123	—	70.2	134	—	73.1
Kannan	156	2096	76.0	166	4003	79.8	171	7960	82.3	176	15606	84.7	180	31847	86.9
Bai-Gal	137	1944	70.3	152	3906	75.9	163	7753	79.9	169	16053	82.9	173	32003	85.9
128 bit security															
n	1024			2048			4096			8192			16384		
$\log_2 q$, σ	38, 3.2			70, 3.2			134, 3.2			261, 3.2			511, 3.2		
Cost	β	d	λ	β	d	λ	β	d	λ	β	d	λ	β	d	λ
[Alb17] $\text{SILKE}_{\text{sparse}}$	138	—	73.2	145	—	77.4	151	—	81.2	163	—	84.0	149	—	86.4
Kannan	225	2076	96.1	238	4050	100.9	245	8011	103.9	250	16017	106.4	257	31635	109.4
Bai-Gal	189	1901	86.6	211	3830	94.4	204	7348	99.3	185	13543	102.8	204	28236	105.9

Table 3.3: Solving costs for LWE instances underlying HELib as given in [Alb17] and using Kannan's resp. Bai and Galbraith's embedding under the 2016 estimate. The dimension of the LWE secret is n. In all cases, BKZ-β is estimated to cost $8d\,2^{0.292\,\beta+16.4}$ operations.

n, $\log_2 q$, σ	1024, 35, 3.19			2048, 60, 3.19			4096, 116, 3.19			8192, 226, 3.19			16384, 435, 3.19		
Cost	β	d	λ	β	d	λ	β	d	λ	β	d	λ	β	d	λ
[CLP17]	230	—	97.6	282	—	115.1	297	—	119.1	307	—	123.1	329	—	130.5
[Alb17]+	255	—	104.9	298	—	118.4	304	—	121.2	310	—	124.0	328	—	130.2
Kannan	257	2085	105.5	304	4041	120.2	307	8047	122.0	312	15876	124.5	328	31599	130.1
Bai-Gal	237	1984	99.6	288	4011	115.5	299	8048	119.7	309	15729	123.6	326	31322	129.5

Table 3.4: Solving costs for parameter choices in SEAL v2.1 as given in [CLP17], using [Alb17] as implemented in the [APS15] estimator commit `84014b6` ("[Alb17]+"), and using Kannan's resp. Bai and Galbraith's embedding under the 2016 estimate. In all cases, BKZ-β is estimated to cost $8d\,2^{0.292\,\beta+16.4}$ operations.

$\approx \log_2 n$ bits of security between the best known attack on LWE and claimed security to account for a loss of security in the reduction.

	TESLA-0			TESLA-1			TESLA-2		
n, $\log_2 q$, σ	644, 31, 55			804, 31, 57			1300, 35, 73		
Cost	β	d	λ	β	d	λ	β	d	λ
Classical									
[ABB+17]	—	—	110.0	—	—	142.0	—	—	204.0
[ABB+17]+	255	—	110.0	358	—	140.4	563	—	200.9
Kannan	248	1514	102.4	339	1954	129.3	525	3014	184.3
Post-Quantum									
[ABB+17]	—	—	71.0	—	—	94.0	—	—	142.0
[ABB+17]+	255	—	68.5	358	—	90.7	563	—	136.4
Kannan	248	1415	61.5	339	1954	81.1	525	3014	122.4

Table 3.5: Bit complexity estimates for solving TESLA parameter sets [ABB+17]. The entry "[ABB+17]+" refers to reproducing the estimates from [ABB+17] using a current copy of the estimator from [APS15] which uses $c = 1$ instead of $c = \|\mathbf{e}\|$. As a consequence the values in the respective rows are slightly lower than in [ABB+17]. We compare with Kannan's embedding under the 2016 estimate, Bai and Galbraith's embedding is not necessary since TESLA uses normal form LWE. Classically, BKZ-β is estimated to cost $8d\,2^{0.292\,\beta+16.4}$ operations; quantumly BKZ-β is estimated to cost $8d\,\sqrt{\beta^{0.0225\,\beta}\cdot 2^{0.4574\,\beta}/2^{\beta/4}}$ operations in [ABB+17].

BCIV17. [BCIV17] is a somewhat homomorphic encryption scheme obtained as a simplification of the FV scheme [FV12] and proposed as a candidate for enabling privacy friendly energy consumption forecast computation in smart grid settings. The authors propose parameters for obtaining 80 bits of security, derived using the estimator from [APS15] available at the time of their publication. As a consequence of applying (3.2), we observe a moderate loss of security, as reported in Table 3.6.

3.5 Conclusions

In this chapter we have experimentally compared the two approaches [GN08b, ADPS16] used in the lattice-based cryptography literature to choose the block

80 bit security $n = 4096$, $\log_2 q = 186$, $\sigma = 102$							
Embedding	β	d	λ	Embedding	β	d	λ
Kannan	156	8105	77.9	Bai-Gal	147	7818	75.3

Table 3.6: Solving costs for proposed Ring-LWE parameters in [BCIV17] using Kannan's resp. Bai and Galbraith's embedding under the 2016 estimate. In both cases, BKZ-β is estimated to cost $8d\,2^{0.292\,\beta+16.4}$ operations.

size when solving the unique-SVP problem using BKZ. We have verified that the estimate proposed in [ADPS16] closely matches our observations, which means that lattice reduction attacks will be cheaper than previously predicted (cf. Figure 3.1). Hence, we have re-estimated the security of a few encryption and signature schemes proposed in the literature to measure this impact.

However, we have also observed that [ADPS16]'s success condition for BKZ, (3.2), does not capture entirely the probabilistic nature of the attack; e.g. slightly smaller block sizes than predicted have non-negligible probability of solving uSVP (c.f. Table 3.1). In the next chapter we will extend the recent techniques introduced by Dachman-Soled *et al.* [DDGR20] to try and compute the exact probability that a given block size successfully solves uSVP.

3.5.1 Developments since publication

Since this chapter was published as [AGVW17], a few related works have appeared further analysing, using or extending the model from [ADPS16]. First, our changes to the LWE estimator [APS15] were integrated in the estimator's code base, meaning that security estimates generated with it, such as those in the Homomorphic Encryption Security Standard [ACC+18], automatically adopted the 2016 estimate. At Africacrypt 2019, Bai *et al.* [BMW19] published more experiments further confirming the assumptions used in the 2016 estimate, and confirming that the second intersection between the basis profile and the norms of the target vector's projections discussed in Section 3.3.3 should not affect cryptographic parameters. Finally, Dachman-Soled *et al.* [DDGR20] extended the 2016 model to provide a probabilistic estimator of the success

probability of solving uSVP using Progressive BKZ that returns precise predictions also in the small block size regime. This became the basis for Chapter 4 of this thesis (published as [PV21]), where we modify Dachman-Soled *et al.*'s simulator to output more fine-grained results on the success probability of solving uSVP, verify the validity of the approach experimentally, and investigate the accuracy of the 2016 model and its extensions when applied to LWE with non-Gaussian secret and error distributions.

Acknowledgements. We thank Léo Ducas and Rachel Player for helpful discussions.

Chapter 4

On the Fine-Grained Cost of Solving uSVP

Contents

As lattice-based key encapsulation, digital signature, and fully homomorphic encryption schemes near standardisation, ever more focus is being directed to the precise estimation of the security of these schemes. The primal attack reduces key recovery against such schemes to instances of the unique Shortest Vector Problem (uSVP). Dachman-Soled et al. *(Crypto 2020) recently proposed a new approach for fine-grained estimation of the cost of the primal attack when using Progressive BKZ for lattice reduction. In this chapter we review and extend their technique to BKZ 2.0 and provide extensive experimental evidence of its accuracy. Using this technique we also explain results from the primal attack experiments presented in Chapter 3, where attacks often succeeded with smaller than expected block sizes. Finally, we use our simulators to re-estimate*

the cost of attacking the three lattice KEM finalists of the NIST Post Quantum Standardisation Process.

4.1 Motivation

A popular computational problem chosen to design lattice-based schemes is the Learning With Errors (LWE) problem (with its ring and module variants). As mentioned in Section 1.4.3, a variety of attack strategies against this problem exist, with the practically better performing being the *primal*, *dual* and *hybrid* attacks. All three rely on lattice reduction algorithms, such as BKZ [SE91, SE94, CN11], Progressive BKZ [AWHT16], Self-Dual BKZ [MW16], G6K [ADH+19] and Slide Reduction [GN08a], to find either a unique (up to sign) embedded shortest vector, or more generally a good lattice basis. In particular, the primal attack is often estimated as the cheapest option [ACD+18].

The primal attack against LWE consists of using lattice reduction to solve an instance of the unique Shortest Vector Problem (uSVP). The most popular lattice reduction algorithm is BKZ. In Chapter 3, we discussed how compliexity estimates for solving uSVP directly depend on estimating the smallest block size β such that BKZ-β successfully recovers the unique shortest vector. This β is commonly found by following the methodology introduced in [ADPS16], which we experimentally investigated in Chapter 3.

While we confirmed the overall validity of the approach in [ADPS16], in our experiments reported in Table 3.1, we noticed that smaller than expected block sizes can result in a non-negligible probability of solving uSVP instances arising from the primal attack, when using BKZ. The same phenomenon was later observed in similar experiments run by Bai *et al.* [BMW19]. Some concerns were raised [BCLv19] that this could indicate an overestimate of the complexity of the primal attack for cryptographically sized instances. Furthermore, the experiments carried out in Chapter 3 only focused on recovering a unique shortest vector sampled coefficient-wise from a discrete Gaussian distribution. While we claimed that the [ADPS16] methodology would also hold for binary and ternary distributions, we did not provide experimental evidence. Recent

Input: LLL reduced lattice basis $\mathbf{B}$ of rank n
Input: $\tau \in \mathbb{Z}^+$
1 $\beta \leftarrow 3$
2 **while** $\beta \leq n$ **do** `/* round */`
3 run τ tours of BKZ-β on basis $\mathbf{B}$
4 $\beta \leftarrow \beta + 1$

Algorithm 7: Progressive BKZ Algorithm, as used in this chapter.

work [CCLS20] revisited the binary and ternary case in the small block size regime $\beta \leq 45$ and concluded that discrete Gaussian errors are more secure. We disagree, and discuss [CCLS20] further in Section 4.4.2.

Dachman-Soled *et al.* [DDGR20] recently proposed an approach for estimating the complexity of the primal attack that makes use of probability distributions for the norms of particular projections of the unique shortest vector, rather than only expected values. This results in a new approach that allows one to better predict the behaviour of the attack when considering block sizes smaller than those expected to be successful by the [ADPS16] methodology. The authors of [DDGR20] use this approach to develop a simulator that predicts the expected block size by which Progressive BKZ (PBKZ) [AWHT16] will solve an isotropic uSVP instance, that is an instance where the shortest vector (up to sign) in the lattice is spherically distributed, as if sampled from a Gaussian distribution with standard deviation $\sigma = 1$. In this work, we call such a simulator a *uSVP simulator*. They use this uSVP simulator in the setting of solving LWE instances with extra hints about the secret, and verify the accuracy of their predictions as the number of hints varies.

For the purposes of this chapter, we define Progressive BKZ as in Algorithm 7. Progressive BKZ consists in running τ tours of BKZ-β for progressively larger block sizes β, until $\beta = n$. We call each step of the loop on Line 2 a "round" of PBKZ, to distinguish them from the "tours" run inside the BKZ subroutine. In our experiments we do not let β increase further than some $\beta_{\max} < n$. When using PBKZ to solve uSVP, at the end of each round we check for whether a solution was found, and in case break out of the loop before reaching $\beta = \beta_{\max}$.

Our contributions. Our first contribution is the implementation of a variant of the uSVP simulator for Progressive BKZ, and the development of a new uSVP simulator for BKZ 2.0. Rather than only returning the expected successful block size, we extract full probability mass functions for successful block sizes, which allow for a more direct comparison to experimental results. Our simulators are also faster than that in [DDGR20], simulating success probabilities for Kyber1024 in 31 seconds against the 2 hours of [DDGR20]. This allows for potentially easier inclusion in parameter selection scripts, such as the LWE estimator [APS15]. We note that since the time of writing, the latest version of the simulator proposed in [DDGR20] adopted some of the techniques we use for the speed-up.

Our second contribution is extensive experiments on the success probability of different block sizes for BKZ 2.0 and Progressive BKZ, on uSVP lattices generated from LWE instances with discrete Gaussian, binary or ternary secret and error distributions. Our experiments show that the uSVP simulators accurately predict the block sizes needed to solve uSVP instances via lattice reduction for all distributions tested, and further explain the phenomenon of smaller-than-expected block sizes solving uSVP noticed in Chapter 3.

As a final contribution, we re-estimate the security of the three lattice KEM finalists of the NIST PQC process [Nat16] using our uSVP simulators. We compare the expected block sizes they return to those predicted by the original methodology of [ADPS16]. We note that our uSVP simulators estimate that a slightly larger average block size than predicted is required, meaning that [ADPS16] likely resulted in an underestimate of their security.[1] We also observe that this phenomenon can, in large part, be attributed to the original [ADPS16] methodology using the Geometric Series Assumption. Replacing this assumption with the output of the [CN11] BKZ simulator reduces the predictive gap between the [ADPS16] methodology and our uSVP simulators.

All of our code and data can be found at `github.com/fvirdia/usvp-simulation`.

[1] A similar phenomenon had also been observed in [DDGR20] for NTRU-HPS.

Related work. The Geometric Series Assumption (GSA), used to predict the output quality of lattice reduction, was introduced in [Sch03]. A simulator, specifically for the output quality of BKZ, was introduced in [CN11]. This simulator more accurately predicts the final, or *tail*, region of the basis profile of a BKZ reduced lattice, improving over the GSA. A refined BKZ simulator was presented in [BSW18], which improves over the [CN11] simulator in the first region, or *head*, of the basis profile. Alkim *et al.* [ADPS16] introduced a BKZ specific method for estimating the block size required to solve uSVP instances arising from the primal attack; its accuracy was investigated in [AGVW17, BMW19] (the first being the published version of Chapter 3). This method, combined with basis profile simulation after BKZ reduction and arguments about distributions describing the norms of projections of the unique short vector, is extended in [DDGR20] to predict the expected block size by which Progressive BKZ will solve isotropic uSVP instances.

Chapter roadmap. In Section 4.2 we review the available methods for simulating the profiles of BKZ- and LLL-reduced lattice bases with more accuracy than the GSA provides. In Section 4.3 we review the approach of [DDGR20] and use it to propose uSVP simulators for BKZ 2.0 and Progressive BKZ. In Section 4.4 we describe our experiments and results. In Section 4.5 we use our uSVP simulators to provide preliminary estimates of the block sizes required to successfully perform key recovery attacks on the three NIST PQC lattice KEM finalists, and compare this to predictions using the [ADPS16] methodology.

4.2 Simulating BKZ and LLL

In Chapter 1 we introduced the Geometric Series Assumption (GSA), a heuristic that suggests that the shape of a reduced lattice basis profile follows a geometric progression, where the common ratio depends on the lattice reduction algorithm used. Given a lattice Λ, the GSA together with the constraint that $\prod \|\mathbf{b}_i^*\| = \mathrm{vol}(\Lambda)$ can be used to approximate the profile of a reduced basis for Λ.

As we mentioned when introducing the GSA, its output represent only a first approximation of a reduced basis profile. In this section we discuss some simulators that can be used to more accurately describe the profile of BKZ- and LLL-reduced lattice bases.

BKZ. The GSA's output can be seen as a global view of a reduced lattice basis, using only the constant volume of the full lattice Λ to estimate the basis profile. However, the volume of local *blocks* is not constant as LLL or BKZ is run on a basis. Chen and Nguyen propose a *BKZ simulator* [CN11] that takes this intuition into account to improve on the GSA in the case of BKZ. It takes as *input* a basis profile $\{\|\mathbf{b}_i^*\|^2\}_i$ and simulates a tour of BKZ-β by calculating, block by block, the Gaussian heuristic of the current block, "inserting" a vector of that length at the beginning of said block, and redistributing the necessary length to the subsequent Gram–Schmidt vectors to keep $\mathrm{vol}(\Lambda)$ constant. Since projected sublattices of small rank, e.g. smaller than 45, do not behave as random lattices [GN08b, CN11], in order to simulate the profile for the final indices of the basis the BKZ simulator stops using the Gaussian heuristic and instead uses experimentally generated average norms for unit volume lattices (scaled appropriately). This design also allows for one to simulate a fixed number of tours, rather than assuming convergence, as in the GSA. In practice, the [CN11] simulator better captures the shape of a reduced basis profile over the last few indices.

The simulation process can be made probabilistic by "inserting" a vector with length drawn from a probability distribution centred on the length suggested by the Gaussian heuristic. This is done by Bai *et al.* [BSW18], whose simulator further improves the predictions of [CN11] by better capturing the shape of reduced basis profiles over the first few indices.

Throughout our work we make use of the Chen–Nguyen simulator as implemented in Fpylll [FPY17]. In Algorithm 8 we define a BKZSim subroutine that returns a [CN11] simulation for an input basis profile. Here $\mathrm{LWE}_{n,q,\chi,m}$ is a basis produced as in (3.8) with $c = 1$, assuming normal form so that $\nu = 1$ and $\chi = \chi_s = \chi_e$.

Input: (n, q, χ, m) or profile $\{\|\mathbf{b}_i^*\|^2\}_i$
Input: β, τ
if $\{\|\mathbf{b}_i^*\|^2\}_i$ *not provided as input* **then**
  $\{\|\mathbf{b}_i^*\|^2\}_i \leftarrow$ simulated profile of LLL reduced $\mathrm{LWE}_{n,q,\chi,m}$ instance
$\{\|\mathbf{b}_i^*\|^2\}_i \leftarrow$ [CN11] simulation of τ tours of BKZ-β on $\{\|\mathbf{b}_i^*\|^2\}_i$
return $\{\|\mathbf{b}_i^*\|^2\}_i$

Algorithm 8: BKZSim subroutine.

LLL. As part of our uSVP simulations, we will require in input the profile of LLL-reduced bases. To produce these we considered three options. We compare the output for the three approaches in Figure 4.1.

The first option is to run LLL on an example basis. In the case of the instances used in the experiments which we will be describing in Section 4.4, such a reduction can be easily performed on any particular embedding basis. However, this is not the case for cryptographically sized embeddings, where Fplll's implementation of LLL can only run with high enough floating point precision by using MPFR [FHL+07], which becomes impractically slow.

The second option is to use a GSA slope corresponding to LLL reduction, by setting $\alpha = \delta^{-2} = 1.02^{-2}$ as the GSA factor. This correctly predicts the slope of the main section of the profile, but does not account for the role played by the q-vectors[2] in the embedding basis, which are short enough to not be affected by LLL [How07], resulting in the characteristic "Z-shape" of LLL-reduced bases for q-ary lattices.

The third option is to use a specific basis profile simulator for LLL that captures the effect of the q-vectors. We opt for the third option. While the approach to simulating the Z-shape immediately follows from the observation that the vectors in the middle of the basis follow the GSA [How07], we provide below a full description of how we do it.

[2] That is, vectors with one coefficient valued q and all the others zero.

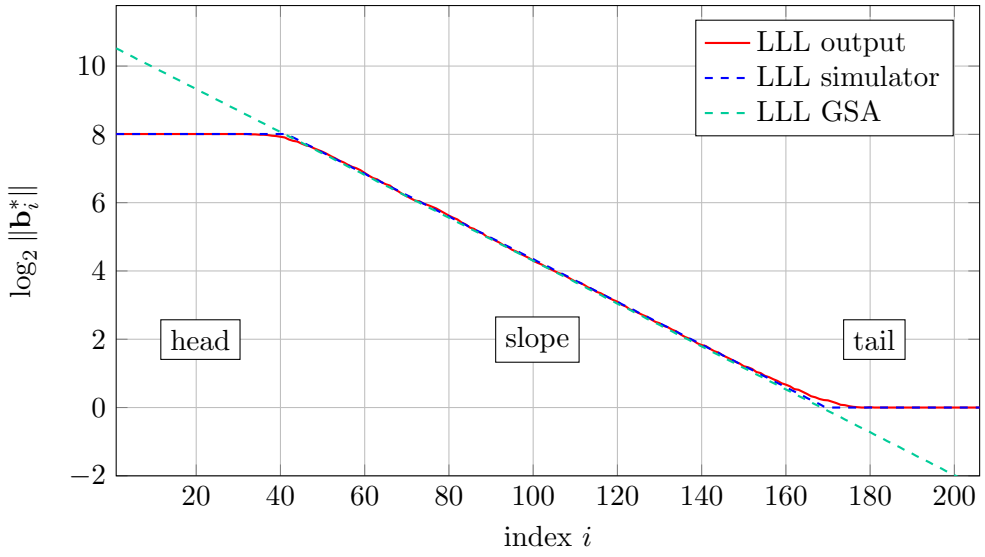


Figure 4.1: Comparison between the output profile of LLL averaged over 25 input bases, the output of the LLL simulator used for our estimates, and the GSA. The input bases being reduced are for q-ary lattices corresponding to embeddings of "$n = 100$" LWE instances as parametrised in Table 4.1

4.2.1 LLL "Z-shape" simulation

The Z-shape nickname refers to the shape of the log-plot for the profile of an LLL-reduced basis $\mathbf{B}$ when providing in input a q-ary lattice basis such as (3.8), with the q-vectors set as the first basis vectors.[3] In such cases, most of the q-vectors will not be altered by LLL, since they are orthogonal and short. This results in the basis profile having a flat *head* corresponding to the first Gram–Schmidt vectors $\mathbf{b}_1^*$, $\mathbf{b}_2^*, \ldots$ being q-vectors. Depending on the lattice's volume and rank, the final Gram–Schmidt vectors will be 1-vectors obtained from the identity matrix minor in the basis, resulting in a flat *tail* in the profile. The middle indices of the log-plot of the basis profile will be located along a straight line with the slope predicted by the GSA for LLL with $\log \alpha = -2 \log \delta$, where δ is LLL's root-Hermite factor $\delta \approx 1.02$ [NS06]. An example of the Z-shape can be seen in Figure 4.1.

[3] While a similar Z-shaped profile will result even if the q-vectors are not at the beginning of the basis, the effect will be more pronounced if they are.

In the most straightforward case, given a normal-form LWE lattice with volume q^m, dimension d and basis (3.8), the LLL simulator predicts the Z-shape by first computing the GSA slope section of the profile. This is achieved by noticing that vectors in this section will have log-norm $\log \|\mathbf{b}_i^*\| \in (0, \log q)$, decreasing by $\log \alpha$ at each index by the GSA. Then the head section will have enough q-vectors so that the output profile describes a lattice with volume q^m, and the remaining vectors will be 1-vectors in the tail. This procedure may result in a volume that is not exactly equal to q^m. In this case, we pick the maximum number of q-vectors such that the implied volume is $< q^m$, and shift the slope up to match q^m. In practice this effect is minimal. This description matches all cases used in this chapter, the resulting pseudo-code can be found in Algorithm 9. Some corner cases, including $\nu \neq 1$ in (3.8), can arise and are dealt with in our Python implementation of the simulator.

4.3 Simulating solving uSVP

In this section, we review and extend recent work on capturing the probabilistic nature of the described uSVP win condition. In [DDGR20], Dachman-Soled *et al.* revisit the [ADPS16] heuristic methodology described in Section 3.2.2. The authors are concerned with accurately predicting the effects that introducing side channel information to their lattice embedding has on the success probability of solving uSVP using Progressive BKZ, while also maintaining accuracy in the small block size regime, $\beta \leq 45$. The authors describe a *uSVP simulator* (not to be confused with the BKZ simulator of [CN11]), designed to predict the success probability of Progressive BKZ solving an *isotropic* uSVP instance by a specific block size.[4] Using their uSVP simulator, they predict the expected successful block size for a series of experiments they run, and verify the accuracy of their predictions. We start by simplifying the [DDGR20] uSVP simulator for Progressive BKZ, and then develop a similar uSVP simulator for BKZ 2.0. We focus on the simulator as described in [DDGR20] at the time of release. Since the time of writing of this chapter as [PV21], the latest version of the simulator proposed in [DDGR20] adopted some of the techniques described below, for allowing $\tau > 1$ and faster simulations.

[4] Any uSVP instance used in the primal attack can be made isotropic, where $\sigma = 1$.

Input: m, q, d // m q-vectors, dimension d
// δ is LLL's root-Hermite factor ≈ 1.02
// q^m is the lattice's volume
$\log \alpha \leftarrow -2 \log \delta$
// compute the profile's slope
$\texttt{slope} \leftarrow [\log q + \log \alpha, \log q + 2 \log \alpha, \ldots, \varepsilon]$ s.t. $\varepsilon + \log \alpha \leq 0$
if $\#\texttt{slope} \geq d$ **then**
  $\texttt{slope} \leftarrow$ last d entries of $\texttt{slope}$
  shift $\texttt{slope}$ vertically such that $\sum_i \texttt{slope}_i = \log q^m$
  $\texttt{log-profile} \leftarrow \texttt{slope}$
  return $\texttt{log-profile}$
$\ell \leftarrow \#\texttt{slope}$
$v \leftarrow \sum_i \texttt{slope}_i$
// compute the profile's head
$\texttt{head} \leftarrow []$
while $v + \sum_i \texttt{head}_i + \log q < \log q^m$ **and** $\ell + \#\texttt{head} < d$ **do**
  $\texttt{head} \leftarrow \texttt{head} \cup [\log q]$
$\ell \leftarrow \ell + \#\texttt{head}$
$v \leftarrow v + \sum_i \texttt{head}_i$
// compute the profile's tail
$\texttt{tail} \leftarrow []$
while $\ell + \#\texttt{tail} < d$ **do**
  $\texttt{tail} \leftarrow \texttt{tail} \cup [0]$
shift $\texttt{slope}$ vertically such that $\sum_i \texttt{head}_i + \sum_i \texttt{slope}_i = \log q^m$
$\texttt{log-profile} \leftarrow \texttt{head} \cup \texttt{slope} \cup \texttt{tail}$
return $\texttt{log-profile}$

Algorithm 9: LLL Z-shape simulator, assuming a basis as in (3.8) with $\nu = 1$. Returns the logarithm of the basis profile, $\{\log \|\mathbf{b}_i^*\|\}_i$.

Notation. In Algorithms 10, 11, and 12 below, given a (possibly estimated) lattice basis profile "`profile` $= \{\|\mathbf{b}_i^*\|^2\}_i$", we refer to exact or estimated values for $\|\mathbf{b}_i^*\|^2$ as `profile`$[i]$.

4.3.1 Progressive BKZ

The approach proposed in [DDGR20] to estimate the required block size to solve a uSVP instance is to simulate the status of a lattice basis as it is being reduced, and with it the probability at each step of the lattice reduction algorithm that the target vector is recovered.

Input: d
$p_{\text{tot}} \leftarrow 0$, $\bar{\beta} \leftarrow 0$
`profile` $\leftarrow$ GSA profile of an LLL reduced, rank d, isotropic uSVP instance basis
for $\beta \leftarrow 3$ **to** d **do** `/* PBKZ round */`
  `profile` $\leftarrow$ BKZSim(`profile`$, \beta, 1$)
  $p_{\text{lift}} \leftarrow P[\mathbf{v}$ rec. in $\lfloor d/\beta \rfloor$ rounds $\mid \pi_{d-\beta+1}(\mathbf{v})$ rec. this round$]$
  $p_{\text{rec}} \leftarrow P[x \leftarrow \chi^2_\beta : x \leq$ `profile`$[d-\beta+1]]$
  $p_{\text{new}} \leftarrow (1-p_{\text{tot}}) \cdot p_{\text{rec}} \cdot p_{\text{lift}}$
  $\bar{\beta} \leftarrow \bar{\beta} + \beta \cdot p_{\text{new}}$
  $p_{\text{tot}} \leftarrow p_{\text{tot}} + p_{\text{new}}$
  if $p_{tot} \geq 0.999$ **then break**
return $\bar{\beta}$

Algorithm 10: Isotropic uSVP simulator for Progressive BKZ with $\tau = 1$, as proposed in [DDGR20]. We omit the details of computing p_{lift} for simplicity and note that p_{rec} represents $P[\pi_{d-\beta+1}(\mathbf{v})$ recovered this round$]$. Returns the expected block size $\bar{\beta}$ required to solve uSVP.

Let W be the event of solving uSVP during the run of Progressive SVP, W_β the probability of being able to solve uSVP during the round with block size β, and $F_\beta = \neg W_\beta$. Following the notation in Algorithm 7, we assume $\tau = 1$, meaning that for each block size β exactly one tour of BKZ-β is run. Dachman-Soled *et al.* implicitly partition W as follows

$$P[W] = P[W_3] + P[W_4 \wedge F_3] + P[W_5 \wedge F_4 \wedge F_3] + \cdots = \sum_{\beta=3}^{d} P\left[W_\beta \wedge \bigwedge_{j=3}^{\beta-1} F_j\right].$$

Their computation of the expected winning block size $\bar{\beta}$ amounts to implicitly defining a probability mass function for the random variable B representing the

first viable block size to solve the uSVP instance, and computing its expected value. In the case of Progressive BKZ, a block size β being the first viable means that it is the round of BKZ run with block size β (i.e. the tour of Line 3 of Algorithm 7 with block size β) and not any earlier round using a smaller block size, that will solve the uSVP instance. The resulting probability mass function for the distribution of B can be modelled as

$$P[B = \beta] = P\left[W_\beta \wedge \bigwedge_{j=3}^{\beta-1} F_j\right].$$

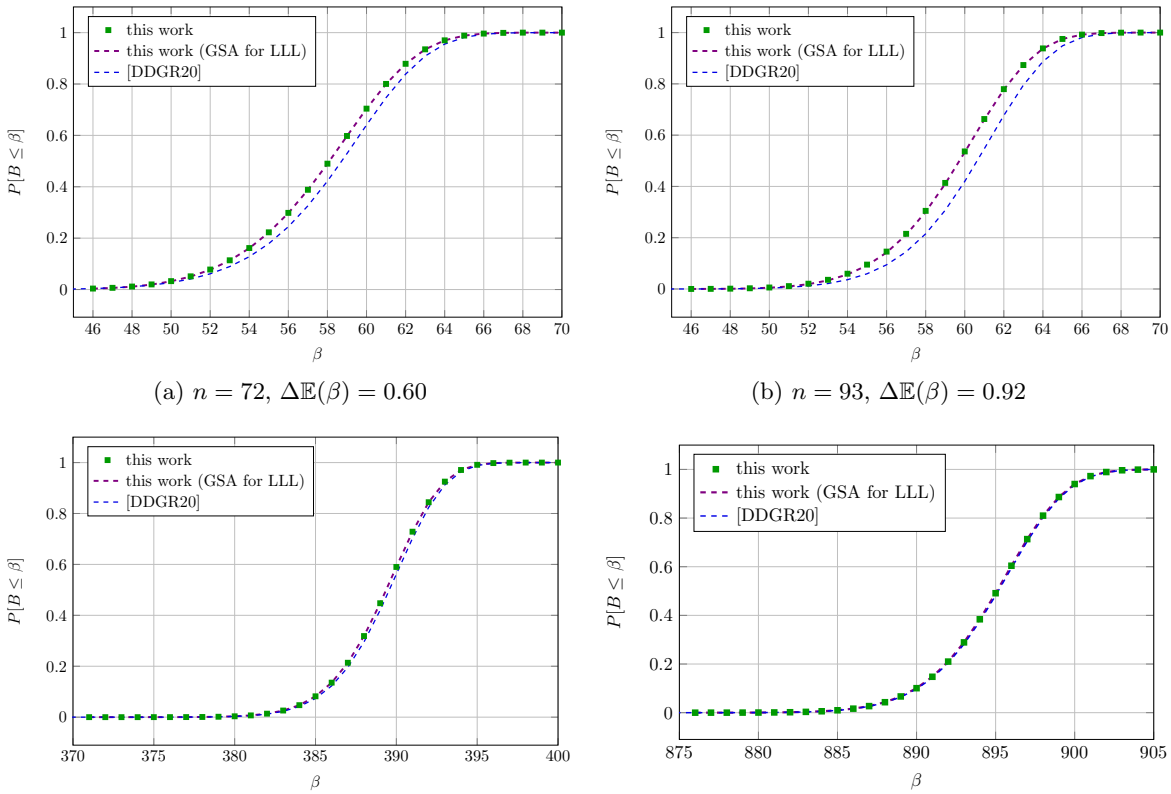
The probability $P[W_\beta]$ is itself modelled as the product of the probability of successfully recovering $\pi_{d-\beta+1}(\mathbf{v})$ by calling O_{SVP} on the last full size block,

$$P[\pi_{d-\beta+1}(\mathbf{v}) \text{ recovered using block size } \beta] \approx P[x \leftarrow \chi^2_\beta \colon x \leq \texttt{profile}[d-\beta+1]],$$

and the probability of successfully lifting the projection over subsequent rounds, p_{lift}. In their implementation of Algorithm 10, Dachman-Soled *et al.* use a chain of conditional probabilities to compute p_{lift}. Events W_i and F_j for $i \neq j$ are considered to be independent since every round of lattice reduction rerandomises the basis, therefore $P[B = \beta]$ is computed as the relevant product

$$P\left[W_\beta \wedge \bigwedge_{j=3}^{\beta-1} F_j\right] = P[W_\beta] \cdot \prod_{j=3}^{\beta-1} P[F_j].$$

We introduce two simplifications to the above uSVP simulator. Firstly, we noticed experimentally that running BKZ with block sizes smaller than 40 will not solve instances for which the [ADPS16] approach predicts a winning block size of $\beta \gtrsim 60$, where most cryptographic applications (and our experiments) reside. Therefore, we skip probability computations for any block sizes smaller than 40. Furthermore, values of p_{lift} approach 1 quickly as β increases, such that one can simply assign $p_{\mathrm{lift}} = 1$ for $\beta \geq 40$; a similar phenomenon is noted in Section 3.3.3. Finally, by allowing multiple tours per block size, we define a uSVP simulator, Algorithm 11, for Progressive BKZ as described in Algorithm 7 where τ may be greater than 1. A comparison between the output of Algorithms 10 and 11 can be found in Figure 4.2 for four isotropic LWE instances, where $\tau = 1$. To produce Figure 4.2, we tweaked the original [DDGR20] code in order to extract the implicit probability mass



(a) $n = 72$, $\Delta\mathbb{E}(\beta) = 0.60$

(b) $n = 93$, $\Delta\mathbb{E}(\beta) = 0.92$

(c) Kyber 512, $\Delta\mathbb{E}(\beta) = 0.20$

(d) Kyber 1024, $\Delta\mathbb{E}(\beta) = 0.09$

Figure 4.2: Comparison between the output of Algorithm 10 [DDGR20] and Algorithm 11 (this work) for isotropic parameters ($\sigma = 1$) from Table 4.1, and on Kyber 512 and 1024 [SAB+19]. The difference in predicted mean first viable block size between the two simulators is reported as $\Delta\mathbb{E}(\beta)$, and is always smaller than 1.

function $P[B = \beta]$. Our simplifications significantly speed up the simulation by avoiding the expensive computation of p_{lift}. In particular, our simulations for Kyber 512 (resp. 1024) take 4 seconds (resp. 31 seconds) against the 20 minutes (resp. 2 hours) of [DDGR20]. We can see that the output probabilities $P[B \leq \beta]$ and the expected successful block sizes differ only slightly and optimistically for the attacker on low dimensional instances when using our simulator, with this difference shrinking for cryptographically sized problems.

Input: (n, q, χ, m), τ
$p_{\text{tot}} \leftarrow 0$, $P \leftarrow \{\}$, $\beta \leftarrow 3$
$d \leftarrow n + m + 1, \sigma^2 \leftarrow \mathbb{V}(\chi)$
`profile` $\leftarrow$ simulated profile of LLL reduced $\text{LWE}_{n,q,\chi,m}$ instance
while $\beta < 40$ **do**
    `profile` $\leftarrow$ BKZSim(`profile`, β, τ)
    $\beta \leftarrow \beta + 1$
while $\beta \leq d$ **do** `/* PBKZ rounds */`
    for *tour* $\leftarrow 1$ **to** τ **do** `/* BKZ tours */`
        `profile` $\leftarrow$ BKZSim(`profile`, $\beta, 1$)
        $p_{\text{new}} \leftarrow P[x \leftarrow \sigma^2 \chi_\beta^2 \colon x \leq \texttt{profile}[d - \beta + 1]]$
        $P[\beta] \leftarrow (1 - p_{\text{tot}}) \cdot p_{\text{new}}$
        $p_{\text{tot}} \leftarrow p_{\text{tot}} + P[\beta]$
        if $p_{tot} \geq 0.999$ **then break**
    $\beta \leftarrow \beta + 1$
return P

Algorithm 11: Unique-SVP success probability simulator for Progressive BKZ, running τ tours for each block size, then increasing the block size by 1. Returns the probability mass function $P[B = \beta]$ of solving uSVP in the round using block size β.

4.3.2 BKZ

Using the same approach as for Algorithm 10 and Algorithm 11, we implemented a uSVP simulator for BKZ, described in Algorithm 12. In this case, the basis profile after a number of tours of BKZ-β is simulated in one shot using the [CN11] simulator. Given that the block size is fixed, the probabilities are only accumulated over tours. It should be noted that the event of β being the first viable block size changes in the case of BKZ. In this case, no unsuccessful tours with a smaller block size are run by the algorithm. Instead, we consider β being first viable if running BKZ-$(\beta - 1)$ for τ tours would not result in a solution to the uSVP instance but running BKZ-β would.

Algorithm 12 returns the probability that τ tours of BKZ-β will solve uSVP, but does not exclude the possibility of winning with a smaller block size. We assume in our model that if τ tours of BKZ-β solve a given uSVP instance, then τ tours of BKZ-β', for $\beta' > \beta$, also will. The values output by Algorithm 12 for a given instance can therefore be interpreted as a cumulative mass function for the first viable block size, i.e. $P[B \leq \beta]$. By running the simulator for increasing

block sizes until it outputs probability 1, one may recover the probability mass function $P[B = \beta]$ as

$$P[B = \beta] = P[B \leq \beta] - P[B \leq \beta - 1].$$

Input: (n, q, χ, m), β, τ
$p_{\text{tot}} \leftarrow 0, \sigma^2 \leftarrow \mathbb{V}(\chi)$
$d \leftarrow n + m + 1$
for *tour* $\leftarrow 1$ **to** τ **do** `/* BKZ tours */`
  `profile` $\leftarrow \text{BKZSim}((n, q, \chi, m), \beta, \text{tour})$
  $p_{\text{new}} \leftarrow P[x \leftarrow \sigma^2 \chi_\beta^2 : x \leq \texttt{profile}[d - \beta + 1]]$
  $p_{\text{tot}} \leftarrow p_{\text{tot}} + (1 - p_{\text{tot}}) \cdot p_{\text{new}}$
return p_{tot}

Algorithm 12: Unique-SVP success probability estimator when running τ tours of BKZ-β. Returns the probability of solving the uSVP instance.

4.4 Experiments

In this section, we describe the experiments we run to check the accuracy of Algorithms 11 and 12, and discuss the results. We start by describing our original batch of experiments in Section 4.4.1. In Section 4.4.2 we make some observations about our experimental results, and describe further tweaked experiments that we run to verify our understanding of the results.

4.4.1 Initial experiments

Our aim in this section is threefold: first, we want to provide experimental evidence for the accuracy of our BKZ and Progressive BKZ uSVP simulators when predicting the success probability of the primal attack against LWE with discrete Gaussian secret and error for different block sizes; second, we want to compare our experiments in Chapter 3 to our uSVP simulations; and finally, we want to explore the effect that using binary or ternary distributions has on the primal attack. Throughout our experiments, we use BKZ 2.0 as implemented in Fpylll [FPY17] version 0.5.1dev, writing our own Progressive BKZ script by using Fpylll's BKZ 2.0 as a subroutine.

For our first goal, we choose three different parametrisations of the LWE problem, for which the [ADPS16] approach predicts an expected successful block size of either 60 or 61. The parameters can be found in Table 4.1. All parameter sets in these batches use discrete Gaussian secret and error with $\mathbb{V}(\chi_s) = \mathbb{V}(\chi_e) = \sigma^2$. The number of LWE samples used, m, is determined by what the LWE estimator [APS15] predicts to be optimal, using (3.2). For each parameter set we generate 100 instances, and reduce them using either BKZ or Progressive BKZ. We then check whether lattice reduction positioned the embedded shortest target vector in the first index of the reduced basis.

In the case of BKZ, for each basis we run a number of tours of BKZ with block size $\beta = 45, \ldots, 65$. The number of tours, τ, takes the values $5, 10, 15, 20, 30$. This results in a total of 100 bases, reduced independently 21×5 times each, once for every combination of β and τ. For every set of 100 reductions, we record the success rate by counting the number of solved instances. We run a similar set of experiments using Progressive BKZ, allowing $\tau \geq 1$ tours per block size, in order to see at what point running extra tours per block size becomes redundant. For this reason, we reduce each basis 5 times, once per value of τ in $1, 5, 10, 15, 20$. After every call to the BKZ subroutine, we check whether the instance is solved. If not, we increase the block size by 1 and run a further round of PBKZ.

The resulting success rates for BKZ and Progressive BKZ (with $\tau = 1$) are plotted in Figure 4.3, together with the output of our uSVP simulators, interpolated as curves. Figure 4.4 contains similar plots for Progressive BKZ with $\tau \geq 1$. In Figure 4.6 we plot the differences $\Delta\mathbb{E}$ and $\Delta\sqrt{\mathbb{V}}$ between the mean and standard deviation for the simulated and experimentally measured probability distributions for the first viable block size, for both Progressive BKZ and BKZ,

$$\Delta\mathbb{E}(\beta) = \text{simulated } \mathbb{E}(\beta) - \text{measured } \mathbb{E}(\beta),$$
$$\Delta\sqrt{\mathbb{V}}(\beta) = \text{simulated } \sqrt{\mathbb{V}}(\beta) - \text{measured } \sqrt{\mathbb{V}}(\beta).$$

For our second goal, we take the success probabilities reported Table 3.1 of Chapter 3. In Figure 4.5 we report the measured success rates at optimal

n	q	σ	m_{2016}	β_{2016}
72	97	1	87	61
93	257	1	105	61
100	257	$\sqrt{2/3}$	104	60

Table 4.1: List of LWE parameters used for testing our uSVP simulators. The instances are in normal form. We use the Bai–Galbraith embedding and the number of samples used, m_{2016}, is given by the LWE estimator (commit `428d6ea`).

and smaller than optimal block sizes, and we superimpose our BKZ success probability simulations for the same lattice parameters.

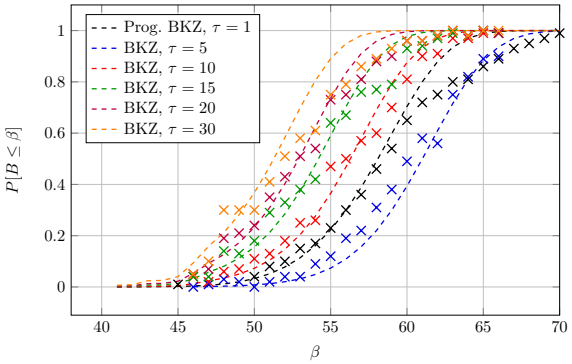
Finally, for our third goal, we run Progressive BKZ experiments for τ in $1, 5, 10, 15, 20$ on three parameter sets using bounded uniform secrets. In particular, we pick the $n = 72$ and $n = 93$ parameters from Table 4.1 but sample secret $\mathbf{s}$ and error $\mathbf{e}$ coefficients uniformly from the set $\{-1, 1\}$, and the $n = 100$ parameters with secret and error coefficients sampled uniformly from $\{-1, 0, 1\}$. This preserves the same standard deviations as in Table 4.1, while adding more structure to the target vector. In the first case, the $\mathbf{s}$ and $\mathbf{e}$ are equivalent to those of a scaled and centred LWE instance with binary secret and error (using a centered and scaled embedding as in Section 3.4.1), while in the second case, the problem is LWE with ternary $\mathbf{s}$ and $\mathbf{e}$. The resulting success probability plots can be found in Figure 4.7.

4.4.2 Observations

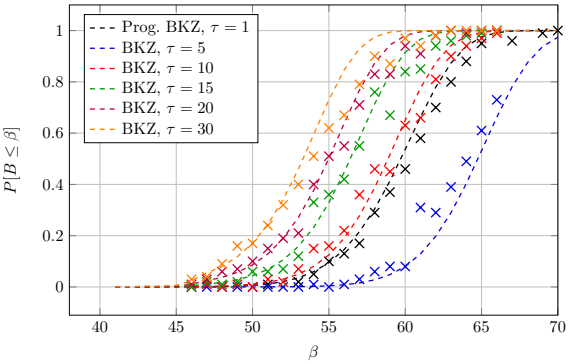
Experimental success rates for both BKZ and Progressive BKZ are in line with the output of the simulators described in Section 4.3. We now look at the results.

4.4.2.1 Progressive BKZ

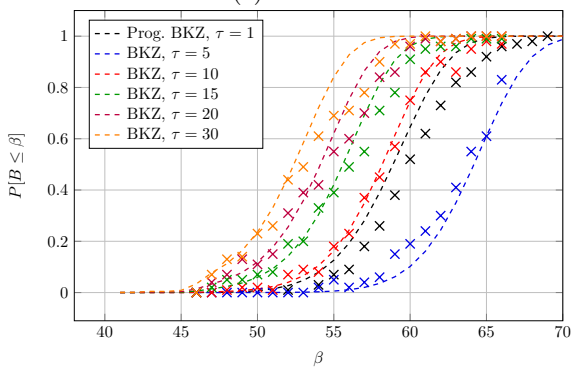
In the case of Progressive BKZ, simulations seem to predict accurately the success probabilities for $\tau \leq 10$ and all secret and error distributions used.



(a) $n = 72$



(b) $n = 93$

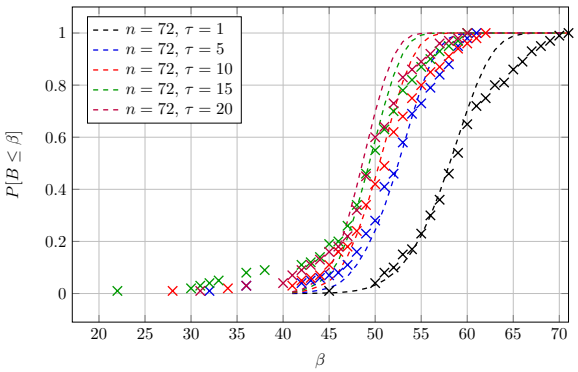


(c) $n = 100$

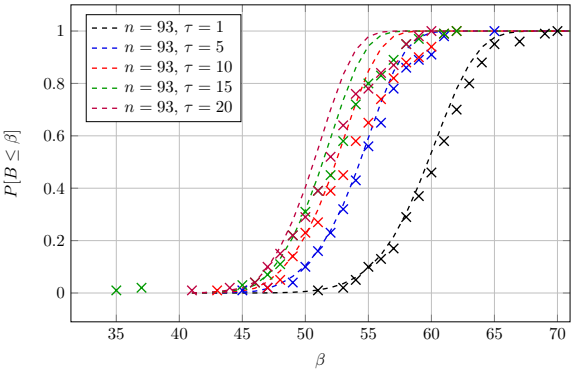
Figure 4.3: Comparison of simulated success probabilities with experimental results for BKZ and Progressive BKZ (with $\tau = 1$). Dashed lines are simulations, crosses are experiments. In the case of Progressive BKZ, 100 total instances are reduced. In the case of BKZ, each experimental result is averaged over 100 instances, with experiments using up to block size 65.

Throughout our experiments reported in Figure 4.4, we observe two ways in which experiments slightly deviate from predictions.

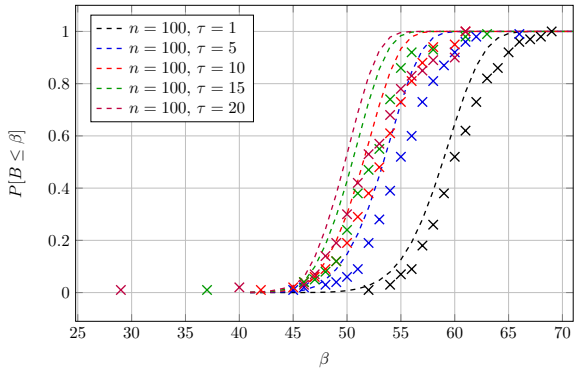
Redundant tours. Firstly, the success probability appears to stop significantly increasing for $\tau > 10$, even when the simulation does predict some improvement. We expect this to be a consequence of the large amount of lattice reduction being performed. Indeed, whenever the BKZ-β subroutine is called, the basis has already been reduced with τ tours of BKZ-$(\beta - j)$ for $j = 1, \ldots, \beta - 3$. This suggests that only little progress on the basis profile can be made with each new tour of BKZ-β. In our experiments, we use Fpylll's BKZ 2.0 implementation with auto-abort, which triggers by default after the slope of the basis profile does not improve for five tours, the slope being computed using a simple linear regression of the logarithm of the basis profile. This means that if it is the case that little progress can be made, fewer than τ tours will be run.



(a) $n = 72$



(b) $n = 93$



(c) $n = 100$

Figure 4.4: Comparison of simulated success probabilities with experimental results for Progressive BKZ with $\tau \geq 1$ on instances with discrete Gaussian secret and error distributions. Dashed lines are simulations, crosses are experiments.

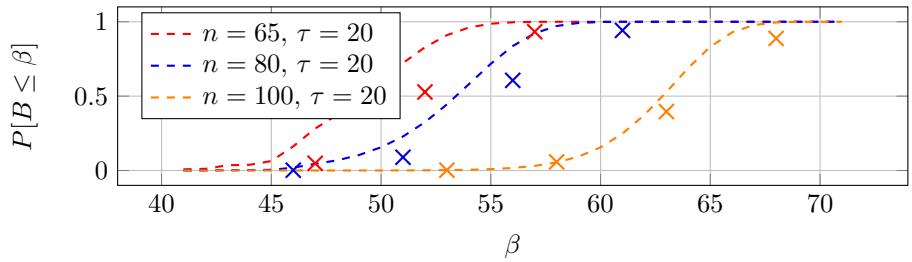
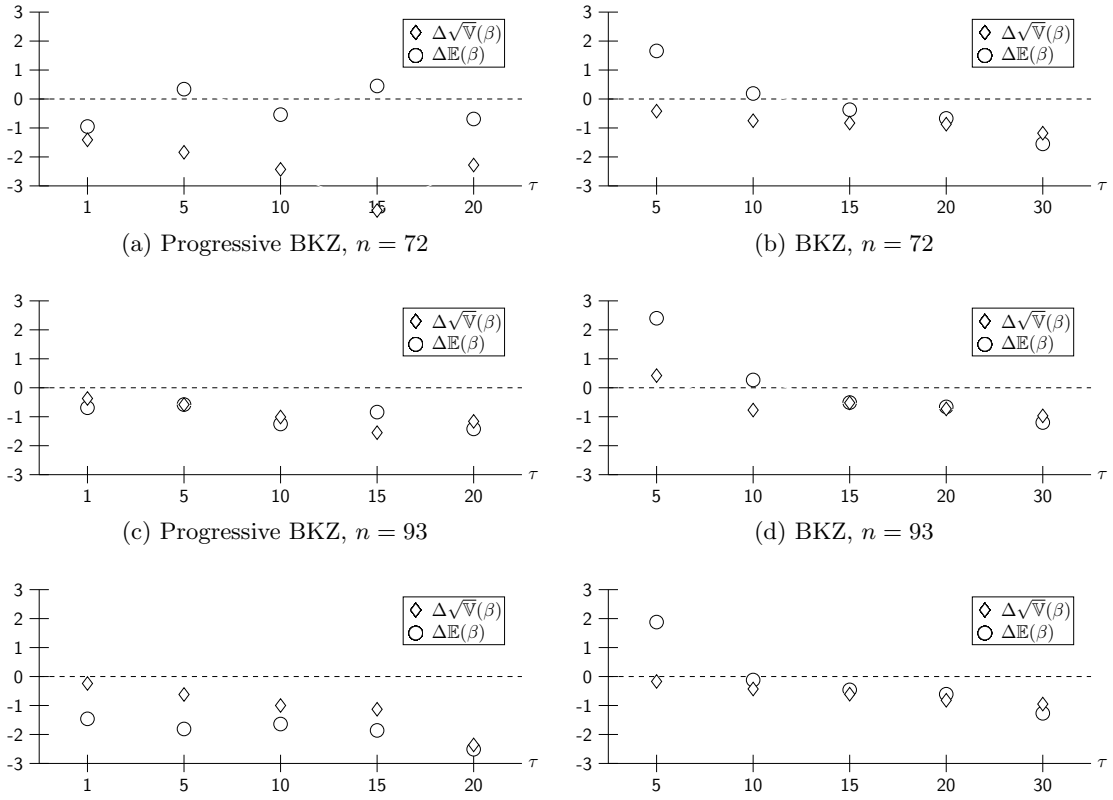


Figure 4.5: Comparison of simulated BKZ success probabilities with experimental results reported in Table 3.1 of Section 3.



(a) Progressive BKZ, $n = 72$

(b) BKZ, $n = 72$

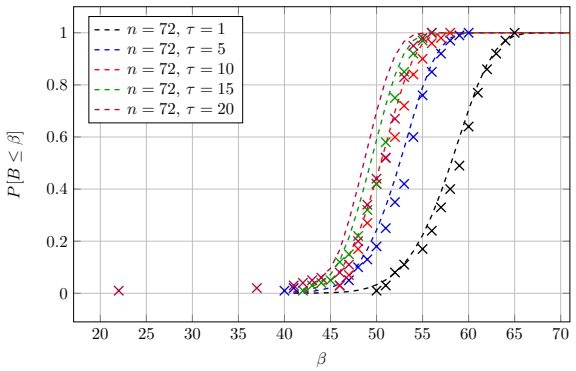
(c) Progressive BKZ, $n = 93$

(d) BKZ, $n = 93$

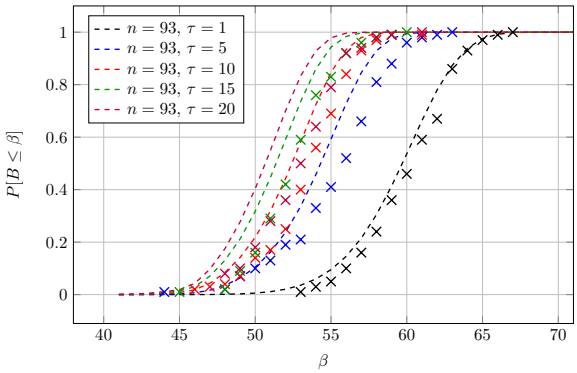
(e) Progressive BKZ, $n = 100$

(f) BKZ, $n = 100$

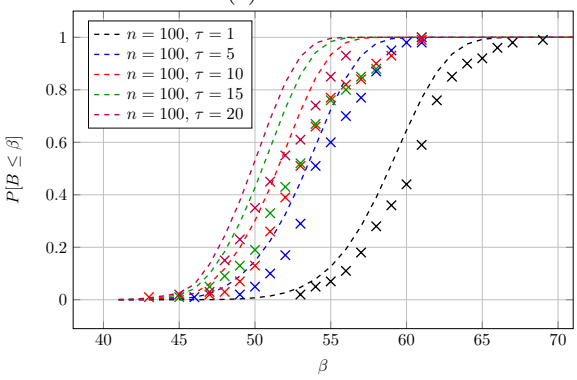
Figure 4.6: The measured difference $\Delta\mathbb{E}(\beta)$ (resp. $\Delta\sqrt{\mathbb{V}}(\beta)$) between the simulated and experimental successful block size mean (resp. standard deviation), as τ grows.



(a) $n = 72$



(b) $n = 93$



(c) $n = 100$

Figure 4.7: Comparison of simulated success probabilities with experimental results for Progressive BKZ on LWE instances with scaled and centred binary secret and error (Figures 4.7a and 4.7b), and ternary secret and error (Figure 4.7c). Dashed lines are simulations, crosses are experiments. Each experimental result is averaged over 100 instances. No changes were made to the uSVP simulators.

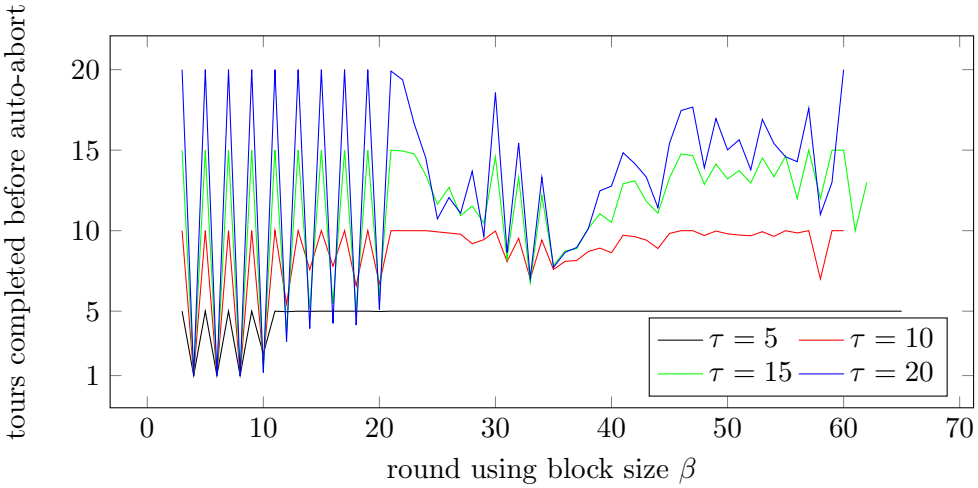


Figure 4.8: Measured number of tours run by the BKZ 2.0 subroutine of Progressive BKZ with $\tau \geq 5$ for each round of reduction with block size β. Numbers are from experiments using the $n = 100$ parameters from Table 4.1, with discrete Gaussian secret and error. Values are averaged over 100 instances. Less than τ tours are run if either BKZ-β does not change the basis or auto-abort triggers.

To verify this, we rerun experiments while measuring the number of tours run by the BKZ subroutine. The data for the $n = 100$ experiments can be found in Figure 4.8, and seems to confirm that auto-abort for $\beta > 20$ is much more frequently triggered for $\tau > 10$. This problem does not affect Progressive BKZ with $\tau = 1$ since even with auto-abort one tour is always run, and only slightly affects $\tau = 5$ and $\tau = 10$.[5] Indeed, predictions match experiments well in the $\tau \leq 10$ cases (cf. Figure 4.4).

Sample variance. The other phenomenon is the presence of a slight plateau in the probability plots as $P[B \leq \beta] \geq 0.8$. In the case of $n = 72$ we also see that smaller than predicted block sizes accumulate a significant success probability. Interestingly, this effect does not appear to be present in the case of binary secret and error LWE, see Figures 4.7a and 4.7b. We expect that this phenomenon is caused by the slight variation in *sample variance* throughout our experiments. Indeed, if we think of our target vector $\mathbf{v} = (v_1, \ldots, v_d)$ as sampled coefficient-

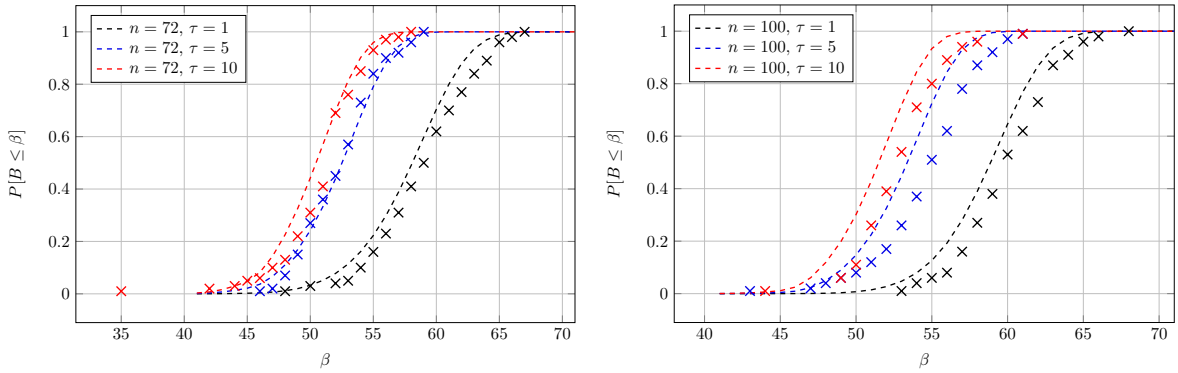
[5] Auto-abort will also not trigger for $\tau = 5$, however in this case sometimes the BKZ-β subroutine with $\beta \leq 10$ returns after only one tour due to not making any changes to the basis.

wise from some distribution χ with variance σ^2, in practice the resulting sample variance for each particular LWE instance $s^2 \coloneqq \frac{1}{d}\sum_{i=1}^{d}(v_i - \bar{v})^2$, with $\bar{v} \coloneqq \frac{1}{d}\sum v_i$ the *sample mean*, will likely slightly deviate from σ^2. We would therefore expect $\|\pi_i(\mathbf{v})\|^2$ to follow a distribution slightly different to $\sigma^2 \cdot \chi^2_{d-i+1}$. However, in the case of $\chi = U(\{-1, 1\})$, the distribution resulting from scaled and centred binary LWE embeddings, this distribution has a very small variance of s^2, i.e. $\mathbb{V}(s^2)$,[6] meaning that most sampled target vectors will have sample variance almost exactly $\mathbb{V}(\chi) = 1$.

To verify this hypothesis, we run a set of $n = 72$ and $n = 100$ discrete Gaussian experiments from Table 4.1, where we resample each LWE instance until the target vector's sample variance is within a 2% error of σ^2, and then run Progressive BKZ with τ in $1, 5, 10$. The resulting experimental probability distributions, shown in Figure 4.9, do not present plateaus (and in the case of $n = 72$, they also do not present the high success probability for small block sizes), supporting our hypothesis. In practice, this effect should not significantly affect cryptographic parameters, as $\mathbb{V}(s^2) \in O(\frac{1}{d})$ [KK51, Eq. 7.20], keeping the effect of fluctuations in $\|\pi_{d-\beta+1}(\mathbf{v})\|^2$ small as the embedding dimension d increases.

Binary and ternary distributions. Our uSVP simulators output similarly accurate simulations for scaled and centred binary and ternary secret and errors, as seen in Figure 4.7, without making any alterations. This is in line with the notion that the hardness of solving uSVP via lattice reduction depends on the standard deviation of the target vector's coefficients rather than on their exact distribution. In recent work [CCLS20], Chen *et al.* run small block size ($\beta \leq 45$) experiments and from their results conclude that the [ADPS16] methodology may be overestimating the security of binary and ternary secret LWE instances, and that discrete Gaussian secrets offer "greater security levels". We believe their conclusions to be incorrect. First, their experiments are exclusively run in the small block size regime, where it is known that lattice heuristics often do not hold [GN08b, §4.2], [CN11, §6.1]. Second, their methodology

[6] Following [KK51], we compute $\mathbb{V}(s^2)$ as approximately 0.00995, 0.00112, and 0.00005 for a discrete Gaussian with $\sigma^2 = 1$, $U(\{-1, 0, 1\})$ and $U(\{-1, 1\})$ respectively, for sets of 200 ($\approx d$) samples.



(a) $n = 72$ (b) $n = 100$

Figure 4.9: Progressive BKZ success probability against LWE instances with discrete Gaussian secret and error and $(n, \sigma^2) \in \{(72, 1), (100, 2/3)\}$, such that their sample variance is within 2% of σ^2.

does not take into account the norm of their embedded shortest vector. In their experiments they compare $\text{LWE}_{n,q,\chi,m}$ instances where χ is swapped between several distributions with different variances. They use the [BG14b] embedding, which results in target vectors whose expected norms grow with the variance of χ. This means instances with narrower χ will be easier to solve, something that can already be predicted by running the LWE estimator using the `secret_distribution` parameter (which will also implicitly account for any advantageous secret vector coefficient guessing, as to reduce the LWE instance's dimensionality). This however does not mean that Gaussian secrets offer inherently "greater security levels" than binary and ternary secrets, but rather that given two LWE instances (where any useful coefficient guessing has already occurred) with the same values for n, q and m, the larger the secret variance, the harder the instance. Gaussian secrets with variance smaller than $1/4$ would result in lower security than binary secrets in such a setting. We think the experiments to determine whether discrete Gaussian secrets are more secure than binary or ternary secrets should therefore compare LWE instances with different secret distributions, but equal variances, as done in this section, and that parameter selection for small secret LWE should take the secret's variance into consideration.

4.4.2.2 BKZ

In the case of BKZ, simulations seem to stay similarly accurate across all secret dimensions n, as reported in Figure 4.3. It should be noted that, even though a larger gap than for Progressive BKZ can be seen between predictions and experiments in the case of $\tau = 5$, this predictive gap in expected block size of less than 3 corresponds to about 1 bit in a core-sieve cost model [ADPS16]. Furthermore, this gap narrows as τ increases. Following experimental results from [Che13, Figure 4.6] and [Alb17], designers often [ACD+18] consider it sufficient to reduce a basis using $\tau = 16$ tours of BKZ when specifying BKZ cost models, due to the basis quality not improving significantly after 16 tours. Our simulators seem accurate for values of τ in such a regime.

Another observation is that Progressive BKZ with $\tau = 1$ outperforms BKZ with $\tau = 5$. Indeed, the earlier performs approximately β tours of increasing block size versus the latter's five tours of block size β. It seems therefore that for these lattice parameters Progressive BKZ applies "more" lattice reduction. We do not attempt to give a closed formula for the minimum block size for which BKZ outperforms Progressive BKZ in output quality, and keep in mind that a direct comparison of first viable block sizes does not alone capture the relative cost of the two algorithms due to Progressive BKZ also performing tours with smaller block sizes.

We also see that the phenomenon of success probabilities not increasing when $\tau \geq 10$ that was observed for Progressive BKZ does not appear to occur in the case of BKZ. This is compatible with our understanding of this phenomenon in the case of Progressive BKZ. Indeed, BKZ-β will not auto-abort as often due to the input basis not having already been reduced with, for example, τ tours of BKZ-$(\beta - 1)$.

However, a different interesting phenomenon can be observed. Sometimes, as the block size is increased, the experimental success probability of BKZ lowers, see the BKZ experiments in Figure 4.3. For example, this happens between block sizes 60 and 61 in Figure 4.3a when running $\tau = 5$ tours of BKZ. Originally we believed this to be caused by the preprocessing strategies used

in Fpylll. Indeed, at the time of writing, preprocessing strategies for block size β (resp. $\beta+1$) could include running BKZ-β' (resp. BKZ-β''), with $\beta' > \beta''$, resulting in inferior quality preprocessing for BKZ-$(\beta+1)$ than for BKZ-β. We replaced the default preprocessing strategies with a custom one such that preprocessing block sizes are non-decreasing as a function of β, however this did not remove the effect.

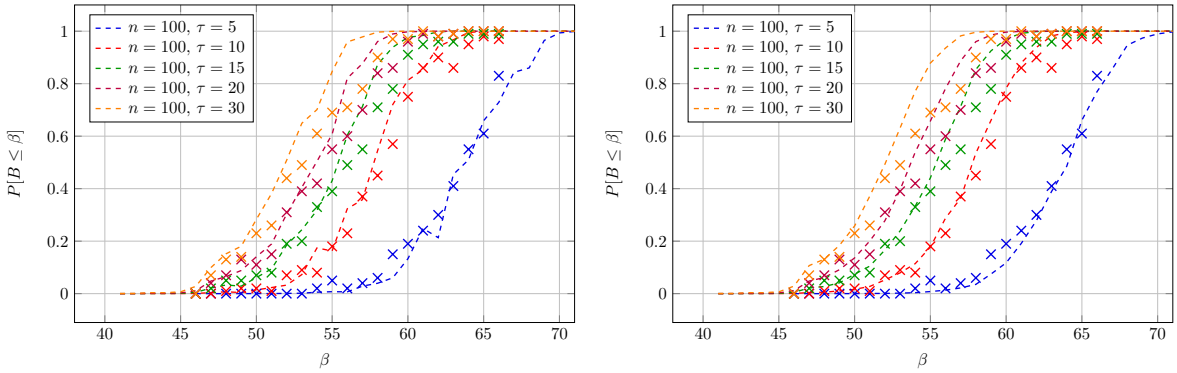
A possible cause for this phenomenon could be that basis profiles output by the [CN11] simulator do not capture the possibility that Gram–Schmidt vector norms can be non decreasing as a function of their index. This means that one could have a BKZ-β reduced basis such that $\|\mathbf{b}^*_{d-\beta}\| < \|\mathbf{b}^*_{d-\beta+1}\|$.[7] This event happening across instances or block sizes could be a potential cause for the phenomenon. The probabilistic BKZ simulator developed in [BSW18] seems to better capture this phenomenon, when run with a fixed PRNG seed. An example of the output of our uSVP simulator for BKZ when replacing the [CN11] simulator with the [BSW18] simulator can be found in Figure 4.10. However, our experimental measurements are averaged over 100 runs. Running our uSVP simulator with the [BSW18] simulator, and averaging its output, results in a simulation with strictly increasing probabilities, unlike our measurements. In any case, the overall success probability predictions stay reasonably accurate.

Finally, looking at Figure 4.5, it seems that our simulations are consistent with the measurements reported in Table 3.1 of Chapter 3. The simulators therefore seem to explain the reported success probabilities of lower than expected block sizes in Chapter 3.

4.5 Cryptographically sized LWE instances

In previous sections we developed simulators for the success probability of solving uSVP instances and tested them against uSVP embedding lattices

[7] In general, by Definition 19 of BKZ-β reducedness we have $\|\mathbf{b}^*_i\|^2 = \theta \cdot \|\pi_i(\mathbf{b}_{i+1})\|^2 \leq \|\pi_i(\mathbf{b}_{i+1})\|^2$ for some $\theta \in (0,1]$. Then $\|\mathbf{b}^*_i\|^2 < \|\mathbf{b}^*_{i+1}\|^2$ *iff* $(1+\mu^2_{i+1,i}) \cdot \theta < 1$ where $\mu_{i+1,i} = \langle \mathbf{b}_{i+1}, \mathbf{b}^*_i \rangle / \|\mathbf{b}^*_i\|^2 \in [-1/2, 1/2]$.



(a) [BSW18]

(b) Averaged [BSW18]

Figure 4.10: Both figures show BKZ experiments and uSVP simulations for $n = 100$ instances with Gaussian secret and error, where the calls to the [CN11] simulator made in Algorithm 12 are replaced. The left plot shows simulations where the [BSW18] simulator is used with a fixed PRNG seed. The right plot shows the same experimental data with with simulations obtained by averaging the output of the [BSW18] simulator over 10 different seeds.

generated from small LWE instances that could be solved in practice. An immediate application could be to use such simulators to estimate the behaviour of lattice reduction when used against cryptographically sized instances.

Here we use the simulator to compute the expected first viable block sizes required to solve LWE and NTRU instances proposed for the NIST PQC standardisation process. In particular we look at the second round versions of the three lattice KEM finalists; Kyber [SAB+19], NTRU [ZCH+19], and Saber [DKRV19]. An interesting option would be to use the simulators to predict what block size is required to solve an instance with a target low success probability. However, as we discuss in Section 4.4.2, the simulations are not necessarily fully accurate for smaller or larger block sizes, due to the fluctuations in sample variance that an instance can have. While the effect should be minor for cryptographically sized instances, low probability attacks may also include combinatorial techniques not captured by our simulators. Therefore, extracting block sizes for low probability attacks from the simulated probabilities may not capture all of the necessary subtleties. Furthermore, we will see that the window of block sizes predicted to be first viable is relatively narrow, so that lower success probability attacks without combinatorial tricks should not be significantly cheaper than higher success probability attacks.

In Table 4.2, we look at parameter sets from the lattice KEM finalists in the third round of the NIST PQC standardisation process [Nat16], as specified during the second round. We provide expected first viable block sizes $\mathbb{E}(\text{succ. } \beta)$ (and their standard deviations $\sqrt{\mathbb{V}}(\text{succ. } \beta)$) when using 15 tours of BKZ, and Progressive BKZ with $\tau = 1$ or 5 (see Algorithm 7). We choose $\tau = 15$ for BKZ due to our experiments seemingly confirming the accuracy of our estimator for this value and its closeness to 16, which is commonly found in BKZ cost models. We choose $\tau = 1$ and $\tau = 5$ in the case of Progressive BKZ since our experiments suggest both cases are accurately predicted by the uSVP simulator; this allows us to see if running more tours in the BKZ subroutine has any effect on the complexity of cryptographically sized parameters.

Two clear disclaimers should be made. First, in Table 4.2 we list the expected block size required to solve uSVP instances for the primal attack. While in an aggressive cost model for these algorithms, such as core-SVP [ADPS16], one could be tempted to make direct cost comparisons between algorithms based only on β, in the case of BKZ we assume that τ tours of BKZ-β are run, while in the case of Progressive BKZ about $\tau\beta$ tours of varying block size are run. Second, for both algorithms we fixed the same number of samples m, chosen with the aid of the LWE estimator as the optimal number of samples when using the "2016 estimate" (except in the case of NTRU, where we assume $m = n$ samples). This is not necessarily the optimal number of samples for each specific block size when computed using a uSVP simulator. We therefore avoid making claims and comparisons regarding the exact cost of solving uSVP using the two algorithms, and propose our results as an intermediate step between using the current LWE estimator and finding a theoretically cheapest attack using our simulators.

4.5.1 Observations

In almost all cases the mean required block size $\mathbb{E}(\text{succ. } \beta)$ is predicted to be larger than the LWE estimator currently suggests. Our results for using Progressive BKZ with $\tau = 1$ against NTRU-HPS are in line with what Dachman-Soled *et al.* [DDGR20, Table 5] predict (NTRU-HPS being the only examined

					BKZ 2.0, $\tau = 15$				Progressive BKZ, $\tau = 1$		Progressive BKZ, $\tau = 5$	
scheme	n	q	σ_s	σ_e	β_{2016}	m	$\mathbb{E}$(succ. β)	$\sqrt{\mathbb{V}}$(succ. β)	$\mathbb{E}$(succ. β)	$\sqrt{\mathbb{V}}$(succ. β)	$\mathbb{E}$(succ. β)	$\sqrt{\mathbb{V}}$(succ. β)
Kyber 512	512	3329	1	1	381	484	386.06	2.56	389.53	2.88	385.70	2.32
Kyber 768	768	3329	1	1	623	681	634.41	2.96	638.23	3.30	634.00	2.66
Kyber 1024	1024	3329	1	1	873	860	891.13	3.31	895.24	3.66	890.63	2.96
LightSaber	512	8192	$\sqrt{5/2}$	$\sqrt{21}/2$	404	507	408.81	2.65	412.24	2.96	408.35	2.39
Saber	768	8192	$\sqrt{2}$	$\sqrt{21}/2$	648	736	659.36	3.00	663.10	3.32	658.85	2.68
FireSaber	1024	8192	$\sqrt{3/2}$	$\sqrt{21}/2$	890	891	907.76	3.34	911.78	3.68	907.16	2.97
ntruhps2048509	508	2048	$\sqrt{2/3}$	$\sqrt{1/2}$	374	508	375.93	2.58	379.56	2.92	375.71	2.36
ntruhps2048677	676	2048	$\sqrt{2/3}$	$\sqrt{\frac{127}{338}}$	521	676	522.78	2.82	526.77	3.18	522.67	2.57
ntruhps4096821	820	4096	$\sqrt{2/3}$	$\sqrt{\frac{51}{82}}$	621	820	628.78	2.83	632.54	3.17	628.43	2.55
ntruhrss701	700	8192	$\sqrt{2/3}$	$\sqrt{2/3}$	471	700	477.20	2.48	480.51	2.77	476.72	2.23

Table 4.2: Security estimates for some lattice schemes. The number of samples m used in the embedding for Kyber and Saber is chosen using the LWE estimator, as to optimise the cost of the attack following the 2016 estimate for BKZ [ADPS16]. In the case of NTRU, the number of samples m is chosen equal to n. β_{2016} is the block size suggested by the LWE estimator. For BKZ and Progressive BKZ, $\mathbb{E}$(succ. β) and $\sqrt{\mathbb{V}}$(succ. β) are the mean and standard deviation of the distribution of first viable block sizes.

scheme in common). The increase in $\mathbb{E}(\text{succ. } \beta)$ may seem counter-intuitive. The [ADPS16] already aims to recover $\mathbb{E}(\text{succ. } \beta)$, with the simulators described in Section 4.3 capturing the success probability of smaller block sizes, possibly reducing the value of $\mathbb{E}(\text{succ. } \beta)$. Indeed, the increase seems to be mainly due to the use of the [CN11] simulator rather than the GSA for predicting the profile of a BKZ reduced basis (i.e. the right hand side of (3.2)). An illustrative example of this happening in the case of Kyber 512 can be see in Figure 4.11. Indeed, patching the LWE estimator to partially[8] use the [CN11] simulator, we obtain $\mathbb{E}(\text{succ. } \beta)$ of Kyber 512 (resp. Kyber 768, Kyber 1024) of 390 (resp. 636, 890), narrowing the gap with the predictions obtained in Table 4.2 by using our uSVP simulators. The small standard deviations reported in Table 4.2 suggest that the success probability of block sizes below $\mathbb{E}(\text{succ. } \beta)$ decrease quickly.

4.6 Conclusions

Overall, our data suggests that the experiments in Section 4.4 show that the techniques in Section 4.3 help to more accurately predict lattice reduction success probabilities for solving uSVP. It also suggests that in the case of short vectors sampled coefficient-wise from bounded uniform distributions, it is the variance of the distribution, and not the exact probability mass function, that determines the hardness of the primal attack against the LWE instance. The uSVP simulators also seem to explain the success probability for smaller than expected block sizes reported in Table 3.1 of Chapter 3.

As part of our experiments, we also tested whether using Progressive BKZ with $\tau > 1$ could be beneficial for an attacker. This seems to be useful to some small degree from the point of view the of success probabilities, although BKZ seems to perform comparatively well. However, Progressive BKZ could be of interest to an attacker that wants to start performing lattice reduction as part of a

[8] For simplicity of implementation, our patch uses the GSA to predict the required block size to perform lattice reduction and the optimal number of samples, as before. It uses the [CN11] simulator for the basis profile output by BKZ, and to predict the block size required to win by running O_{SVP} on the last basis block.

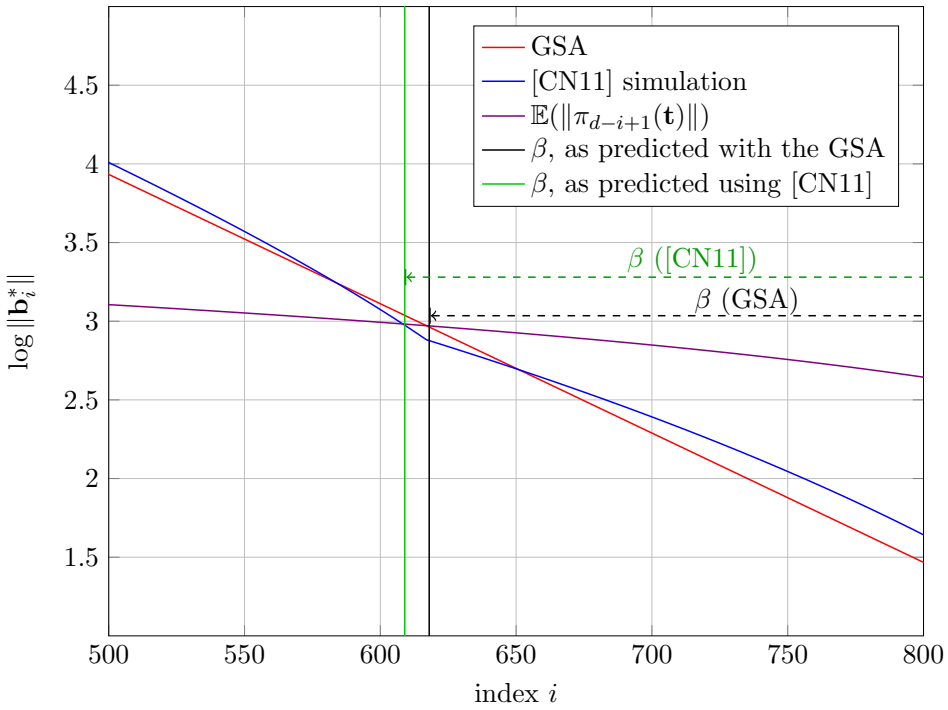


Figure 4.11: Example plot showing the effect on the [ADPS16] methodology of using the [CN11] BKZ simulator rather than the GSA, in the case of Kyber 512. Due to the resulting higher basis profile, the GSA leads to picking a smaller block size. The required winning block size in the [ADPS16] methodology is the distance from the vertical line indicating the intersection to the final basis index d. Note that this plot is zoomed in ($d > 800$).

long term attack, but initially has access to fewer resources[9] than necessary to run BKZ with the expected first viable block size. Progressive BKZ would then allow them to increase their resources as the attack progresses, with $\tau > 1$ allowing them to stop at an overall slightly smaller final block size.

We also note that our preliminary estimates for the success probabilities of lattice reduction on cryptographically sized instances result in higher block sizes than output by the LWE estimator [APS15]. This seems to be mostly due to our use of a BKZ simulator rather than the GSA. A patch to the LWE estimator substituting the GSA with a BKZ simulator could mitigate this effect.

While the simulators presented in [DDGR20] and in this chapter cover BKZ and Progressive BKZ, the techniques are more general and could apply to other lattice reduction algorithms. As an example, it could be interesting to develop a uSVP simulator to assist designing and evaluating new strategies for the General Sieve Kernel (G6K) [ADH+19].

Acknowledgements. We would like to thank Martin Albrecht and Léo Ducas for useful conversations, and for their help simulating the LLL output profile, and again Martin Albrecht for generating new Fplll preprocessing strategies with non-decreasing block sizes.

[9] Say, memory if using lattice sieving to implement O_{SVP}.

Chapter 5

RLWE-based Schemes Using an RSA Coprocessor

Contents

In this chapter we repurpose existing RSA/ECC coprocessors for (ideal) lattice-based cryptography by exploiting the availability of fast long integer multiplication. Such coprocessors are deployed in smart cards, in passports and identity cards, in secured microcontrollers and in hardware security modules (HSM). In particular, we demonstrate an implementation of a variant of the Module-LWE-based Kyber Key Encapsulation Mechanism (KEM) that is tailored for high

performance on a commercially available smart card chip (SLE 78). To benefit from the RSA/ECC coprocessor we use Kronecker substitution in combination with schoolbook and Karatsuba [KO62] polynomial multiplication. Moreover, we speed up symmetric operations in our Kyber variant using the AES coprocessor to implement a PRNG and a SHA-256 coprocessor to realise hash functions. This allows us to execute CCA-secure Kyber768 key generation in 79.6 ms, encapsulation in 102.4 ms and decapsulation in 132.7 ms.

5.1 Motivation

From a practical perspective, two crucial requirements of cryptographic schemes are efficiency and ease of deployment. Indeed, submissions to the NIST process for standardisation of post-quantum cryptography (PQC) are encouraged to provide optimised software implementations aimed at general purpose microprocessors. However, implementations of quantum-safe schemes are also required in constrained (often embedded) environments such as microcontrollers or smart cards.

In the smart card setting, low-power general purpose 16 or 32-bit CPUs are commonly augmented by cryptographic coprocessors capable of executing Diffie-Hellman key exchanges, encryptions or signatures based on RSA or elliptic curves. As such, these cryptographic coprocessors come equipped with an integer multiplier capable of handling multiplication (and addition) in $\mathbb{Z}_N$ for $\log_2 N \approx 2048$.

Contribution. In this chapter, we repurpose existing cryptographic coprocessors to accelerate lattice-based cryptography. For this we make use of variants of Kronecker substitution combined with low-degree polynomial arithmetic. Using this strategy, we manage to implement a variant of the Kyber Key Encapsulation Mechanism (KEM) [SAB+17] using the Kyber768 parameter set promising 161 bits of security, as described in the first round of the NIST

standardisation process.[1] Our various implementations target a commercially available smart card (SLE 78 with 16 Kbyte RAM) and its RSA, AES, and SHA-256 coprocessors. To evaluate Kronecker substitution we implement standard Kronecker substitution (KS1) together with Karatsuba-based polynomial multiplication, and Kronecker substitution with negated evaluation points (KS2) [Har09] using schoolbook-based polynomial multiplication. We compare our results with an implementation of Kyber and NewHope on the same target device that are not utilising large integer multiplication on the coprocessor, implementations of RSA as well as related work. In summary, our work provides evidence that lattice-based post-quantum cryptography can be competitive with RSA on contactless high-security 16-bit smart cards with only limited RAM when RSA, AES and SHA-2 coprocessors are used.

Approach and chapter roadmap. The key computational task in {Ring, Module}-LWE encryption/decryption is to evaluate

$$\textsc{MulAdd}\big(a(x),\, b(x),\, c(x),\, f(x)\big) \coloneqq a(x) \cdot b(x) + c(x) \bmod f(x)$$

for polynomials $a(x), b(x), c(x) \in \mathbb{Z}_q[x]/(f(x))$. In this work, we realise the MulAdd gadget using a combination of a variant of Kronecker substitution and low-degree polynomial arithmetic in the spirit of Schönhage's trick [Sch77]. Kronecker substitution is a well-known and well-utilised technique in computer algebra to reduce polynomial multiplication to integer multiplication. Briefly, we start from standard Kronecker substitution [VZGG13, p. 245] by considering $a(2^\ell) \cdot b(2^\ell) + c(2^\ell) \bmod f(2^\ell)$ where e.g. $a(2^\ell)$ represents the integer obtained by evaluating $a(x)$ at 2^ℓ for some sufficiently big integer ℓ. However, for typical parameter choices, e.g. those of Kyber or NewHope [ADPS16], this strategy produces integers too large for our hardware multiplier to handle. Thus, in Section 5.3 we apply a variant of Harvey [Har09] to our use-case. Harvey proposed Kronecker variants which permit to half the required bit-size of the integers being multiplied at the cost of doubling the number of multiplications. This provides a worthwhile trade-off for medium-sized integers where quasi-linear

[1] We stress that our variant of Kyber is not interoperable with Kyber as specified in [SAB+17]. The main differences are choices for symmetric functions and that Kyber explicitly requires the usage of the Number Theoretic Transform (NTT), which we cannot realise efficiently with our approach.

integer multiplication algorithms [SS71, HVDH19] are not yet competitive. However, in our context Harvey's technique on its own still does not suffice to reduce the integer operands to match our hardware multiplier. Thus, in Section 5.4 we describe how we utilise (low-degree) polynomial arithmetic on top. Overall, we obtain an implementation which computes the IND-CCA Kyber768 decapsulation in $8 \cdot (3^2 + 3 + 3) = 120$ modular multiplications of 2049-bit numbers. In contrast, decrypting 2048-bit RSA requires roughly $2 \cdot 1.5 \cdot 1024 = 3072$ multiplications of 1024-bit numbers in Chinese Remainder Theorem (CRT) representation.[2] We describe our implementation in detail in Section 5.5, discuss performance in Section 5.6 and finish with a discussion in Section 5.7.

A proof of concept implementation of our MulAdd technique can be found at `github.com/fvirdia/lwe-on-rsa-copro`.

Large modulus LWE. In lattice-based cryptography, noisy variants of Kronecker substitution have been used to show various polynomial-time equivalences. In [BLP+13] a reduction from n-dimensional LWE with modulus q to 1-dimensional LWE with modulus q^n is provided using

$$A \coloneqq \sum_{i=0}^{n-1} a_i q^i, \quad S \coloneqq \sum_{i=0}^{n-1} s_i q^{n-i-1}, \quad A \cdot S \bmod q^n \approx \langle \mathbf{a}, \mathbf{s} \rangle \cdot q^{n-1}. \tag{5.1}$$

This reduction is extended to the Approximate-GCD problem in [CS15]. In [CLT13], a variant of the Approximate-GCD problem is defined for realising fully homomorphic encryption which permits to pack several plaintext bits into one big integer using the CRT. The reduction from [BLP+13] is extended in [AD17] to a reduction from Module-LWE to large modulus Ring-LWE and a dimension-halving, modulus squaring self-reduction of Ring-LWE. In [Gu19], it is noted that given $A \coloneqq \sum_{i=0}^{n-1} a_i q^i$, $S \coloneqq \sum_{i=0}^{n-1} s_i q^i$ and

[2] Of course, this metric does not account for the cost of embedding of polynomials into integers as well as additional operations required in lattice-based cryptography, like randomness sampling or expensive CCA transformations. Moreover, the data structures in RSA are much smaller than in lattice-based cryptography so that transfers between CPU and coprocessors with internal memory appropriate to hold RSA-2048 base, exponent, modulus and result have much less impact on performance.

$c(x) = a(x) \cdot s(x) \bmod x^n + 1$, we have

$$A \cdot S \bmod (q^n + 1) \approx_s \sum_i^{n-1} c_i q^i,$$

where $\approx_s$ means $\approx$ in each "slot" defined by powers of q. This observation then gives rise to the I-RLWE problem, which also permits packing several plaintext bits into one large integer. In [Gu19], a reduction from Ring-LWE to I-RLWE is given, but this reduction does not consider the noise distribution, only its size.[3] In [Ham17], a variant of I-RLWE over a pseudo-Mersenne field is given to instantiate an MLWE KEM. Similarly, [AJPS17] can be considered as an integer variant of NTRU.

Post-quantum cryptography on microcontrollers. Microcontrollers and embedded processors usually have only very limited amount of available RAM and space to store program code, and operate with relatively simple 8-, 16-, or 32-bit processor architectures. They are sometimes also referred to as *constrained devices* and are mostly used in embedded applications where low energy consumption, reduced device costs, and other aspects like real-time capabilities are required. Such requirements are commonly not fulfilled by computer systems or powerful System-on-Chips (SoC) with external non-volatile memory (NVM) or RAM. A special class of constrained devices are smart cards or chip cards which are used in electronic banking, secured identification (e.g. passports or national ID cards), authentication, or transport and ticketing applications. Smart cards are usually equipped with protection mechanisms against a wide range of invasive or non-invasive attacks and they often feature dedicated accelerators to speed up and to protect cryptographic operations (e.g. AES, ECC or RSA). Most commercial chip cards are certified according to Common Criteria[4] and evaluated in a laboratory.

The implementation of post-quantum cryptography on constrained devices is an active research area. Most works in the literature focus on performance but from a practical standpoint RAM consumption, code footprint and maintainability

[3] We note, though, that according to all known cryptanalytic results for public-key *encryption* based on LWE, the noise distribution does not play a significant role if it provides enough entropy.

[4] See `http://www.commoncriteriaportal.org/products/#IC`.

of the code-base are also important metrics. Examples of PQC implementations are works that deal with multivariate signatures [CHT12], code-based encryption [vMOG15] and hash-based signatures [HRS16]. In the area of lattice-based cryptography, examples are an implementation of NTRU [BSJ15], an implementation of BLISS signatures on 32-bit ARM [OPG14], an implementation of CPA-secure public-key encryption based on Ring-LWE on an 8-bit AVR [LPO+17] and 32-bit ARM [dCRVV15]. An implementation of the NewHope key exchange protocol which is similar to Ring-LWE encryption is given in [AJS16]. In [KBMSRV18] an implementation of Saber is provided which is an MLWE-based KEM that does not rely on the NTT for polynomial multiplication.

Similarly, the protection of lattice-based cryptography against side-channel attacks has already been explored. An implementation of a masked decryption of Ring-LWE CPA-secure PKE is described in [RdCR+16] and an implementation of a CCA-secure and masked variant is given in [OSPG18]. A masked implementation of the GLP signature scheme is provided in [BBE+18]. What had received comparably less attention in the literature up to the time of publication of this chapter as [AHH+18] were flexible cryptographic coprocessors for lattice-based cryptography in the spirit of RSA or ECC coprocessors (cf. [SBPV07]) and instruction set extensions (cf. a multiply-accumulate instruction [Wen13]).

5.2 Preliminaries

We use the notation for polynomials set in Section 1.1. In particular, we let $R = \mathbb{Z}[x]/(x^n + 1)$ where n is a power of two, and let $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ for some positive integer q.

5.2.1 Kyber

A recent construction relying on the MLWE problem is the Kyber Key Encapsulation Mechanism. Kyber has been submitted to the NIST PQC standardisation process [SAB+17] and a variant is also published as an academic

paper [BDK+18]. It is defined by an intermediate IND-CPA secure Public-Key Encryption (PKE) scheme which is then transformed to an IND-CCA secure KEM using a generic transform [HHK17].[5] We note that Kyber unambiguously refers to the IND-CCA secure KEM, i.e. [SAB+17] does not formally propose a public-key encryption scheme nor a KEM which only claims IND-CPA security. Furthermore, we stress that we will be referring throughout to Kyber as it was presented during the first round of the NIST standardisation process, and will not discuss the updates made to the its parameters in the following rounds.

Definition 27 (Simplified Kyber.CPA following [BDK+18]; cf. [SAB+17])**.** *For $n = 256$ let $k, n, q, \eta, d_t, d_u, d_v$ be positive integers. Let $\mathcal{M} = \{0,1\}^n$ be the plaintext space, where each message $m \in \mathcal{M}$ can be seen as a polynomial in R with coefficients in $\{0,1\}$. Define the functions*

$$\begin{aligned} \textsc{Compress}_q(x,d) &:= \lfloor (2^d/q) \cdot x \rceil \bmod^{(+)} 2^d, \\ \textsc{Decompress}_q(x,d) &:= \lfloor (q/2^d) \cdot x \rceil, \end{aligned}$$

let χ be a centered binomial distribution with support $\{-\eta, \ldots, \eta\}$, and let χ_n be the distribution of polynomials of degree n with entries independently sampled from χ. Define the public-key encryption scheme Kyber.CPA $=$ *(*Kyber.CPA.Gen*,* Kyber.CPA.Enc*,* Kyber.CPA.Dec*) as in Algorithms 13, 14 and 15.*

$(\rho, \sigma) \xleftarrow{\$} \{0,1\}^{256} \times \{0,1\}^{256}$
$\mathbf{A} \xleftarrow{\rho} R_q^{k \times k}$
$(\mathbf{s}, \mathbf{e}) \xleftarrow{\sigma} \chi_n^k \times \chi_n^k$
$\mathbf{t} \leftarrow \textsc{Compress}_q(\mathbf{As} + \mathbf{e}, d_t)$
return $pk_{CPA} := (\mathbf{t}, \rho)$, $sk_{CPA} := \mathbf{s}$

Algorithm 13: Kyber.CPA.Gen.

In Kyber, the parameters that define the base ring R_q are fixed at $n = 256$ and $q = 7681$. The parameters that define key and ciphertext compression are also fixed and set to $d_u = 11, d_v = 3$ and $d_t = 11$. The three different security levels are obtained by different choices of k and η. All relevant Kyber parameters are summarised in Table 5.1.

[5] We note that [SAB+17] does not include the Targhi-Unruh tag.

Input: $pk_{\text{CPA}} = (\mathbf{t}, \rho)$
Input: $m \in \mathcal{M}$
Input: $r \xleftarrow{\$} \{0,1\}^{256}$
1 $\mathbf{t} \leftarrow \text{Decompress}_q(\mathbf{t}, d_t)$
2 $\mathbf{A} \xleftarrow{\rho} R_q^{k \times k}$
3 $(\mathbf{r}, \mathbf{e}_1, e_2) \xleftarrow{r} \chi_n^k \times \chi_n^k \times \chi_n$
4 $\mathbf{u} \leftarrow \text{Compress}_q(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1, d_u)$
5 $v \leftarrow \text{Compress}_q(\langle \mathbf{t}, \mathbf{r} \rangle + e_2 + \lfloor \frac{q}{2} \rceil \cdot m, d_v)$
6 **return** $c \coloneqq (\mathbf{u}, v)$

Algorithm 14: Kyber.CPA.Enc.

Input: $sk_{\text{CPA}} = \mathbf{s}$
Input: $c = (\mathbf{u}, v)$
1 $\mathbf{u} \leftarrow \text{Decompress}_q(\mathbf{u}, d_u)$
2 $v \leftarrow \text{Decompress}_q(v, d_v)$
3 **return** $\text{Compress}_q(v - \langle \mathbf{s}, \mathbf{u} \rangle, 1)$

Algorithm 15: Kyber.CPA.Dec.

The performance of an implementation of Kyber depends highly on the speed of the polynomial multiplication algorithm and the performance of the PRNG instantiations as a large number of pseudo-random data is required when generating $\mathbf{A} \xleftarrow{\rho} R_q^{k \times k}$ or when sampling noise from χ_n^k. Regarding operations in R_q, Kyber.CPA.Gen requires the computation of k^2 multiplications and $(k-1)k + k$ additions (line 4 of Algorithm 13). For encryption as defined in Kyber.CPA.Enc, k^2 multiplications and $(k-1)k + k$ additions (line 4 of Algorithm 14) as well as k multiplications and $(k-1) + 2$ additions (line 5) are needed. The decryption routine Kyber.CPA.Dec can be implemented with k multiplications and $k-1+1$ additions (line 3 of Algorithm 15). Note that Kyber specifies a Number Theoretic Transform (NTT). The NTT allows to implement a fast polynomial multiplication by computing $c = \text{NTT}^{-1}(\text{NTT}(a) \circ \text{NTT}(b))$ for $a, b, c \in R_q$, where $\circ$ denotes coefficient-wise multiplication. Kyber exploits that the NTT is a one-to-one map and assumes that randomly sampled polynomials in $\mathbf{A}$ are already in the transformed domain. Thus, an implementation using a different multiplication algorithm than the NTT would have to apply an inverse transformation first and then use the polynomial multiplication algorithm of its choice to stay compatible with the original specification.

Given two hash functions $G\colon \{0,1\}^* \to \{0,1\}^{2\times 256}$ and $H\colon \{0,1\}^* \to \{0,1\}^{256}$, Kyber is obtained from Kyber.CPA using a Fujisaki-Okamoto style transform

Scheme	n	q	k	η	Bit-sec.	NIST lv.	failure	$\|pk\|$	$\|sk\|$	$\|ctxt\|$
Kyber512	256	7681	2	5	102	1	2^{-145}	736	1632	800
Kyber768	256	7681	3	4	161	3	2^{-142}	1088	2400	1152
Kyber1024	256	7681	4	3	218	5	2^{-169}	1440	3168	1504

Table 5.1: Parameters proposed to NIST for instantiating Kyber KEM. Sizes of the public key ($|pk|$), secret key ($|sk|$), and ciphertext ($|ctxt|$) are given in bytes. "Bit-sec." refers to the "quantum" bit security claimed by the designers.

from [HHK17] as shown in Algorithms 16, 17, 18. Within KYBER.DECAPS a re-encryption has to be computed whose result is compared to the received ciphertext. Thus Kyber specifies exactly how to generate the uniformly random matrix $\mathbf{A}$ as well as polynomials from the error distribution χ_n from a seed. For this the authors of Kyber have chosen different instantiations from the SHA3 family (SHAKE-128, SHAKE-256, SHA3-256 and SHA3-512).

$((\mathbf{t}, \rho), \mathbf{s}) \leftarrow$ KYBER.CPA.GEN()
$z \xleftarrow{\$} \{0,1\}^{256}$
$h \leftarrow H(\mathbf{t}, \rho)$
return $pk := (\mathbf{t}, \rho)$, $sk := (\mathbf{s}, \mathbf{t}, \rho, h, z)$

Algorithm 16: KYBER.GEN.

Input: $pk = (\mathbf{t}, \rho)$
$m \xleftarrow{\$} \{0,1\}^{256}$
$m \leftarrow H(m)$
$(\hat{K}, r) \leftarrow G(m, H(pk))$
$(\mathbf{u}, v) \leftarrow$ KYBER.CPA.ENC$(pk, m; r)$
$c \leftarrow (\mathbf{u}, v)$
$K \leftarrow H(\hat{K}, H(c))$
return (c, K)

Algorithm 17: KYBER.ENCAPS.

5.2.2 Target platform

We use an Infineon SLE 78CLUFX5000 chip card[6] with 16 Kbyte RAM and 500 Kbyte NVM which features a 16-bit CPU running at 50 MHz. The target chip is equipped with common peripherals (watchdog, timers), internal secu-

[6] We refer the reader to `https://www.infineon.com/cms/de/product/security-smart-card-solutions/security-controllers/sle-78/` for more information on the SLE 78 family.

Input: $sk = (\mathbf{s}, \mathbf{t}, \rho, h, z)$
Input: $c = (\mathbf{u}, v)$
$m' \leftarrow$ KYBER.CPA.DEC$(\mathbf{s}, (\mathbf{u}, v))$
$(\hat{K}', r') \leftarrow G(m', h)$
$(\mathbf{u}', v') \leftarrow$ KYBER.CPA.ENC$(pk, m'; r')$
if $(\mathbf{u}', v') = (\mathbf{u}, v)$ **then**
  $K \leftarrow H(\hat{K}', H(c))$
else
  $K \leftarrow H(z, H(c))$
return K

Algorithm 18: KYBER.DECAPS.

rity functions and encryption procedures, a True Random Number Generator (TRNG), as well as a symmetric coprocessor to accelerate AES, a coprocessor to compute SHA-256 and an asymmetric coprocessor for RSA and ECC acceleration. The chip allows contact-based as well as contactless operation where it is powered by a field generated by a common smart card reader. It is intended for use in applications like passports, identity cards, access control or payment cards (e.g. banking, value or credit cards). A similar target device from the SLE 78 family has previously been used to implement hash-based XMSS signatures [HBB13] and eta pairings [GK15].

The asymmetric coprocessor on the SLE 78CLUFX5000 allows fast basic long number calculations on integers slightly larger than 2048 bits (addition, subtraction, integer multiplication, modular multiplication). In practice it is mainly used by cryptographic libraries for RSA and ECC. However, for an earlier generation smart card (Infineon SLE 66P) Garcia and Seifert describe an implementation of AES on the modular arithmetic coprocessor [GS02].

As there is no standard for RSA/ECC coprocessors, our low-level implementation is certainly vendor specific. However, the general approach described in Section 5.3 and Section 5.4 should be transferable to a large number of devices as most other smart card vendors appear to use similar approaches. Additional devices that could profit from our work could be server systems like the IBM PCIe Cryptographic Coprocessor[7] or existing FPGA-based RSA/ECC accelerator cards or RSA/ECC accelerator intellectual property (IP) cores.

[7] See `http://www-304.ibm.com/jct01003c/common/ssi/ShowDoc.wss?docURL=/common/ssi/rep_sm/1/649/ENUS4767-_h01/index.html`.

5.3 Kronecker substitution

Kronecker substitution (KS) is a classical technique in computer algebra for reducing polynomial arithmetic to large integer arithmetic, cf. [VZGG13, p. 245] and [Har09]. The fundamental idea behind this technique is that univariate polynomial and integer arithmetic are identical except for carry propagation in the latter. Thus, coefficients are simply packed into an integer in such a way as to terminate any possible carry chain. For example, say, we want to multiply two polynomials $f(x) \coloneqq x + 2$ with $g(x) \coloneqq 3x + 4$ in $\mathbb{Z}[x]$. We may write $f(100) = 100 + 2 = 102$ and $g(100) = 300 + 4 = 304$. Multiplying gives $102 \cdot 304 = 31008$ or $3x^2 + 10x + 8$. In implementations, we use powers of two as evaluation points since this permits efficient "packing" (polynomial to integer) and "unpacking" (integer to polynomial) using only cheap bit shifts.

In this work, we employ Kronecker substitution for computing

$$\textsc{MulAdd}\big(a(x),\, b(x),\, c(x),\, f(x)\big) \coloneqq a(x) \cdot b(x) + c(x) \bmod f(x)$$

with all polynomials having signed coefficients from different ranges.

In more detail, we first pack the polynomials into integers A, B, C, F using Algorithm 19 (Snort). We then compute $D \coloneqq A \cdot B + C \bmod F$. Finally, we unpack D to $d(x)$ using Algorithm 20 (Sneeze). We note that our packing/unpacking algorithms are straight-forward adaptations of standard Kronecker packing/unpacking to the signed case, cf. [Har09]. We made public high-level, proof-of-concept SageMath [S+17] implementations for the algorithms in this section at `https://github.com/fvirdia/lwe-on-rsa-copro`.

Lemma 20 establishes the correctness of this procedure. While correctness of Kronecker substitution is well-established [Har09], we give a complete proof of correctness and in particular the required precision in order to maintain the same error as in Kyber, since faithful re-encryption is required for standard IND-CCA transforms such as the one in [HHK17] utilised by Kyber. On the other hand, loosening this requirement, permits to decrease precision (the parameter ℓ below) and hence to improve performance. Before stating and

Input: $g \in \mathbb{Z}[x]$
Input: $f \in \mathbb{Z}[x]$
Input: bitlength ℓ
1 **return** $g(2^\ell) \bmod^{(+)} f(2^\ell)$
Algorithm 19: SNORT(g, f, ℓ).

proving Lemma 20, we describe in Example 18 the approach to signed coefficient unpacking that we use, which underpins the proof of the lemma.

Example 18. *Let $f = f_1\, x + f_0$ and $F = f(B)$ be the result of a polynomial multiplication using KS with evaluation point $B = 100$. In the case of unsigned coefficients, we would know that $|f_i| < B$ and would recover f_0 as $F \bmod^{(+)} B$. However, in the case of signed coefficients, this is not sufficient. Indeed, let $f = x - 3$ such that $F = 97$. Then $\bar{f}_0 = F \bmod^{(+)} 100 = 97 \neq f_0$. We can however unpack f if we change our guarantee on f's coefficients to $|f_i| < B/2$. Indeed, in this case*

$$\bar{f}_0 = F \bmod^{(+)} B = f_1 \cdot B + f_0 \bmod^{(+)} B = f_0 \bmod^{(+)} B = f_0 + \mathtt{t} \cdot B,$$

for some $\mathtt{t} \in \mathbb{Z}$. Since we know that $|f_0| < B/2$, $\mathtt{t}$ must be either 0 (if $\bar{f}_0 < B/2$) or 1 (if $\bar{f}_0 \geq B/2$). In the first case $f_0 = \bar{f}_0$, otherwise $f_0 = \bar{f}_0 - B$. We can now subtract f_0 from F, divide by B and move on to recover f_1 by repeating the same process.

Remark 19. *The procedure described in Example 18 essentially amounts to replacing $\bmod^{(+)}$ with $\bmod^{(-)}$ during KS unpacking, and subtracting f_0 from F and dividing by B when moving to the next coefficient rather than just floor-dividing F by B.*

Lemma 20. *Let $a, b, c \in \mathbb{Z}[x]$ such that $a = \sum_{i=0}^{n-1} a_i x^i$, $b = \sum_{i=0}^{n-1} b_i x^i$, $c = \sum_{i=0}^{n-1} c_i x^i$ with $|a_i| \leq \alpha$, $|b_i| \leq \beta$, and $|c_i| \leq \gamma$. Let*

$$d \coloneqq \sum_{i=0}^{n-1} d_i\, x^i \equiv a \cdot b + c \bmod f$$

with $|d_i| \leq \delta$, where δ is positive and depends on $\alpha, \beta, \gamma, n, f$ and f is monic of degree n such that $f(2^\ell) > 2^{n\ell} - 1$. Let $\varphi \coloneqq \max_{i<n} |f_i|$, and let $\ell > \log_2(\delta + \varphi) + 1$ be an integer (e.g. $\ell = \lceil \log_2(\delta + \varphi + 1) \rceil + 1$).

Input: $G \in \{0, \ldots, f(2^\ell) - 1\}$
Input: $f \in \mathbb{Z}[x]$, monic
Input: bitlength ℓ
$n \leftarrow \deg(f)$
$G^{[-1]} \leftarrow G$
for $i \leftarrow 0$ **to** $n-1$ **do** `// step` i
  $e^{(i)} \leftarrow G^{[i-1]} \bmod^{(+)} 2^\ell$
  $G^{[i]} \leftarrow \left(G^{[i-1]} - e^{(i)}\right)/2^\ell$
  if $e^{(i)} > 2^{\ell-1}$ **then** `// negative coefficient`
    $e^{(i)} \leftarrow e^{(i)} - 2^\ell$
    $G^{[i]} \leftarrow G^{[i]} + 1$
  $r^{(i)} \leftarrow e^{(i)}$
for $i \leftarrow 0$ **to** $n-1$ **do** $r^{(i)} \leftarrow r^{(i)} - f_i\, G^{[n-1]}$ `// subtract` $\mathtt{b} \cdot f(x)$
return $\{r^{(i)}\}_{i=0}^{n-1}$

Algorithm 20: SNEEZE(G, f, ℓ). The label $G^{[i]}$ represents the state of the $n\ell$-bit integer variable G at step i.

If

$$A \coloneqq \textsc{Snort}(a,\, f,\, \ell),$$
$$B \coloneqq \textsc{Snort}(b,\, f,\, \ell),$$
$$C \coloneqq \textsc{Snort}(c,\, f,\, \ell),$$

and

$$D \coloneqq A \cdot B + C \bmod^{(+)} f(2^\ell),$$

then SNEEZE $(D,\, f,\, \ell)$ *returns* $\{r^{(i)}\}_{i=0}^{n-1}$ *where* $r^{(i)} = d_i$ *for* $i \in \{0, \ldots, n-1\}$.

Proof. We need to uniquely encode any possible d as an integer modulo $f(2^\ell)$. Since the encodings of the coefficients d_i are ℓ bits long, and we need to store n of them, this means that we require $f(2^\ell) > 2^{n\ell} - 1$.

When SNEEZE is called, we set

$$G^{[-1]} \coloneqq D = A \cdot B + C \bmod^{(+)} f(2^\ell).$$

Since $d \equiv a \cdot b + c \bmod f$, it follows by explicit computation that

$$G^{[-1]} = D = \sum_{i=0}^{n-1} d_i\, 2^{\ell i} + \mathtt{b}\, f(2^\ell)$$

where the last equality is over the integers, for some $\mathtt{b} \in \mathbb{Z}$. Given that

$$\left|\sum_{i=0}^{n-1} d_i\, 2^{\ell i}\right| \leq \delta\, \frac{2^{n\ell}-1}{2^{\ell}-1} \leq (2^{\ell-1}-1)\, \frac{2^{n\ell}-1}{2^{\ell}-1} < 2^{n\ell-1},$$

the assumption that $f(2^\ell) > 2^{n\ell} - 1 > 2^{n\ell-1}$ implies that $\mathtt{b} \in \{0, 1\}$.

The main computation in Sneeze is done between lines 3 and 9, and amounts to signed coefficient unpacking of $d + \mathtt{b}f$ as described in Example 18. Formally, we define some conditions on the output of the i-th iteration of the loop and prove they hold by induction.

Claim 2. *After step $i \in \{0, \ldots, n-1\}$, we have*

$$r^{(i)} = d_i + \mathtt{b}\, f_i \tag{5.2}$$

and

$$G^{[i]} = \sum_{j=i+1}^{n-1} d_j\, 2^{\ell(j-i-1)} + \mathtt{b} \sum_{j=i+1}^{n} f_j\, 2^{(j-i-1)\ell} \tag{5.3}$$

Assume Conditions 5.2, 5.3 hold for step $i - 1 \geq 0$. We start on line 4 by assigning

$$\begin{aligned} e^{(i)} &= G^{[i-1]} \bmod^{(+)} 2^{\ell} \\ &= \sum_{r=i}^{n-1} d_r\, 2^{\ell(r-i)} + \mathtt{b} \sum_{j=i}^{n} f_j\, 2^{(j-i)\ell} \bmod^{(+)} 2^{\ell} \\ &= d_i + \mathtt{b}\, f_i + \mathtt{t}_i\, 2^{\ell} \end{aligned}$$

for some $\mathtt{t}_i \in \mathbb{Z}$ such that $e^{(i)} \in [0,\, 2^\ell - 1]$. Similar to before, by definition of ℓ and the fact that $\mathtt{b} \in \{0, 1\}$, we have

$$|d_i + \mathtt{b}\, f_i| \leq \delta + \varphi < 2^{\ell-1} \text{ for all } i \in \{0, \ldots, n-1\} \tag{5.4}$$

Hence $\mathtt{t}_i \in \{0, 1\}$ for $i < n$. We then set (line 5)

$$G^{[i]} = \frac{G^{[i-1]} - e^{(i)}}{2^{\ell}} = \sum_{r=i+1}^{n-1} d_r\, 2^{\ell(r-i-1)} + \mathtt{b} \sum_{j=i+1}^{n} f_j\, 2^{(j-i-1)\ell} - \mathtt{t}_i$$

and balance $e^{(i)} \bmod 2^\ell$ (lines 6–8). By the size consideration made in Inequality 5.4, this amounts to subtracting $\mathtt{t}_i 2^\ell$ from $e^{(i)}$. We keep account of this subtraction by adding back $\mathtt{t}_i$ to $G^{[i]}$ (line 8). Finally, we assign $r^{(i)} \leftarrow e^{(i)}$.

Hence Conditions 5.2, 5.3 hold for step $i \geq 1$. Similarly, we can see that Conditions 5.2, 5.3 also hold for step $i = 0$, proving the claim.

By Condition 5.3 and f being monic, after step $i = n - 1$ we have $G^{[n-1]} = \mathtt{b} < 2^\ell$, which would become the coefficient of an n-th power of x in d. Line 10 takes care of reducing this modulo f, which results in assigning

$$r^{(i)} \leftarrow r^{(i)} - f_i\, G^{[n-1]} = d_i + \mathtt{b}\, f_i - f_i\, \mathtt{b} = d_i \quad \text{for all } i < n,$$

completing the proof. □

Since operating on $G^{[i]}$ involves integer arithmetic on $n\ell$ bit integers, we may modify Algorithm 20 to correct carries on $e^{(i)}$ in order to avoid executing line 8 of Algorithm 20. This variant of the algorithm is given as Algorithm 21. Note that with this change the only large integer operations are division with remainder modulo 2^ℓ and thus cheap, while the final output of the algorithm is the same.

Lemma 20 can be in particular used in the case where f is a power of two cyclotomic polynomial or a prime cyclotomic polynomial.[8] This results in the following corollaries.

Corollary 4 (Power of two cyclotomic)**.** *Let α, β, γ be as above, let n be a power of 2, and let $f(x) = x^n + 1$. Let $\delta \coloneqq n\alpha\beta + \gamma$. Then Lemma 20 applies.*

Proof. We need to verify that

$$f(2^\ell) > 2^{n\ell} - 1 \tag{5.5}$$

and that

$$|d_i| \leq \delta \tag{5.6}$$

Condition 5.5 holds since $f(2^\ell) = 2^{n\ell} + 1$. Condition 5.6 follows by explicitly evaluating

$$d(x) = \sum_{i=0}^{n-1} d_i\, x^i \coloneqq a(x) \cdot b(x) + c(x) \pmod{x^n + 1}$$

[8] The latter being proposed by the LIMA team [SAL+17] for use with safe-primes, as to avoid unsafe error distributions resulting from error sampling in the coefficient embedding.

which implies that

$$d_i = \sum_{[j+k]_n = i} -1^{[\![j+k \geq n]\!]}\, a_j\, b_k + c_i$$

and hence

$$\max_{\substack{\{a_j\}_j, \{b_k\}_k, \\ \{c_m\}_m}} |d_i| \leq n\alpha\beta + \gamma =: \delta.$$

□

Corollary 5 (Prime cyclotomic)**.** *Let α, β, γ be as above, let $n = p - 1$ where p is prime, and let $f = \sum_{i=0}^{n} x^i$. Let $\delta := (2n-1)\alpha\beta + \gamma$. Then Lemma 20 applies.*

To prove Corollary 5, we first need the following lemma.

Lemma 21. *Let $a = \sum_{i=0}^{n-1} a_i\, x^i$, $b = \sum_{i=0}^{n-1} b_i\, x^i$ with $a_i, b_i \in \mathbb{Z}$, and let $f = \sum_{i=0}^{n} x^i$. Let $c_i := \sum_{j+k=i} a_j b_k$ such that $c := \sum_{i=0}^{2n-2} c_i\, x^i = a \cdot b$ and let $d := \sum_{i=0}^{n-1} d_i\, x^i \equiv c \pmod f$. Then*

$$d = \sum_{i=0}^{n-3} (c_i - c_n + c_{i+n+1})\, x^i + (c_{n-2} - c_n)\, x^{n-2} + (c_{n-1} - c_n)\, x^{n-1}$$

and each d_i is a sum of at most $2n-1$ terms of the form $a_j b_k$.

Proof. Let $f^{(m)} := \sum_{i=0}^{m} x^i$ (it follows that $f \equiv f^{(n)}$). Since a and b have degree $< n$, we know that we need to reduce modulo f only the powers x^{i+n} for $i = 0, \ldots, n-2$ of c. For $i \geq 1$ we have

$$\begin{aligned} x^{i+n} &\equiv x^i(x^n - f^{(n)}(x)) \pmod f \\ &= -x^i(f^{(n-1)}) \\ &= -x^{i-1}(xf^{(n-1)}) \\ &= -x^{i-1}(f^{(n)} - 1) \\ &\equiv x^{i-1} \pmod f, \end{aligned}$$

while for $i = 0$, $x^n \equiv -f^{(n-1)} \pmod f$. Hence, we can write

$$\begin{aligned} c &= \sum_{i=0}^{2n-2} c_i\, x^i \\ &= \sum_{i=0}^{n-1} c_i\, x^i + c_n\, x^n + \sum_{i=0}^{n-3} c_{n+i+1}\, x^{n+i+1} \\ &\equiv \sum_{i=0}^{n-1} c_i\, x^i - c_n \sum_{i=0}^{n-1} x^i + \sum_{i=0}^{n-3} c_{n+i+1}\, x^i \pmod f \\ &\equiv \sum_{i=0}^{n-3} (c_i - c_n + c_{n+i+1})\, x^i + (c_{n-2} - c_n)\, x^{n-2} + (c_{n-1} - c_n) x^{n-1} \pmod f \end{aligned}$$

where each c_i is a sum of

$$\#\{(j,k) \in [0, n-1]^2 \cap \mathbb{Z}^2 \mid j + k = i\} = n - |i - n + 1| \tag{5.7}$$

terms $a_j b_k$, where (5.7) can be shown by considering first the case where $i < n$ (easy), and the case where $i = n + h$ for some $h \geq 0$ (we need $j \in [0, n-1]$ and $k = n + h - j \in [0, n-1]$; check for how many j the constraint $k \leq n - 1$ can be satisfied with a given h, this is $n - 1 - h = 2n - i - 1$).

Hence,

$$d = \sum_{i=0}^{n-3} (c_i - c_n + c_{n+i+1})\, x^i + (c_{n-2} - c_n)\, x^{n-2} + (c_{n-1} - c_n) x^{n-1}$$

where by explicit computation d_{n-1} is a sum of $2n-1$ terms $a_j b_k$, d_{n-2} is a sum of $2n-2$ such terms and, for $i \leq n-3$, d_i has $3n - |i - n + 1| - |n - n + 1| - |n + i + 1 - n + 1| = 2n - 2$ such terms. □

We can now prove Corollary 5.

Proof of Corollary 5. We need to verify that

$$f(2^\ell) > 2^{n\ell} - 1 \tag{5.8}$$

and that

$$|d_i| \leq \delta \tag{5.9}$$

Condition 5.8 holds since $f(2^\ell) = 2^{n\ell} + 2^{(n-1)\ell} + \cdots + 1$. Condition 5.9 follows by explicitly evaluating

$$d = \sum_{i=0}^{n-1} d_i \, x^i \equiv a \cdot b + c \pmod{f}$$

using Lemma 21, which implies that

$$\max_{\substack{\{a_j\}_j, \{b_k\}_k, \\ \{c_m\}_m}} |d_i| \leq (2n-1)\alpha\beta + \gamma =: \delta.$$

□

The proof of Lemma 20 can also be directly adapted to the MLWE setting where we let

$$\left\{a_i = \sum_{j=0}^{n-1} a_{i,j} x^j\right\}_{i=1}^{\kappa}, \quad \left\{b_i = \sum_{j=0}^{n-1} b_{i,j} x^j\right\}_{i=1}^{\kappa}, \quad c = \sum_{j=0}^{n-1} c_j x^j$$

with $|a_{i,j}| \leq \alpha$, $|b_{i,j}| \leq \beta$, and $|c_j| \leq \gamma$ and want to compute $\sum_{i=1}^{\kappa} a_i \cdot b_i + c \pmod{f}$, by letting

$$\ell > \log_2(\kappa(\delta - \gamma) + \gamma + \varphi) + 1.$$

Overall, we arrive at the following corollary.

Corollary 6 (KyberMulAdd)**.** *Let a_i, b_i, $c \in \mathbb{Z}[x]$ be as above, with $\alpha = \lfloor \frac{q}{2} \rfloor$, and $\beta = \gamma = \eta$, and let $f = x^n + 1$. Let*

$$\ell > \log_2 \left(\kappa n \left\lfloor \frac{q}{2} \right\rfloor \eta + \eta + 1\right) + 1$$

be an integer. Let $A_i \coloneqq \textsc{Snort}(a_i, f, \ell)$, $B_i \coloneqq \textsc{Snort}(b_i, f, \ell)$, $C \coloneqq \textsc{Snort}(c, f, \ell)$, and $D \coloneqq \mathbf{A} \cdot \mathbf{B} + C \mod^{(+)} f(2^\ell)$. Then $\textsc{Sneeze}(D, f, \ell)$ returns $d \coloneqq \sum_{i=1}^{\kappa} a_i \cdot b_i + c \pmod{f}$.

Remark 22. *From $d \in R$, the result in R_q can be obtained by coefficient-wise modular reduction.*

Input: $G \in \{0, \ldots, f(2^\ell) - 1\}$
Input: $f \in \mathbb{Z}[x]$, monic
Input: bitlength ℓ
1 $n \leftarrow \deg(f)$
2 $G^{[-1]} \leftarrow G$, $c \leftarrow 0$
3 **for** $i \leftarrow 0$ **to** $n-1$ **do** `// step` i
4 $\quad e^{(i)} \leftarrow G^{[i-1]} \bmod^{(+)} 2^\ell$
5 $\quad G^{[i]} \leftarrow \left(G^{[i-1]} - e^{(i)}\right) / 2^\ell$
6 $\quad e^{(i)} \leftarrow e^{(i)} + c$
7 $\quad$ **if** $e^{(i)} > 2^{\ell-1}$ **then** $e^{(i)} \leftarrow e^{(i)} - 2^\ell$, $c \leftarrow 1$ **else** $c \leftarrow 0$
8 $\quad r^{(i)} \leftarrow e^{(i)}$
9 **for** $i \leftarrow 0$ **to** $n-1$ **do**
10 $\quad r^{(i)} \leftarrow r^{(i)} - f_i \left(G^{[n-1]} + c\right)$ `// subtract b · f(x)`
11 **return** $\{r^{(i)}\}_{i=0}^{n-1}$

Algorithm 21: SNEEZE-FAST(G, f, ℓ). Same as SNEEZE, but avoiding large integer arithmetic for carry propagation. The label $G^{[i]}$ represents the state of the $n\ell$-bit integer variable G at step i.

5.3.1 Compact Kronecker substitution

In [Har09], David Harvey presents two improved packing techniques for Kronecker substitution, reducing integer sizes at the cost of performing more multiplications: KS2 or "negated evaluation points" evaluates at $\left(2^\ell, -2^\ell\right)$ and KS3 or "reciprocal evaluation points" evaluates at $\left(2^\ell, 2^{-\ell}\right)$. Each technique halves the required integer bit size at the cost of performing two multiplications. Note that integer arithmetic is super-linear (e.g. Karatsuba multiplication is used for medium-sized inputs and has a cost of $3^{\log_2 L}$ multiplications for integers of size L, see below) and thus this trade-off produces a noticeable speed-up. The two techniques are orthogonal and can be combined, which reduces bit sizes by a factor of four at the cost of increasing the number of multiplications to four. The combined algorithm is referred to as KS4.

The KS2 algorithm proceeds as follows. Assume $a(x), b(x)$ are such that their product $c(x) \coloneqq a(x) \cdot b(x)$ has positive coefficients bounded by $2^{2\ell}$. Let

$$\begin{aligned} c^{(+)} &\coloneqq \quad c(2^\ell) = a(2^\ell) \cdot b(2^\ell) \quad &&= \sum_{[i]_2=0} c_i \, 2^{i\ell} + \sum_{[i]_2=1} c_i \, 2^{i\ell} \\ c^{(-)} &\coloneqq c(-2^\ell) = a(-2^\ell) \cdot b(-2^\ell) &&= \sum_{[i]_2=0} c_i \, 2^{i\ell} - \sum_{[i]_2=1} c_i \, 2^{i\ell} \end{aligned}$$

Then, we can recover the even coefficients of $c(x)$ from

$$c^{(+)} + c^{(-)} = c(2^\ell) + c(-2^\ell) = 2 \sum_{[i]_2=0} c_i \, 2^{i\ell}$$

and the odd coefficients from

$$c^{(+)} - c^{(-)} = c(2^\ell) - c(-2^\ell) = 2 \cdot 2^\ell \sum_{[i]_2=1} c_i \, 2^{(i-1)\ell}$$

since the sum and the difference cancel out either the even or the odd powers. The coefficients can be either read directly with care to their offset, or dividing the above quantities by the appropriate power of 2 over the integers.

The KS2 algorithm is compatible with arithmetic modulo $f = x^n + 1$, when n is even. When doing this over $\mathbb{Z}_{f(2^\ell)}$ some care must be taken since reducing $c^{(\cdot)}$ modulo $f(2^\ell)$ may change its parity. In such case the coefficients can be recovered by either multiplying $c^{(+)} + c^{(-)}$ by $2^{-1} \bmod^{(+)} f(2^\ell)$ and $c^{(+)} - c^{(-)}$ by $2^{-\ell-1} \bmod^{(+)} f(2^\ell)$, or multiplying both quantities by a desired power of 2 modulo $f(2^\ell)$ and reading the coefficients with the appropriate offset. Packing and unpacking are identical to standard Kronecker substitution, i.e. the proof of Lemma 20 applies directly when working with such an f.

On the other hand, adapting packing and unpacking to combine the KS3 algorithm with modular reduction is somewhat more involved, requiring a fair amount of careful bit shifting. Implementing this strategy would roughly half the number of multiplications required at the cost of a more involved packing/unpacking algorithm. However, since our packing and unpacking routines already take time comparable to the actual multiplications they facilitate and since our target platform does not have efficient bit-shift operations, we did not attempt an implementation of KS3 or KS4.

5.4 Splitting the ring

Commercially available multipliers are usually capable of evaluating $(x, y, z) \mapsto x \cdot y \pmod z$ where $\log x, \log y, \log z < m$ for some fixed value of m which may be lower than what is required to apply Lemma 20 directly. In fact,

for typical parameter sizes of lattice-based cryptography and of RSA, this is expected to be the case. Thus, in this section – where we focus on $f = x^n + 1$ with n a power of two – we explain our strategy for utilising these "too small" multipliers.

Let $a(x)$, $b(x)$, $c(x)$ be polynomials of degree $< n$ as defined in Lemma 20 and let ℓ be the packing length used, we want to compute $a(x) \cdot b(x) + c(x) \pmod{f(x)}$. So far we have considered two ways of doing this. First, we can pack every coefficient of each polynomial individually in a large enough buffer, say of length ℓ, and then directly compute the result using polynomial arithmetic. Alternatively, we can use Lemma 20 and evaluate $a(2^\ell) \cdot b(2^\ell) + c(2^\ell) \bmod^{(+)} (2^{n\ell} + 1)$ packing all the coefficients of each polynomial at once in a buffer of length $n\ell + 1$, and then unpack the final result. A third option consists of interpolating between these two methods by combining Kronecker substitution with (typically low-degree) polynomial arithmetic in order to shorten the lengths of the multiplier's inputs. This approach is similar to fast integer multiplication algorithms by Schönhage [Sch77] or Nussbaumer [Nus80].

The idea is the following. Say we have

$$a(x) = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 \quad \text{and} \quad b(x) = b_0 + b_1\,x + b_2\,x^2 + b_3\,x^3$$

and we want to compute $a(x) \cdot b(x) \pmod{x^4 + 1}$, i.e.

$$\begin{aligned}&(a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3)\,x^3 + (a_2b_0 + a_1b_1 + a_0b_2 - a_3b_3)\,x^2\\ &+(a_1b_0 + a_0b_1 - a_3b_2 - a_2b_3)\,x \;\; + \;\; a_0b_0 - a_3b_1 - a_2b_2 - a_1b_3\end{aligned}$$

but we have a multiplier that would only let us work modulo $x^2 + 1$ given the ℓ required by Lemma 20. Letting $y = x^2$, we can write $a(x, y) = a^{(0)}(y) + a^{(1)}(y)\,x$ where

$$a^{(0)}(y) = a_0 + a_2\,y \quad \text{and} \quad a^{(1)}(y) = a_1 + a_3\,y,$$

and similarly for $b = b(x, y)$. Then, computing $a(x, y) \cdot b(x, y) \pmod{y^2 + 1}$ can be accomplished by packing $A^{(\cdot)} = \textsc{Snort}(a^{(\cdot)})$, $B^{(\cdot)} = \textsc{Snort}(b^{(\cdot)})$, and multiplying

$$\begin{aligned}\hat{C}(x) &\coloneqq a(x, 2^\ell) \cdot b(x, 2^\ell) \bmod^{(+)} 2^{2\ell} + 1\\ &= (A^{(0)} + A^{(1)}\,x) \cdot (B^{(0)} + B^{(1)}\,x) \bmod^{(+)} 2^{2\ell} + 1,\end{aligned}$$

where the coefficients $A^{(\cdot)} = a^{(\cdot)}(y)|_{y=2^\ell}$ and $B^{(\cdot)}$ can be multiplied on the coprocessor since now the substitution is $y = x^2 = 2^\ell$, meaning the packed polynomials will fit the smaller multiplier. If we were to unpack the coefficients of $\hat{C}(x)$, we would obtain

$$\begin{aligned}&(a_1b_1 - a_3b_3 + (a_3b_1 + a_1b_3)\,y)\,x^2 + a_0b_0 - a_2b_2 + (a_2b_0 + a_0b_2)\,y\\ &+(a_1b_0 + a_0b_1 - a_3b_2 - a_2b_3 + (a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3)\,y)\,x.\end{aligned}$$

Note that the coefficients on the second line match our target, but the coefficients on the first line do not (they are not grouped correctly and the signs do not necessarily match). This can be corrected by using the identity $y = x^2$ and thus rewriting $x^2 \to y$ and reducing again modulo y^2+1 (equivalently, rewriting $x^2 \to 2^\ell$ and reducing modulo $2^{2\ell}+1$). From our intermediate representation $\hat{C}(x) = \hat{C}_0 + \hat{C}_1\,x + \hat{C}_2\,x^2$, this can be done by defining $C(x) = C_0 + C_1\,x$ with

$$C_0 \coloneqq \left(\hat{C}_0 + (2^\ell \cdot \hat{C}_2 \bmod^{(+)} 2^{2\ell}+1)\right) \bmod^{(+)} 2^{2\ell}+1 \quad \text{and} \quad C_1 = \hat{C}_1,$$

and then unpacking $C(x)$ to obtain $a \cdot b \pmod{x^4+1}$.

More generally, this can be formally described as follows. Let $n = m \cdot \omega$, where $m \ll \omega$. Given a polynomial $p(x) = \sum_{i=0}^{n-1} p_i x^i$ of degree $< n$, we can set $y = x^m$, and then rewrite p as

$$\begin{aligned}p(x,y) &= \left(p_0 + p_{0+m}\,y + \cdots + p_{0+(\omega-1)m}\,y^{\omega-1}\right)\,x^0\\ &\quad + \left(p_1 + p_{1+m}\,y + \cdots + p_{1+(\omega-1)m}\,y^{\omega-1}\right)\,x^1\\ &\quad + \ldots\\ &\quad + \left(p_{m-1} + p_{m-1+m}\,y + \cdots + p_{m-1+(\omega-1)m}\,y^{\omega-1}\right)\,x^{m-1}\\ &= p^{(0)}(y) + p^{(1)}(y)\,x + \cdots + p^{(m-1)}(y)\,x^{m-1}\end{aligned}$$

where we write $p^{(i)}(y) \coloneqq \sum_{j=0}^{\omega-1} p_{i+jm}\,y^j$, polynomials in y of degree $< \omega$ (i.e. $p^{(i)} \leftarrow \textsc{Ff}(p,m,i)$, cf. Algorithm 22). The idea is to pack each $p^{(i)}$, $p \in \{a,b,c\}$, into buffers $P^{(i)} \coloneqq p^{(i)}(2^\ell) \bmod^{(+)} (2^{\omega\ell}+1)$ of length $\omega\ell+1$, and then evaluate

$$a(x,2^\ell) \cdot b(x,2^\ell) + c(x,2^\ell) \bmod^{(+)} (2^{\omega\ell}+1),$$

where $p(x,2^\ell) \equiv \sum_{i=0}^{m} P^{(i)}\,x^i$. By Lemma 20, the integer modulo operation will act on the coefficients as reduction modulo $y^\omega + 1 \equiv x^n + 1 \pmod{y - x^m}$ would.

Working with polynomials $a(x, y)$, $b(x, y)$, the resulting polynomial $a(x, y) \cdot b(x, y)$ will be a linear combination of monomials of the form $y^i\, x^j$. If we were to substitute $x^m = y$ back now, we would obtain monomials of degree $\geq n$ every time that $im + j \geq n$, which we do not want. Furthermore, depending on how we index the $y^i\, x^j$ in our code, we may be in need of combining ("grouping") constant coefficients from different monomials $y^i\, x^j \neq y^r\, x^s$ mapping to the same power of x.

To better see what adjustments need to be done to the resulting polynomial in x, we look at $a(x, y) \cdot b(x, y) \pmod{y^\omega + 1}$ in detail.

$$\begin{aligned} a(x,y) \cdot b(x,y) &= \sum_{i,r=0}^{m-1} a^{(i)}(y)\, b^{(r)}(y)\, x^{i+r} \\ &= \sum_{i,r=0}^{m-1} \sum_{j,s=0}^{\omega-1} a_{i+jm}\, b_{r+sm}\, y^{j+s}\, x^{i+r} \\ &\equiv \sum_{i,r=0}^{m-1} \sum_{j,s=0}^{\omega-1} (-1)^{[\![j+s \geq \omega]\!]} a_{i+jm}\, b_{r+sm}\, y^{[j+s]_\omega}\, x^{i+r} \pmod{y^\omega + 1} \end{aligned}$$

Given that $y^{[j+s]_\omega}\, x^{i+r} \equiv x^{m \cdot [j+s]_\omega + i + r} \pmod{y - x^m}$, we can see that after reducing modulo $y^\omega + 1$ it will be necessary to further reduce modulo $y - x^m$ whenever $m \cdot [j + s]_\omega + i + r \geq n$, which can happen only if $i + r \geq m$. We do this by sending any monomial $y^j\, x^i$ where $i \geq m$ to $y^{j+1}\, x^{i-m} \pmod{y^\omega + 1}$, or equivalently by mapping monomials x^i with $i \geq m$ to $2^\ell\, x^{i-m}$, as done in Line 10 of Algorithm 23. This also takes care of groupings. Then, we can simply Sneeze every coefficient to obtain the final result. The full procedure results in Algorithms 22 and 23.

A possible optimisation could be that of choosing ℓ more aggressively. Indeed, we only ever need to pack polynomials of degree ω, and hence we could use this value in place of n. This would save $\approx \log m$ bits per packed coefficient while still being able to perform the reduction modulo $y^\omega + 1 \equiv 2^{\omega\ell} + 1$, overall resulting in a saving of size $\approx \omega \log m$ bits per packed polynomial $p^{(i)}(y)$. In this case one would need to unpack the $P^{(i)}$ before the second reduction and final grouping, and handle these afterwards on the CPU.

Input: polynomial $g \in R$
Input: step size m, dividing n
Input: offset o
1 $\omega \leftarrow n/m$
2 **return** $\sum_{j=0}^{\omega} g_{m \cdot j + o}\, x^j$

Algorithm 22: $\textsc{Ff}(g, m, o)$. Return a new polynomial containing every mth coefficient of g, starting at offset o.

At the heart of Algorithm 23 is polynomial multiplication of two, typically low-degree, polynomials in line 7. A straightforward choice to realise this multiplication is schoolbook multiplication. This has quadratic complexity but is a simple algorithm. Another natural option is Karatsuba multiplication [KO62]. In its simplest form, the algorithm computes the product $a + b \cdot x$ and $c + d \cdot x$ in $\mathbb{Z}[x]$ by computing the products $t_0 = a \cdot c$, $t_1 = b \cdot d$ and $t_2 = (a+b) \cdot (c+d) = ac + ad + bc + bd$ and outputting $t_0 + (t_2 - t_0 - t_1) \cdot x + t_2 x^2$. It has a cost of $3^{\lceil \log_2 L \rceil}$ multiplications for degree $L - 1$ polynomials. We note that finding better multiplication formulas for larger degrees is an active area of research [Mon05, FH07, CÖ10, BDEZ12].

5.5 Implementation

Using the strategies outlined in Sections 5.3 and 5.4, we are now ready to fix an implementation of Kyber and the KyberMulAdd gadget (see Corollary 6) using a big integer multiplier. We focus on the Kyber768 parameter set and implement our variants of the scheme on the Infineon SLE 78 (SLE 78CLUFX5000) equipped with an RSA, an AES and a SHA-256 coprocessor and 16 Kbyte RAM. All our software is native code written in C and assembly language.

5.5.1 Description of Kyber using Kronecker substitution

First we provide a description of our variant of Kyber.CPA that takes into account Kronecker substitution. The algorithms closely resemble the

Input: polynomial $a(x) \in R$
Input: polynomial $b(x) \in R$
Input: bitlength ℓ
Input: width parameter ω, dividing n
$f \leftarrow x^{\omega} + 1$
$m \leftarrow n/\omega$
// construct polynomials $A(x), B(x)$ of degree $< m$
for $i \leftarrow 0$ **to** $m - 1$ **do**
  $A_i \leftarrow \textsc{Snort}(\textsc{Ff}(a(x),\, m,\, i),\, f,\, \ell)$
  $B_i \leftarrow \textsc{Snort}(\textsc{Ff}(b(x),\, m,\, i),\, f,\, \ell)$
$F \leftarrow 2^{\omega\ell} + 1$
// polynomial multiplication modulo integer F
$\hat{C}(x) \leftarrow A(x) \cdot B(x) \bmod^{(+)} F$
// construct polynomial $C(x)$ of degree $< m$
$C_{m-1} \leftarrow \hat{C}_{m-1}$
for $i \leftarrow 0$ **to** $m - 2$ **do**
  $C_i \leftarrow \left(\hat{C}_i + \left(2^{\ell} \cdot \hat{C}_{m+i} \bmod^{(+)} F\right)\right) \bmod^{(+)} F$
// construct tuple $\hat{c}$ of polynomials $\hat{c}_i$ each of degree $< \omega$
for $i \leftarrow 0$ **to** $m - 1$ **do**
  $\hat{c}_i \leftarrow \textsc{Sneeze}\,(C_i,\, f,\, \ell)$
// construct polynomial $c(x)$ of degree $< n$
for $i \leftarrow 0$ **to** $\omega - 1$ **do**
  for $j \leftarrow 0$ **to** $m - 1$ **do**
    $c_{m \cdot i + j} \leftarrow (\hat{c}_j)_i$
return $c(x)$

Algorithm 23: $a(x) \cdot b(x) \bmod x^n + 1$ using an integer multiplier capable of performing modular multiplication of integers up to $\omega\ell + 1$ bits.

implementation on our target device and include certain optimisations for performance and reduction of memory consumption.

In Algorithm 24 we describe our implementation of Kyber.CPA.Gen[9] and follow the notation of [SAB+17] where appropriate. The sampling of a uniform polynomial $a_{i,j} \in \mathbf{A}$ is done by $\textsc{Parse}(\text{XOF}(\rho||i||j))$ for a random seed $\rho \in \{0,1\}^{256}$ using an Extendable Output Function (XOF) denoted as $\text{XOF}(\cdot)$. The sampling of a secret or noise polynomial in R_q is described by $\text{CBD}(\text{PRF}(\sigma, N))$ where CBD stands for centred binomial distribution and where PRF is a pseudo-random function (PRF) that takes a random seed $\sigma \in \{0,1\}^{256}$ and an

[9] Instead of using SHA3-512 to hash the randomness, we directly take the output from the on-chip TRNG using the $\text{TRNG}(\cdot)$ function; see below.

integer N for domain separation. In [SAB+17] it is specified that $\text{PRF}(\sigma, N) = \text{SHAKE-256}(\sigma, N)$ and that $\text{XOF} = \text{SHAKE-128}$.

With regard to arithmetic, it is easy to see that $s_0, \ldots, s_{k-1}$ are used k times each, when computing $\mathbf{A} \cdot \mathbf{s}$. Thus a straightforward optimisation is to pack them into a big integer only once. This resembles some similarity to the NTT, where it is also possible to achieve speed-ups by the very simple observation that polynomials that are used several times have to be transformed into the NTT domain only once. To obtain more control over the usage of Snort and Sneeze, which is already integrated into the high-level gadget KyberMulAdd, we split KyberMulAdd into sub-functions. The $\hat{C} = \text{MulAddSingle}(A, B, C)$ function takes as input $A = \text{Snort}(a)$, $B = \text{Snort}(b)$, $C = \text{Snort}(c)$ for $a, b, c \in R_q$ and computes $\hat{D}(x) \leftarrow A(x) \cdot B(x) + C(x) \bmod^{(+)} F$ as specified in line 7 of Algorithm 23. The $D = \text{FinalEll}(\hat{D})$ function takes $\hat{D}$ and constructs the polynomial $D(x)$ of degree $< m$ (line 10 of Algorithm 23) by multiplying by 2^{ℓ}. To save stack memory we do not generate the full matrix $\mathbf{A}$ but only one coefficient after the other. All in all, our approach to key generation requires $k^2 + 2k$ calls to Snort, k^2 big integer multiplications realised by MulAddSingle and k calls to Sneeze as well as FinalEll.

CPA-secure Kyber encryption is described in Algorithm 25 where the computation of $\mathbf{A}^T\mathbf{r}+\mathbf{e}_1$ can be realised in the same way as the key generation procedure by packing each polynomial of $\mathbf{r}$ into $\mathbf{R} \in \mathbb{Z}^k$ only once and with on-the-fly generation of polynomials of $\mathbf{A}$ to save stack memory. The only difference is that we initialise $\hat{U}_{tmp}$ with on-the-fly sampled and packed error polynomials $e_i \in \mathbf{e}_1$ before computing the k scalar products. For $\langle \mathbf{t}, \mathbf{r} \rangle + e_2 + \lfloor \frac{q}{2} \rceil \cdot m$ we sample e_2 by $e \leftarrow \text{CBD}(\text{PRF}(\sigma, N))$, set $\hat{V} \leftarrow \text{Snort}(e + \bar{m})$ and then compute the scalar product in a loop with $\hat{V} \leftarrow \text{MulAddSingle}(R_i, T_{tpm}, \hat{V})$. All in all, Kyber.CPA.Imp.Enc requires $k^2 + 3k + 1$ calls to Snort, $k^2 + k$ big integer multiplications by MulAddSingle and $k + 1$ calls to Sneeze as well as FinalEll.

In Algorithm 26 we describe CPA-secure Kyber decryption. The implementation of the scalar product to compute $\langle \mathbf{s}, \mathbf{u} \rangle$ follows the approach from encryption. To reuse MulAddSingle and to save code needed for a subtrac-

$\rho \xleftarrow{\$} \text{TRNG}()$ // $\rho \in \{0,1\}^{256}$ sampled from internal TRNG
$\sigma \xleftarrow{\$} \text{TRNG}()$ // $\sigma \in \{0,1\}^{256}$ sampled from internal TRNG
$N \leftarrow 0$
// Sample s and transform to S
for $i \leftarrow k-1$ **to** 0 **do**
  $s_{tmp} \leftarrow \text{CBD}(\text{PRF}(\sigma, N))$
  $N \leftarrow N+1$
  $S_i \leftarrow \text{Snort}(s_{tmp})$
// Compute $\mathbf{As}+\mathbf{e}$
for $i \leftarrow 0$ **to** $k-1$ **do**
  $e \leftarrow \text{CBD}(\text{PRF}(\sigma, N))$
  $N \leftarrow N+1$
  $\hat{T} \leftarrow \text{Snort}(e)$
  for $j \leftarrow 0$ **to** $k-1$ **do**
    $a_{tmp} \leftarrow \text{Parse}(\text{XOF}(\rho||i||j))$
    $A_{tmp} \leftarrow \text{Snort}(a_{tmp})$
    $\hat{T} \leftarrow \text{MulAddSingle}(A_{tmp}, S_j, \hat{T})$
  $T \leftarrow \text{FinalEll}(\hat{T})$
  $t_i \leftarrow \text{Sneeze}(T)$
$pk \leftarrow \text{Encode}_{d_t}(\text{Compress}_q(\mathbf{t}, d_t)||\rho)$
$sk \leftarrow \text{Encode}_{13}(\mathbf{s} \bmod^{(+)} q)$
return $pk_{CPA} \coloneqq pk$, $sk_{CPA} \coloneqq sk$

Algorithm 24: Kyber.CPA.Imp.Gen, function names follow [SAB+17].

tion gadget we first negate v by computing $\hat{V} \leftarrow \text{Snort}(-v)$ and then negate the final result again as $\text{Compress}_q(-v, 1)$ to obtain $v - \langle \mathbf{s}, \mathbf{u} \rangle$. We need $2k+1$ calls of Snort, k big integer multiplications by MulAddSingle and one call to Sneeze as well as FinalEll.

5.5.2 Implementation of Kyber on SLE 78

We now give details of our implementation of CPA and CCA-secure Kyber768 (thus $k = 3$) on the SLE 78 that are independent of the chosen approach for packing and big integer multiplication (see Section 5.5.3 and Section 5.5.4). All our implementations are not fully compatible with the specification as Kyber is explicitly defined with a specific NTT and assumes that the pseudo-random polynomials of $\mathbf{A}$ are already output by the sampler in the NTT domain.

Input: $m \in \mathcal{M}$
Input: pk_{CPA}
$\mathbf{t}, \rho \leftarrow \text{DECODE}_{d_t}(pk_{\text{CPA}})$
$\mathbf{t} \leftarrow \text{DECOMPRESS}_q(\mathbf{t})$
$N \leftarrow 0$
// Sample MLWE secret **r** and transform to **R**
for $i \leftarrow k-1$ **to** 0 **do**
  $r_{tmp} \leftarrow \text{CBD}(\text{PRF}(\sigma, N))$
  $N \leftarrow N+1$
  $R_i \leftarrow \text{SNORT}(r_{tmp})$
// Compute $\mathbf{A}^T\mathbf{r} + \mathbf{e_1}$
for $i \leftarrow 0$ **to** $k-1$ **do**
  $e \leftarrow \text{CBD}(\text{PRF}(\sigma, N))$
  $\hat{U}_{tmp} \leftarrow \text{SNORT}(e)$
  $N \leftarrow N+1$
  for $j \leftarrow 0$ **to** $k-1$ **do**
    $a_{tmp} \leftarrow \text{PARSE}(\text{XOF}(\rho||i||j))$
    $A_{tmp} \leftarrow \text{SNORT}(a_{tmp})$
    $\hat{U}_{tmp} \leftarrow \text{MULADDSINGLE}(A_{tmp}, R_j, \hat{U}_{tmp})$
  $U_{tmp} \leftarrow \text{FINALELL}(\hat{U}_{tmp})$
  $u_i \leftarrow \text{SNEEZE}(U_{tmp})$
// Compute $\langle \mathbf{t}, \mathbf{r} \rangle + e_2$
$\bar{m} \leftarrow \text{ENCODEMSG}(m)$
$e \leftarrow \text{CBD}(\text{PRF}(\sigma, N))$
$e \leftarrow e + \bar{m}$
$\hat{V} \leftarrow \text{SNORT}(e)$
for $i \leftarrow 0$ **to** $k-1$ **do**
  $T_{tpm} \leftarrow \text{SNORT}(t_i)$
  $\hat{V} \leftarrow \text{MULADDSINGLE}(R_i, T_{tpm}, \hat{V})$
$V \leftarrow \text{FINALELL}(\hat{V})$
$v \leftarrow \text{SNEEZE}(V)$
// Encode ciphertext
$c_1 \leftarrow \text{ENCODE}_{d_u}(\text{COMPRESS}_q(\mathbf{u}, d_u))$
$c_2 \leftarrow \text{ENCODE}_{d_v}(\text{COMPRESS}_q(v, d_v))$
return $c \coloneqq (c_1||c_2)$

Algorithm 25: KYBER.CPA.IMP.ENC, function names follow [SAB+17].

To expand randomness into a longer bitstream, Kyber originally specifies the use of various instances from the SHA3 family as PRNG (originally, XOF is SHAKE-128 and PRF is SHAKE-256). We implemented one version of the samplers that is compatible with the specification where SHAKE-128 and

Input: $c := (c_1||c_2)$
Input: sk_{CPA}
1 $\mathbf{s} \leftarrow \text{Decode}_{13}(sk_{\text{CPA}})$
2 $\mathbf{u} \leftarrow \text{Decompress}_q(\text{Decode}_{d_u}(c_1))$
3 $v \leftarrow \text{Decompress}_q(\text{Decode}_{d_v}(c_2))$
4 $\hat{V} \leftarrow \text{Snort}(-v)$
// Compute $v - \langle \mathbf{s}, \mathbf{u} \rangle$
5 **for** $i \leftarrow 0$ **to** $k-1$ **do**
6 $\quad U_{tmp} \leftarrow \text{Snort}(u_i)$
7 $\quad S_{tmp} \leftarrow \text{Snort}(s_i)$
8 $\quad \hat{V} \leftarrow \text{MulAddSingle}(S_{tmp}, U_{tmp}, \hat{V})$
9 $V \leftarrow \text{FinalEll}(\hat{V})$
10 $v \leftarrow \text{Sneeze}(V)$
11 **return** $\text{Encode}_1(\text{Compress}_q(-v, 1))$

Algorithm 26: Kyber.CPA.Imp.Dec, function names follow [SAB+17].

SHAKE-256 are realised in software. Hardware acceleration is not possible as our target device does not have a SHA3 hardware accelerator. The SHA3 implementation written in C has been optimised to some extent with assembly to remove obvious performance bottlenecks introduced by the compiler. Additionally, we have implemented a (non-compatible) Kyber variant that is using AES-256 in counter mode to implement XOF and PRF. A similar approach has been used by Google in their NewHope experiment where the constant polynomial a was also sampled using AES [Lan16]. Even though there are some theoretical concerns [ADPS16], this approach appears to be secure in practice. When AES-256 is chosen as PRNG we can rely on the AES coprocessor of the SLE 78CLUFX5000 and do not need to implement AES in software.

A difference that is not noticeable by a user is that we, as previously mentioned, do not hash the randomness provided to key generation due to the availability of a TRNG. The hashing of the input randomness in the Kyber specification is intended as a protection against leakage of the internal state of a random number generator. However, on our target device we have access to a certified RNG with appropriate post-processing and thus expensive computation of SHA3-512 is unnecessary.

The implementations of CBD, Parse, Encode, Decode and Decompress$_q$ follow the Kyber C reference implementation and are not particularly optimised using assembly. Our implementation of CCA-secure Kyber using the FO transformation is denoted as Kyber.CCA.Imp.Gen for key generation, Kyber.CCA.Imp.Enc for encapsulation and Kyber.CCA.Imp.Dec for decapsulation and we straightforwardly follow Algorithms 16 to 18. The main additional operations demanded by the CCA conversion are the computation of hash functions to implement random oracles. In one version of our implementation we follow the specification where H is using SHA3-256 and G is using SHA3-512 and where SHA3 is implemented in software. Additionally, we implemented a variant where G is realised by the MAC-based scheme HKDF [Kra10] using a SHA-256 coprocessor and where H is realised by a call to SHA-256. The usage of HKDF is necessary as the output of G has to be longer than a single SHA-256 hash.

5.5.3 Realisation of KyberMulAdd with KS1

The KyberMulAdd gadget consists of the functions Snort, MulAddSingle, FinalEll, and Sneeze. In case of KS1 (standard Kronecker substitution) parameters $(\omega, m) = (64, 4)$ can be used (see Algorithm 23 and Section 5.4). Then 64 coefficients can be packed into one integer and it is possible to perform polynomial arithmetic modulo $x^4 + 1$. When aiming for minimal size we could have used 25 bits of precision per coefficient and thus $64 \cdot 25 = 1600$ bits in total. However, to simplify the packing algorithm we have chosen 32 bits per coefficient (thus $\ell = 32$) which leads to integers of $64 \cdot 32 = 2048$ bits. This way no shifts by arbitrary integers are required as everything is immediately word aligned in Snort. This provides a performance advantage as the SLE 78 needs one cycle for each shift to the right or left. Moreover, the big integer multiplier is relatively fast and thus the trade-off between simpler packing/unpacking and slightly larger integer coefficients turned out to be favourable. However, on different platforms this may not be the case. An issue that costs some performance is the correct handling of carry bits caused by negative coefficients in Snort.

For a single big integer multiplication in MulAddSingle we use the RSA coprocessor on the SLE 78CLUFX5000 which has five registers of length slightly larger than 2048 bits. In a simplified model it is able to compute additions of two registers in 8 cycles while a multiplication with modular reduction takes roughly 9,300 cycles. However, not all registers are general purpose. One register is a working register that contains the result of a computation and is not directly accessible from the CPU. Another register is needed to store the modulus when performing operations modulo p. Thus three registers are available for temporary results or operands. Naturally, for an integer multiplication modulo $\log_2 p = 2048$, two registers are already occupied with operands.

For KS1 with parameters $(\omega, m) = (64, 4)$ and $\ell = 32$ one option to realise the polynomial multiplication $\hat{C}(x) \leftarrow A(x) \cdot B(x) \bmod^{(+)} F$ for $A, B, \hat{C} \in \mathbb{Z}_p[x]$ with $p = F = 2^{\omega\ell} + 1 = 2^{2048} + 1$ described in line 7 of Algorithm 23 would be schoolbook multiplication. As we have to do polynomial arithmetic modulo $x^4 + 1$ this would lead to $4^2 = 16$ multiplications in $\mathbb{Z}_p$ due to the quadratic complexity of schoolbook multiplication. To reduce the number of multiplications we have chosen Karatsuba multiplication for our KS1 implementation of the MulAddSingle function, which leads to 9 multiplications, 17 additions and 16 subtractions in $\mathbb{Z}_p$. These numbers include additions or subtractions required for the modulo $x^4 + 1$ operation. In general, Karatsuba multiplication leads to a large number of additions as a trade-off for fewer multiplications. An approach where the additions are executed on the RSA coprocessor would be possible but requires a lot of transfers. We thus decided to exploit the ability to run the coprocessor and the CPU in parallel. While the RSA coprocessor executes a modular multiplication we compute long integer additions in parallel on the CPU. This can easily be achieved by the appropriate rearrangement of multiplication and addition/subtraction operations in the Karatsuba formula. For simplicity, we give a sort example for $a(x) = a_0 + a_1\, x$ and $b(x) = b_0 + b_1\, x$. A polynomial multiplication can be computed with Karatsuba as $a(x)b(x) = a_0b_0 + ((a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0)\, x + a_1b_1\, x^2$. Here some additions can be performed in parallel to multiplications where $T_1 = a_1 \cdot b_1$ and $T_2 = b_1 + b_0$ is computed in parallel, then $T_3 = a_0 \cdot b_0$ and $T_4 = a_1 + a_0$, then $T_5 = T_2 \cdot T_4$ and $T_6 = T_1 + T_3$. Final additions and computations are

$T_7 = x^2 \cdot T_1$, $T_8 = T_5 - T_6$, $T_9 = x \cdot T_8$, $T_{10} = T_7 + T_9$, and $T_{11} = T_3 + T_{10}$ where $a(x)b(x) = T_{11}$. Note in our specific case also some additions or subtractions caused by the modulo $x^4 + 1$ operation are also hidden behind multiplications. For the remaining additions and subtractions we make use of the coprocessor. To save cycles for transfers we store the result of several additions/subtractions in one register of the coprocessor so that we only have to transfer values into the coprocessor and then read out the final result. The FinalEll function (see line 9 of Algorithm 23) requires 3 multiplications by 2^ℓ. They are implemented on the coprocessor using a special command that allows fast shifting by 32 bits and are thus relatively cheap.

5.5.4 Realisation of KyberMulAdd with KS2

The KyberMulAdd gadget can also be implemented for KS2 (compact Kronecker) with parameters $(\omega, m) = (128, 2)$. Compact Kronecker would allow to pack 128 coefficients into two big integers with 13 bits per coefficients. With 13 bits of precision per coefficient $13 \cdot 128 = 1664$ bits would be required in total. However, similarly to KS1 we use 16 bits for easier packing/unpacking and end up with integers of size of $16 \cdot 128 = 2048$ bits ($\ell = 16$). Computations are then performed on two polynomials modulo $x^2 + 1$. This leads to $2 \cdot 2^2 = 8$ multiplications in $\mathbb{Z}_p$ for $p = F = 2^{\omega\ell} + 1$ when using schoolbook multiplication. With Karatsuba a reduction to $2 \cdot 3 = 6$ multiplication would be possible. As the difference between Karatsuba and schoolbook is small we use schoolbook multiplication to implement KS2. This allows us to store partial products during schoolbook multiplication in the free register of the RSA coprocessor. This way we can perform additions with the RSA coprocessor and save time as we do not have to retrieve every result from the coprocessor into memory.

5.5.5 MulAdd for higher degree polynomials: a NewHope example

In a similar fashion to Sections 5.5.3 and 5.5.4, one could choose to implement MulAdd for RLWE-based schemes working with polynomials of higher degree. For example, NewHope [PAA+17] proposes the following set of parameters

(NewHope512) targetting Category 1 security [Nat16]: $n = 512, q = 12289, \eta = 8, f = x^n + 1$. Lemma 20 suggests using $\ell = 26$ (resp. $\ell = 13$) bits of precision per coefficient for use with KS1 (resp. KS2). To further improve packing and unpacking performance, we consider $\ell = 32$ (resp. $\ell = 16$), which results in parameters $(\omega, m) = (64, 8)$ with $32 \cdot 64 = 2048$ bits per polynomial coefficient (resp. $(\omega, m) = (128, 4)$ with $16 \cdot 128 = 2048$ bits per polynomial coefficient), assuming our integer multiplier supports inputs of length $2048 + 1$ bits. Schoolbook multiplication would then require $8^2 = 64$ (resp. $2 \cdot 4^2 = 32$) multiplications in $\mathbb{Z}_p$ for $p = F = 2^{\omega\ell} + 1$, while recursively applying Karatsuba would result in $3^{\log_2 8} = 27$ (resp. $2 \cdot 3^{\log_2 4} = 18$) multiplications. While this outline gives us a rough estimate of the cost of implementing NewHope512 using different strategies, it does not take into account concrete implementation issues such as the size and number of registers available in the CPU, the number of additions required, or the possible speed-ups from running light operations on the CPU while waiting for the modular multiplier to return a result for each multiplication.

5.6 Performance and comparison

In this section we describe the performance of our Kyber768 implementation on the SLE 78 and compare our results to related work available at the time of publication. All cycle counts are averages of several runs and have been measured on a cycle accurate FPGA-based emulator.

5.6.1 Implementation performance

In Table 5.2 we provide cycle counts of our implementation of Kyber768, its variants, and selected sub-functions. The results show similar performance for the KS1 and KS2 approach in Kyber.CPA.Imp with a small advantage for KS2. The explanation is that KS1 with Karatsuba requires only a single multiplication more than KS2 with schoolbook. The additional additions necessary for Karatsuba in KS1 can effectively be hidden by running them in parallel with the RSA coprocessor and Snort for KS1 is roughly twice

as fast than for KS2. However, this is only a conclusion for the particular parameters using the specific coprocessor. KS1 and KS2 might lead to very different results in case our approach would be used to implement a scheme like NewHope where the polynomial degree n is much larger than in Kyber. Cycle counts for CBD and Parse show that usage of the AES coprocessor provides a significant speed-up compared to the SHA3 software implementation. For CBD the difference is a factor of 300 and for Parse even a factor of 945. With more optimization of the SHA3 software, e.g. by writing it fully in assembly, it might be possible to reduce this to some extent. An additional advantage is that the AES coprocessor already implements some countermeasures against physical attacks. Such attacks are not the focus of our work but a secured PRNG would be easier to realise with the AES hardware (HW) coprocessor than by using a shared software (SW) implementation of SHA3 (see [OSPG18] where this necessity is discussed and performance of a shared SHA3 is given). With roughly $\approx 376,000$ cycles used for sampling in Kyber.CPA.Imp.Gen ($\approx 9 \times$ Parse+$6\times$CBD) and roughly $\approx 407,000$ cycles used in Kyber.CPA.Imp.Enc ($\approx 9 \times$ Parse $+ 7 \times$ CBD) the sampling requires only about 10 percent of the overall runtime. Additionally, in Table 5.3 we have computed the sum of cycles based on the calls to measured subfunctions for KS1. This gives an overview what amount of cycles can be associated to each operation. In all three functions the most cycles are contributed by MulAddSingle and Sneeze. They would be a natural target for further optimization.

Compared to a Kyber768 implementation that is using the NTT as specified in [SAB+17] on the SLE 78 in software, our approach of using the coprocessor to compute the KyberMulAdd gadget provides an advantage. On the SLE 78 a single $n = 256$ NTT costs 997,691 cycles. The computation of KyberCPA.Enc for $k = 3$ requires 10 calls to the NTT[10] which alone would account for roughly $10 \cdot 997{,}691 \approx 10.0$ million cycles plus additional overhead from pointwise multiplication and addition.

In case one would want to make our implementation compatible with Kyber as specified in [SAB+17] in terms of NTT usage and still use the KyberMu-

[10] See [SAB+17, Algorithm 5] where 3 NTTs are required to transform $\mathbf{r}$, 3 inverse NTTs are applied to $\hat{\mathbf{A}} \circ \hat{\mathbf{r}}$, 3 inverse NTTs are needed to transform $\mathbf{t}$ and 1 inverse NTT is then needed to obtain v.

lAdd gadget, they would have to perform k^2 inverse NTTs and then use our multiplication algorithm. This would add roughly $3^2 \cdot 997{,}691 \approx 9.0$ million cycles to Gen and Enc when executed on the CPU. It would basically nullify all gains from a different and faster algorithm for polynomial multiplication.

All in all, when our Kyber variant that is using the AES coprocessor (i.e. AES-HW) is run on our target device with an average clock frequency of 50 MHz we can execute Kyber.CPA.Imp.Gen in 72.5 ms, Kyber.CPA.Imp.Enc in 94.9 and Kyber.CPA.Imp.Dec in 28.4 ms.

For the CCA variant the decryption becomes slower due to the re-encryption but the additional overhead of the hash functions H and G is rather low when the SHA-256 coprocessor is used (HW-SHA-256) to compute SHA-256 and HKDF with HMAC-SHA-256. When H and G are instantiated with SHA3 implemented in software (SW-SHA3) a significant portion of the computation is now attributed to SHA3. In comparison we can execute Kyber.CCA.Imp.Gen in 79.6 ms (2,903 ms with SW-SHA3), Kyber.CCA.Imp.Enc in 102.4 ms (571.2 ms with SW-SHA3) and Kyber.CCA.Imp.Dec in 132.7 ms (394.0 ms with SW-SHA3). An implementation of Kyber that is fully compatible with the specification [SAB+17] would not achieve practical performance mainly due to the slow SHA3 PRNG performance and to a lesser extent due to the slower NTT in software. Of course, further low-level optimization of SHA3 and the NTT could change this picture to some extent.

5.6.2 Comparison with related work

In Table 5.4 we provide a comparison of our results with related work on similar target platforms available at the time of publication of this chapter as [AHH+18]. However, it should be noted that such a comparison will always lack precision as many parameters of published implementations differ in terms of cryptographic (post-quantum) bit-security level, implementation security level, exact variant of a scheme, CPU architecture, maximum clock frequency of the device, or availability of specific accelerators. Moreover, only limited information is available about most smart card platforms and those platforms

Operation	Cycles
Snort (KS1)	31,017
Sneeze (KS1)	295,730
MulAddSingle (KS1)	201,767
FinalEll (KS1)	28,381
Snort (KS2)	70,015
Sneeze (KS2)	295,331
MulAddSingle (KS2)	186,652
FinalEll (KS2)	90,728
NTT ($n = 256$, in SW)	997,691
Pointwise-Multiplication ($n = 256$, in SW)	356,549
CBD(PRF(σ, N)) (Software-SHA3)	9,341,406
CBD(PRF(σ, N)) (Hardware-AES)	31,068
Parse(XOF($\rho\|\|i\|\|j$)) (Software-SHA3)	19,934,170
Parse(XOF($\rho\|\|i\|\|j$)) (Hardware-AES)	21,081
Kyber.CPA.Imp.Gen (HW-AES: PRF/XOF; KS1)	3,953,224
Kyber.CPA.Imp.Enc (HW-AES: PRF/XOF; KS1)	5,385,598
Kyber.CPA.Imp.Dec (KS1)	1,382,963
Kyber.CPA.Imp.Gen (HW-AES: PRF/XOF; KS2)	3,625,718
Kyber.CPA.Imp.Enc (HW-AES: PRF/XOF; KS2)	4,747,291
Kyber.CPA.Imp.Dec (KS2)	1,420,367
Kyber.CCA.Imp.Gen (HW-AES: PRF/XOF; HW-SHA-256: H; KS2)	3,980,517
Kyber.CCA.Imp.Enc (HW-AES: PRF/XOF; HW-SHA-256: G, H; KS2)	5,117,996
Kyber.CCA.Imp.Dec (HW-AES: PRF/XOF; HW-SHA-256: G, H; KS2)	6,632,704
Kyber.CCA.Imp.Gen (HW-AES: PRF/XOF; SW-SHA3: H; KS2)	14,512,691
Kyber.CCA.Imp.Enc (HW-AES: PRF/XOF; SW-SHA3: G, H; KS2)	18,051,747
Kyber.CCA.Imp.Dec (HW-AES: PRF/XOF; SW-SHA3: G, H; KS2)	19,702,139

Table 5.2: Performance of our work on the SLE 78 target device in clock cycles.

are often not available without signing non-disclosure agreements. It is also clear that the requirements for a certified contactless high security controller, where most computations are done using coprocessors, are expected to lead to different CPU designs or low-level implementations than those for a high performance embedded microcontroller.

As we use an RSA coprocessor for lattice-based cryptography, a natural target for a comparison is RSA. The cycle counts given in Table 5.4 for coprocessor supported RSA on our SLE 78 target device are based on the data sheet. With an average clock frequency of 50 MHz, on the SLE 78 RSA encryption can be executed in 6 ms while RSA decryption with CRT needs 120 ms. In comparison with our work this shows that our Kyber implementation is one order of magnitude slower for encryption but performs decryption with similar speed. In case RSA is not used with CRT our Kyber decryption even

Kyber.CPA.Imp.Gen (KS1)			
Function	Calls	Cycles per function	Product
CBD(PRF(σ, N)) (HW-AES)	6	31,068	186,408
Parse(XOF($\rho\|\|i\|\|j$)) (HW-AES)	9	21,081	189,729
Snort	15	31,017	465,255
MulAddSingle	9	201,767	1,815,903
Sneeze	3	295,730	887,190
FinalEll	3	28,381	85,143
Encode/Decode	-	-	400,226
			= 4,029,854
Kyber.CPA.Imp.Enc (KS1)			
Function	Calls	Cycles per function	Product
CBD(PRF(σ, N)) (HW-AES)	7	31,068	217,476
Parse(XOF($\rho\|\|i\|\|j$)) (HW-AES)	9	21,081	189,729
Snort	19	31,017	589,515
MulAddSingle	12	201,767	2,421,204
Sneeze	4	295,730	1,182,920
FinalEll	4	28,381	113,524
Encode/Decode	-	-	676,453
			= 5,390,629
Kyber.CPA.Imp.Dec (KS1)			
Function	Calls	Cycles per function	Product
CBD(PRF(σ, N)) (HW-AES)	0	31,068	0
Parse(XOF($\rho\|\|i\|\|j$)) (HW-AES)	0	21,081	0
Snort	4	31,017	217,119
MulAddSingle	3	201,767	605,301
Sneeze	1	295,730	295,730
FinalEll	1	28,381	28,381
Encode/Decode	-		365,175
			= 1,511,706

Table 5.3: Called functions, number of calls, clock cycles, and final sum of clock cycles.

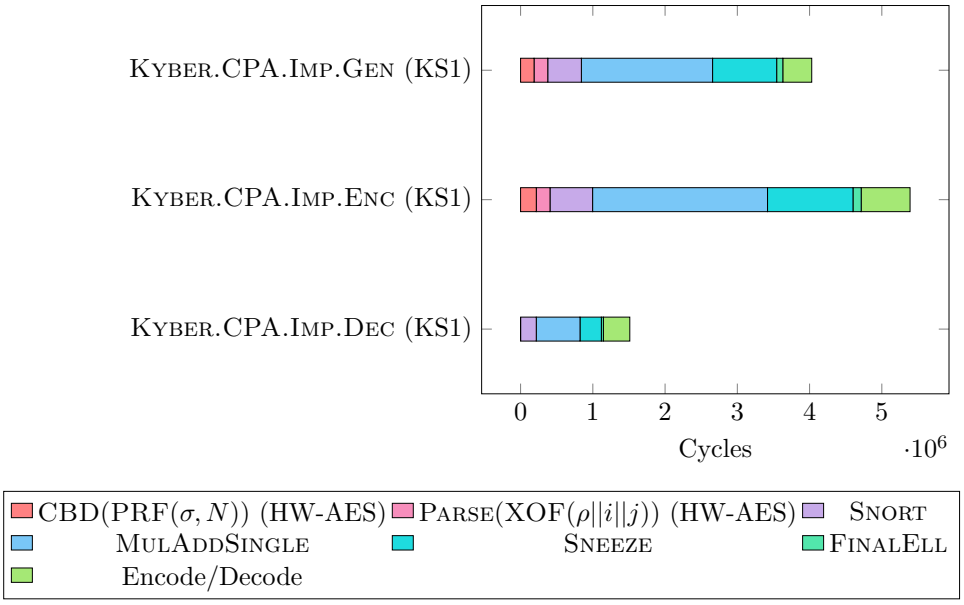


Figure 5.1: Total cycle counts per Kyber.CPA.Imp (KS1) function from Table 5.3.

outperforms RSA. However, it should be noted that the RSA cycle counts do not account for padding like Optimal Asymmetric Encryption Padding (OAEP) which is often used to achieve CCA2 security for RSA. However, they include countermeasures against physical attacks (e.g. exponent blinding or message blinding, see [FWA+13]) while our implementation does not.

Publicly available information on the performance of RSA and ECC on various smart cards running the JavaCard platform can be found in works like [DRHM17, SNS+16], the Bachelor's thesis of Kvašňovský [Kva16] as well as in the JCAlgTest project[11]. Across the selected cards, the runtime for an RSA2048 encryption function call is in the range from 8 to 74 ms while RSA2048 decryption takes between 426 to 2,927 ms and 140 to 1,569 ms when using the Chinese Remainder Theorem (CRT). On-card key generation for RSA2048 is a complex process [NSS+17] with a variable runtime due to the required primality testing and takes between 6,789 and 44,143 ms. There is also a certain overhead by the JavaCard platform compared to a pure native implementation as well as overhead from various countermeasures against physical attacks.

[11] See `https://www.fi.muni.cz/~xsvenda/jcalgtest/comparative-table.html`.

For comparison with other post-quantum schemes we have ported the reference implementation of ephemeral/CPA-secure NewHope with $n = 1024$ claiming 255-bits of post-quantum security onto our target device. To obtain a fair comparison we also changed the internal PRNG to use the coprocessor-based AES in counter-mode and we removed costly randomness hashing in the key generation. With these modifications the main bottleneck in NewHope is the computation of NTTs. When comparing CPA-secure NewHope implementation (claimed 255-bit security level) with our CPA-secure Kyber (claimed 161-bit security level) in an ephemeral key setting[12], we achieve a factor of 6 better performance for Alice (Gen+Dec) and a factor of 7 better performance for Bob (Enc). Note that the implementation of our variant of Kyber that is not using the NTT would most likely lead to a loss of performance on other platforms. However, the implementation of Saber on ARM given in [KBMSRV18] shows that high performance is also possible without using the NTT when parameters are chosen accordingly.

Most modern general purpose ARM-based microcontroller platforms (e.g. Cortex-M) have the advantage of a 32-bit architecture and are equipped with a single-cycle or few-cycles multiplier (optional in Cortex-M0). Thus good performance can be expected for most arithmetic operations, e.g. the inner loop of the NTT. Open-source implementations of Kyber768 and NewHope1024 targeting general purpose ARM controllers are available through the *mupq* project [va18]. It can be seen that in comparison with such a different class of devices our CCA-secure Kyber768 implementation of Gen and Enc is slower than CCA-secure Kyber768 on ARM using the NTT.

5.7 Conclusions

In this chapter we have shown that fast post-quantum cryptography is feasible on current smart card platforms. On a commercially available device it is possible to obtain a significant speed-up of the arithmetic of lattice-based cryptography by reusing already existing coprocessors dedicated to the accel-

[12] Of course, a better target for comparison would be Kyber1024 with 218-bit security but an implementation on SLE 78 is not available as we focused on Kyber768.

Table 5.4: Comparison of our work with other PKE or KEM schemes on various microcontroller platforms in clock cycles.

Scheme	Target	Gen	Enc	Dec
Kyber768[a] (CPA; our work)	SLE 78	3,625,718	4,747,291	1,420,367
Kyber768[b] (CCA; our work)	SLE 78	3,980,517	5,117,996	6,632,704
RSA-2048[c]	SLE 78	-	≈ 300,000	≈ 21,200,000
RSA-2048 (CRT)[d]	SLE 78	-	≈ 300,000	≈ 6,000,000
Kyber768 (CPA+NTT)[e]	SLE 78	≈ 10,000,000	≈ 14,600,000	≈ 5,400,000
NewHope1024[f]	SLE 78	≈ 14,700,000	≈ 31,800,000	≈ 15,200,000
Kyber768[g]	ARM	1,200,351	1,497,789	1,526,564
NewHope-1024[h]	ARM	1,168,224	1,738,922	298,877
CPA-RLWE-512[i]	AVR	-	1,975,806	553,536
CCA-RLWE-1024[j]	ARM	2,669,559	4,176,68	4,416,918
Saber[k]	ARM	1,147,000	1,444,000	1,543,000
QC-MDPC[l]	ARM	-	7,018,493	42,129,589
Curve25519[m]	MSP	5,941,784	11,883,568	5,941,784
Curve25519[n]	ARM	3,589,850	7,179,700	3,589,850

[a] CPA-secure Kyber variant using the AES coprocessor to implement PRF/XOF and KS2 on SLE 78 @ 50 MHz.

[b] CCA-secure Kyber variant using the AES coprocessor to implement PRF/XOF, the SHA-256 coprocessor to implement G and H and KS2 on SLE 78 @ 50 MHz.

[c] RSA-2048 encryption with short exponent and decryption without CRT and with countermeasures on SLE 78 @ 50 MHz. Extrapoliation based on data-sheet.

[d] RSA-2048 decryption with short exponent and decryption with CRT and countermeasures on SLE 78 @ 50 MHz. Extrapoliation based on data-sheet.

[e] Extrapolation of cycle counts of CPA-secure Kyber768 based on our implementation assuming usage of the AES coprocessor to implement PRF/XOF and a software implementation of the NTT with 997,691 cycles for an NTT on SLE 78 @ 50 MHz.

[f] Reference implementation of constant time ephemeral NewHope key exchange ($n = 1024$) [ADPS16] modified to use the AES coprocessor as PRNG on SLE 78 @ 50 MHz.

[g] Kyber768 from *mupq* project [va18] on ARM Cortex-M4F (STM32).

[h] Constant time ephemeral NewHope key exchange (n=1024) [ADPS16] from [AJS16] on ARM Cortex-M0 (STM32) @ 48 MHz.

[i] Constant time CPA-secure RLWE-encryption [LP11] (RLWEenc-IIa with $n = 512$) from [LPO+17] on 8-bit ATxmega128A1 @ 32 MHz.

[j] CCA-secure RLWE-encryption [LP11] ($n = 1024$) from [OSPG18] on ARM Cortex-M4F (STM32) @ 168 MHz. With first order masking decryption is 25,334,493 cycles.

[k] Saber [DKRV17] from [KBMSRV18] on ARM Cortex-M4F (STM32F4) @ 168 MHz. Parameters provide 180-bit of quantum-security.

[l] CPA-secure QC-MPDC public-key encryption [MTSB13] from [vMOG15] on ARM Cortex-M4F (STM32F407) @ 168 MHz. Parameters provide 80-bit pre-quantum security level.

[m] Elliptic curve Diffie-Hellman using Curve25519 [Ber06] from [DHH+15] on 16-bit MSP430X @ 16 MHz. For simplification we report the cost of one point multiplication (PM) in Gen, two PMs in Enc and one PM in Dec.

[n] Elliptic curve Diffie-Hellman using Curve25519 [Ber06] from [DHH+15] on ARM Cortex @ 48 MHz. Reporting as in [l].

eration of RSA or ECC. Our work can thus be used by the industry for a possibly smoother migration towards PQC, by reusing already existing and available hardware. Our work also shows that the NTT might not always be the superior polynomial multiplication algorithm.[13] This seems to be a worthwhile consideration in the context of the NIST standardisation process where some schemes made the NTT part of their definition. Moreover, our results show that the performance of lattice-based schemes on particular embedded devices highly depends on the speed of the underlying PRNG. It might be worthwhile to consider constructions that make use of PRNGs based on AES instead of SHA3 due to the better availability of (secured) AES hardware acceleration on smart cards or constrained devices in general. The same argument applies to the instantiation of hash functions using SHA-256.

With regard to the optimisation of our particular Kyber implementation, a possible next step is an implementation on an ARM-based smart card or embedded secure element equipped with an ECC/RSA coprocessor. On such an architecture the comparison to standard microcontroller-based implementations of PQC (e.g. [vMOG15, DHH+15, OSPG18]) would be much easier. Additionally, it is an open question how much speed-up ECC/RSA coprocessors will actually provide on ARM platforms equipped with a single-cycle multiplier. Here it is also worth to consider that on an ARM processor Snort, Sneeze, and software-based big integer addition are also expected to be significantly faster due to the more efficient instruction set and larger word size, while the CPU and the coprocessor could still execute in parallel.

From the algorithmic side, in the case of the KS1 $\omega = 64$ implementation of Kyber we currently require $\ell \geq 25$ bits of precision, and hence opted for using 32 bits. By using the considerations made in Section 5.4 about swapping ω for n in the formula for computing ℓ, we could get down to $\ell \geq 23$, making it possible to save some memory at the cost of a more complex unpacking ($\ell = 24$ would be of particular interest, being byte-aligned).

[13] See also an NTT-related discussion on the NIST PQC mailing list: `https://groups.google.com/a/list.nist.gov/forum/#!topic/pqc-forum/r9R7OJT6x_c`.

In a more general direction it appears interesting to investigate whether a performance advantage can be obtained with schemes specifically designed with the constraints of the big integer multiplier in mind, such as ThreeBears [Ham17] or Mersenne-75683917 [AJPS17]. However, we note that these schemes use integer sizes too large for direct handling with our coprocessor. In contrast, MLWE-based schemes immediately allow for a piece-wise approach. Thus, another interesting target for implementation could be an MLWE-based scheme that is parametrised with a power-of-two modulus q, e.g. SABER [DKRV17], which permits to efficiently implement the strategy from (5.1). For example, a viable choice could be a prime-cyclotomic ring for $n = 167 - 1 = 2 \cdot 83$ with $q = 2^{13}$ such that each ring element fits directly into a coprocessor register. Another approach would be a Kyber instantiation with a smaller prime modulus q, as we do not have to choose q in a way that a fast NTT exists. Moreover, our results naturally transfer over to the Dilithium signature scheme [LDK+17] and an implementation on the SLE 78 is a natural next step. However, parameters have to be adapted for Dilithium, as it uses a larger modulus $q = 8380417$. Another interesting question is whether it is possible to efficiently use RSA/ECC coprocessors to implement the NTT by treating the big integer multiplier as a vector processor using smart packing of coefficients or a variant of Kronecker substitution.

5.7.1 Developments since publication

Since publication of this chapter [AHH+18], three papers extending this line of research appeared. In [WGY20], the authors investigate implementations of Saber [DKRV17] on a ESP32[14] IoT microcontroller, using KS1/2 together with Karatsuba and Toom-Cook [Too63, CA69] multiplication and different "ring splitting" strategies. In [BRv20] the authors introduce "Kronecker+", a generalisation of the work of Harvey [Har09], and propose a rough analysis of the cost of evaluating Saber using their techniques. Finally, two other alternative techniques to KS1/2 were introduced in [GMR20], where the authors describe them and use them to implement some instances of Kyber, Saber

[14] `https://www.espressif.com/en/products/socs/esp32`

and LAC [LLZ+18] on an unspecified[15] platform with a $\approx$ 2048 bit RSA coprocessor.

Furthermore, the second round specification of Kyber [SAB+19] and the third round specification of Saber [DKR+20] added "90s" variants that replace newer symmetric primitives with less hardware support such as SHA3 with older primitives from the AES and SHA2 families, as we suggested in our conclusions [AHH+18].

[15] The authors write that this is due to intellectual property reasons.

Bibliography

[AAA+19] Héctor Abraham, Rochisha Agarwal, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Matthew Amy, Eli Arbel, Arijit02, Abraham Asfaw, Artur Avkhadiev, Carlos Azaustre, Abhik Banerjee, Aman Bansal, Panagiotis Barkoutsos, Ashish Barnawal, George Barron, and George S. Barron *et al.* Qiskit: An open-source framework for quantum computing, 2019.

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, and William *et al.* Courtney. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, Oct 2019.

[AB19] Martin Albrecht and Gregory Bard. *The M4RI Library.* The M4RI Team, 2019. `https://bitbucket.org/malb/m4ri`.

[ABB+17] Erdem Alkim, Nina Bindel, Johannes A. Buchmann, Özgür Dagdelen, Edward Eaton, Gus Gutoski, Juliane Krämer, and Filip Pawlega. Revisiting TESLA in the quantum random oracle model. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 143–162. Springer, Heidelberg, 2017.

[ABF+20] Martin R. Albrecht, Shi Bai, Pierre-Alain Fouque, Paul Kirchner, Damien Stehlé, and Weiqiang Wen. Faster enumeration-based

lattice reduction: Root hermite factor $k^{1/(2k)}$ time $k^{k/8+o(k)}$. In Micciancio and Ristenpart [MR20], pages 186–212.

[ACC+18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[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 Catalano and De Prisco [CD18], pages 351–367.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Halevi [Hal09], pages 595–618.

[AD17] Martin R. Albrecht and Amit Deo. Large modulus ring-LWE $\geq$ module-LWE. In Takagi and Peyrin [TP17], pages 267–296.

[ADH+19] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Ishai and Rijmen [IR19], pages 717–746.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[AFG14] Martin R. Albrecht, Robert Fitzpatrick, and Florian Göpfert. On the efficacy of solving LWE by reduction to unique-SVP. In Hyang-Sook Lee and Dong-Guk Han, editors, *ICISC 13*, volume 8565 of *LNCS*, pages 293–310. Springer, Heidelberg, November 2014.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part I*, volume 6755 of *LNCS*, pages 403–415. Springer, Heidelberg, July 2011.

[AGPS20] Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 583–613, Cham, 2020. Springer International Publishing.

[AGVW17] Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In Takagi and Peyrin [TP17], pages 297–322.

[AHH+18] Martin R. Albrecht, Christian Hanser, Andrea Hoeller, Thomas Pöppelmann, Fernando Virdia, and Andreas Wallner. Implementing RLWE-based schemes using an RSA co-processor. *IACR TCHES*, 2019(1):169–208, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/7338`.

[AJPS17] Divesh Aggarwal, Antoine Joux, Anupam Prakash, and Mikos Santha. Mersenne-756839. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[AJS16] Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. Newhope on ARM cortex-m. In *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016*, pages 332–349, 2016.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *33rd ACM STOC*, pages 601–610. ACM Press, July 2001.

[Alb17] Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In

Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 103–129. Springer, Heidelberg, April / May 2017.

[AM09] Divesh Aggarwal and Ueli Maurer. Breaking RSA generically is equivalent to factoring. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 36–53. Springer, Heidelberg, April 2009.

[AMG+16] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John M. Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 317–337. Springer, Heidelberg, August 2016.

[AMM20a] Ravi Anand, Arpita Maitra, and Sourav Mukhopadhyay. Evaluation of quantum cryptanalysis on SPECK. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *Progress in Cryptology – INDOCRYPT 2020*, pages 395–413, Cham, 2020. Springer International Publishing.

[AMM20b] Ravi Anand, Arpita Maitra, and Sourav Mukhopadhyay. Grover on SIMON. *Quantum Information Processing*, 19(9):340, Sep 2020.

[AMM+20c] Ravi Anand, Subhamoy Maitra, Arpita Maitra, Chandra Sekhar Mukherjee, and Sourav Mukhopadhyay. Resource estimation of grovers-kind quantum cryptanalysis against FSR based symmetric ciphers. Cryptology ePrint Archive, Report 2020/1438, 2020. `https://eprint.iacr.org/2020/1438`.

[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.

[ARS13] Brittanney Amento, Martin Rötteler, and Rainer Steinwandt. Efficient quantum circuits for binary elliptic curve arith-

metic: Reducing T-Gate complexity. *Quantum Info. Comput.*, 13(7–8):631–644, July 2013.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Oswald and Fischlin [OF15], pages 430–454.

[ARS+16] Martin Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. Cryptology ePrint Archive, Report 2016/687, 2016. `http://eprint.iacr.org/2016/687`.

[ASAM18] Mishal Almazrooie, Azman Samsudin, Rosni Abdullah, and Kussay N. Mutter. Quantum reversible circuit of AES-128. *Quantum Information Processing*, 17(5):112, Mar 2018.

[AWHT16] Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In Fischlin and Coron [FC16], pages 789–819.

[Bab86] László Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, Mar 1986.

[BB17] Gustavo Banegas and Daniel J. Bernstein. Low-communication parallel quantum multi-target preimage search. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017*, volume 10719 of *LNCS*, pages 325–335. Springer, Heidelberg, August 2017.

[BBB+09] Daniel V. Bailey, Lejla Batina, Daniel J. Bernstein, Peter Birkner, Joppe W. Bos, Hsieh-Chung Chen, Chen-Mou Cheng, Gauthier van Damme, Giacomo de Meulenaer, Luis Julian Dominguez Perez, Junfeng Fan, Tim Güneysu, Frank Gurkaynak, Thorsten Kleinjung, Tanja Lange, Nele Mentens, Ruben Niederhagen, Christof Paar, Francesco Regazzoni, Peter Schwabe, Leif Uhsadel, Anthony Van Herrewege, and Bo-Yin Yang. Breaking ECC2K-130. Cryptology ePrint Archive, Report 2009/541, 2009. `http://eprint.iacr.org/2009/541`.

[BBE+18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 354–384. Springer, Heidelberg, April / May 2018.

[BBG+13] Robert Beals, Stephen Brierley, Oliver Gray, Aram W. Harrow, Samuel Kutin, Noah Linden, Dan Shepherd, and Mark Stather. Efficient distributed quantum computing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 469(2153):20120686, 2013.

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

[BCD+16] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1006–1018. ACM Press, October 2016.

[BCIV17] Joppe W. Bos, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Privacy-friendly forecasting for the smart grid using homomorphic encryption and the group method of data handling. In Marc Joye and Abderrahmane Nitaj, editors, *Progress in Cryptology - AFRICACRYPT 2017, Proceedings*, pages 184–201. Springer International Publishing, 2017.

[BCLv19] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[BDEZ12] Razvan Barbulescu, Jérémie Detrey, Nicolas Estibals, and Paul Zimmermann. Finding optimal formulae for bilinear maps. In

Ferruh Özbudak and Francisco Rodríguez-Henríquez, editors, *Arithmetic of Finite Fields*, volume 7369 of *Lecture Notes in Computer Science*, pages 168–186. Springer Berlin Heidelberg, 2012.

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

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 353–367. IEEE, 2018.

[Ben80] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, 22(5):563–591, May 1980.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006.

[BFI19] Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. More results on shortest linear programs. In Nuttapong Attrapadung and Takeshi Yagi, editors, *IWSEC 19*, volume 11689 of *LNCS*, pages 109–128. Springer, Heidelberg, August 2019.

[BG14a] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 28–47. Springer, Heidelberg, February 2014.

[BG14b] Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In Willy Susilo and Yi Mu, editors, *ACISP 14*,

volume 8544 of *LNCS*, pages 322–337. Springer, Heidelberg, July 2014.

[BGB+18] Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear t complexity. *Phys. Rev. X*, 8:041015, Oct 2018.

[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 Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

[BGG+20] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment. In Micciancio and Ristenpart [MR20], pages 62–91.

[BGH13] Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in LWE-based homomorphic encryption. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 1–13. Springer, Heidelberg, February / March 2013.

[BGPW16] Johannes A. 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 David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 24–43. Springer, Heidelberg, April 2016.

[BJ20] Xavier Bonnetain and Samuel Jaques. Quantum period finding against symmetric primitives in practice. Cryptology ePrint Archive, Report 2020/1418, 2020. `https://eprint.iacr.org/2020/1418`.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd ACM STOC*, pages 435–440. ACM Press, May 2000.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Boneh et al. [BRF13], pages 575–584.

[BM19] Alexandra Boldyreva and Daniele Micciancio, editors. *CRYPTO 2019, Part I*, volume 11692 of *LNCS*. Springer, Heidelberg, August 2019.

[BMW19] Shi Bai, Shaun Miller, and Weiqiang Wen. A refined analysis of the cost for solving LWE via uSVP. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, *AFRICACRYPT 19*, volume 11627 of *LNCS*, pages 181–205. Springer, Heidelberg, July 2019.

[BNS19] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. Quantum security analysis of AES. *IACR Trans. Symmetric Cryptol.*, 2019(2):55–93, 2019.

[BP10] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In *SEA 2010*, pages 178–189. Springer, 2010.

[BP12] Joan Boyar and René Peralta. A small depth-16 circuit for the AES S-Box. In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *SEC 2012*. Springer, 2012.

[BP+19] Joan Boyar, René Peralta, et al. Small low-depth circuits for cryptographic applications. *Cryptography and Communications*, 11(1):109–127, 2019.

[BRF13] Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors. *45th ACM STOC*. ACM Press, June 2013.

[Bro16] Daniel R. L. Brown. Breaking RSA may be as difficult as factoring. *Journal of Cryptology*, 29(1):220–241, January 2016.

[BRv20] Joppe W. Bos, Joost Renes, and Christine van Vredendaal. Polynomial multiplication with contemporary co-processors: Beyond kronecker, Schönhage-strassen & nussbaumer. Cryptology ePrint Archive, Report 2020/1303, 2020. `https://eprint.iacr.org/2020/1303`.

[BSJ15] Ahmad Boorghany, Siavash Bayat Sarmadi, and Rasool Jalili. On constrained implementation of lattice-based cryptographic primitives and schemes on smart cards. *ACM Trans. Embed. Comput. Syst.*, 14(3):42:1–42:25, April 2015.

[BSW18] Shi Bai, Damien Stehlé, and Weiqiang Wen. Measuring, simulating and exploiting the head concavity phenomenon in BKZ. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 369–404. Springer, Heidelberg, December 2018.

[BV98] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 59–71. Springer, Heidelberg, May / June 1998.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

[BVWW16] Zvika Brakerski, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Obfuscating conjunctions under entropic ring LWE. In Madhu Sudan, editor, *ITCS 2016*, pages 147–156. ACM, January 2016.

[CA69] Stephen A Cook and Stål O Aanderaa. On the minimum computation time of functions. *Transactions of the American Mathematical Society*, 142:291–314, 1969.

[CCLS20] Hao Chen, Lynn Chua, Kristin Lauter, and Yongsoo Song. On the concrete security of LWE with small secret. Cryptology

ePrint Archive, Report 2020/539, 2020. https://eprint.iacr.org/2020/539.

[CD18] Dario Catalano and Roberto De Prisco, editors. *SCN 18*, volume 11035 of *LNCS*. Springer, Heidelberg, September 2018.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Thuraisingham et al. [TEMX17], pages 1825–1842.

[CDW17] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short stickelberger class relations and application to ideal-SVP. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 324–348. Springer, Heidelberg, April / May 2017.

[CG13] Ran Canetti and Juan A. Garay, editors. *CRYPTO 2013, Part I*, volume 8042 of *LNCS*. Springer, Heidelberg, August 2013.

[Che13] Yuanmi Chen. *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD thesis, Paris 7, 2013. Available at https://archive.org/details/PhDChen13.

[CHK+17] Jung Hee Cheon, Kyoohyung Han, Jinsu Kim, Changmin Lee, and Yongha Son. A practical post-quantum public-key cryptosystem based on spLWE. In Seokhie Hong and Jong Hwan Park, editors, *ICISC 16*, volume 10157 of *LNCS*, pages 51–74. Springer, Heidelberg, November / December 2017.

[CHT12] Peter Czypek, Stefan Heyse, and Enrico Thomae. Efficient implementations of MQPKS on constrained devices. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 374–389. Springer, Heidelberg, September 2012.

[CKLS16] Jung Hee Cheon, Duhyeong Kim, Joohee Lee, and Yongsoo Song. Lizard: Cut off the tail! Practical post-quantum public-

key encryption from LWE and LWR. Cryptology ePrint Archive, Report 2016/1126 (20161222:071525), 2016. `http://eprint.iacr.org/2016/1126/20161222:071525`.

[CKLS18] Jung Hee Cheon, Duhyeong Kim, Joohee Lee, and Yongsoo Song. Lizard: Cut off the tail! A practical post-quantum public-key encryption from LWE and LWR. In Catalano and De Prisco [CD18], pages 160–177.

[CLLc20] Doyoung Chung, Jooyoung Lee, Seungkwang Lee, and Dooho choi. Towards optimizing quantum implementation of AES S-box. Cryptology ePrint Archive, Report 2020/941, 2020. `https://eprint.iacr.org/2020/941`.

[CLP17] Hao Chen, Kim Laine, and Rachel Player. Simple encrypted arithmetic library - SEAL v2.1. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *FC 2017 Workshops*, volume 10323 of *LNCS*, pages 3–18. Springer, Heidelberg, April 2017.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Canetti and Garay [CG13], pages 476–493.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2011.

[CÖ10] Murat Cenk and Ferruh Özbudak. On multiplication in finite fields. *Journal of Complexity*, 26(2):172–186, 2010.

[CS15] Jung Hee Cheon and Damien Stehlé. Fully homomophic encryption over the integers revisited. In Oswald and Fischlin [OF15], pages 513–536.

[CS20] Amit Kumar Chauhan and Somitra Kumar Sanadhya. Quantum resource estimates of grover's key search on aria. In Lejla Batina, Stjepan Picek, and Mainack Mondal, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 238–258, Cham, 2020. Springer International Publishing.

[dCRVV15] Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Efficient software implementation of ring-LWE encryption. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015*, pages 339–344, 2015.

[DD18] Léo Ducas Daniel Dadush. Lecture notes on lattice algorithms and cryptography, February 2018. `https://homepages.cwi.nl/~dadush/teaching/lattices-2018/notes/lecture-4.pdf`.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Micciancio and Ristenpart [MR20], pages 329–358.

[DE05] S. Crocker D. Eastlake, J. Schiller. Randomness requirements for security. RFC 4086, RFC Editor, June 2005.

[den90] Bert den Boer. Diffie-Hellman is as strong as discrete log for certain primes (rump session). In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 530–539. Springer, Heidelberg, August 1990.

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

[DHH+15] Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, and Peter Schwabe. High-speed curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. *Des. Codes Cryptography*, 77(2–3):493–514, December 2015.

[Die82] D. Dieks. Communication by EPR devices. *Physics Letters A*, 92(6):271 – 272, 1982.

[DKP+19] Itai Dinur, Daniel Kales, Angela Promitzer, Sebastian Ramacher, and Christian Rechberger. Linear equivalence of block ciphers with partial non-linear layers: Application to LowMC. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 343–372. Springer, Heidelberg, May 2019.

[DKR+20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

[DKRV17] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. SABER. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[DKRV19] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. SABER. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[DKTW20] Jintai Ding, Seungki Kim, Tsuyoshi Takagi, and Yuntao Wang. Lll and stochastic sandpile models, 2020. `https://arxiv.org/abs/1804.03285`.

[DP19] James H Davenport and Benjamin Pring. Improvements to low-qubit quantum resource estimates for quantum search. In *The Eleventh International Workshop on Coding and Cryptography – WCC 2019*, 2019.

[DP21] James H Davenport and Benjamin Pring. Improvements to quantum search techniques for block-ciphers, with applications to AES. In *Selected Areas in Cryptography – SAC 2020*, 2021.

[DPW19] Léo Ducas, Maxime Plançon, and Benjamin Wesolowski. On the shortness of vectors to be found by the ideal-SVP quantum algorithm. In Boldyreva and Micciancio [BM19], pages 322–351.

[DR99] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. Technical report, National Institute of Standards and Technology, 1999.

[DR01] Joan Daemen and Vincent Rijmen. Specification for the advanced encryption standard (AES). *Federal Information Processing Standards Publication*, 197, 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[DRHM17] Petr Dzurenda, Sara Ricci, Jan Hajny, and Lukas Malina. Performance analysis and comparison of different elliptic curves on smart cards. In *International Conference on Privacy, Security and Trust (PST)*, 2017. to appear, see `https://www.ucalgary.ca/pst2017/files/pst2017/paper-39.pdf`.

[DT17] Fplll Development Team. fplll, a lattice reduction library. Available at `https://github.com/fplll/fplll`, 2017.

[EJMY19] Patrik Ekdahl, Thomas Johansson, Alexander Maximov, and Jing Yang. A new SNOW stream cipher called SNOW-V. *IACR Trans. Symm. Cryptol.*, 2019(3):1–42, 2019.

[FC16] Marc Fischlin and Jean-Sébastien Coron, editors. *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*. Springer, Heidelberg, May 2016.

[Fel68] William Feller. *An Introduction to Probability Theory and Its Applications, Vol. 1, 3rd Edition*. Wiley, oct 1968.

[Fey82] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, Jun 1982.

[Fey86] Richard P. Feynman. Quantum mechanical computers. *Foundations of Physics*, 16(6):507–531, Jun 1986.

[FH07] Haining Fan and M. Anwar Hasan. Comments on "five, six, and seven-term karatsuba-like formulae". *IEEE Trans. Computers*, 56(5):716–717, 2007.

[FHL+07] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. Mpfr: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 2007.

[FMMC12] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, Sep 2012.

[FPY17] The FPYLLL development team. fyplll, a Python (2 and 3) wrapper for fplll. Available at `https://github.com/fplll/fpylll`, 2017.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `http://eprint.iacr.org/2012/144`.

[FWA+13] Dirk Feldhusen, Guntram Wicke, Arnold Abromeit, Lex Schoonen, and Zertifizierungsstelle BSI˙Minimum requirements for evaluating side-channel attack resistance of rsa, dsa and diffie-hellman key exchange implementations. Technical report, German Federal Office for Information Security - BSI, 1 2013. See `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_46_BSI_guidelines_SCA_RSA_V1_0_e_pdf.pdf?__blob=publicationFile&v=1`.

[Gal12] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and

Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, Heidelberg, August 2012.

[Gib19] Elizabeth Gibney. Quantum gold rush: the private funding pouring into quantum start-ups. *Nature*, 574(7776):22–24, October 2019.

[Gid19] Craig Gidney. Windowed quantum arithmetic. *arXiv preprint arXiv:1905.07682*, 2019.

[Gil10] Henri Gilbert, editor. *EUROCRYPT 2010*, volume 6110 of *LNCS*. Springer, Heidelberg, May / June 2010.

[GJS15] Qian Guo, Thomas Johansson, and Paul Stankovski. Coded-BKW: Solving LWE using lattice codes. In Gennaro and Robshaw [GR15], pages 23–42.

[GK15] Peter Günther and Volker Krummel. Implementing cryptographic pairings on accumulator based smart card architectures. In *Mathematical Aspects of Computer and Information Sciences - 6th International Conference, MACIS 2015, Berlin, Germany, November 11-13, 2015, Revised Selected Papers*, pages 151–165, 2015.

[GKMR14] David Gosset, Vadym Kliuchnikov, Michele Mosca, and Vincent Russo. An algorithm for the T-Count. *Quantum Info. Comput.*, 14(15–16):1261–1276, November 2014.

[GLRS16] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying grover's algorithm to AES: Quantum resource estimates. In Takagi [Tak16], pages 29–43.

[GMR20] Aurélien Greuet, Simon Montoya, and Guénaël Renault. Speeding-up ideal lattice-based key exchange using a RSA/ECC coprocessor. Cryptology ePrint Archive, Report 2020/1602, 2020. `https://eprint.iacr.org/2020/1602`.

[GN08a] Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In Ladner and Dwork [LD08], pages 207–216.

[GN08b] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 31–51. Springer, Heidelberg, April 2008.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Gilbert [Gil10], pages 257–278.

[Gor93] Daniel M. Gordon. Discrete logarithms in $\mathbb{F}_p$ using the number field sieve. *SIAM Journal on Discrete Mathematics*, 6(1):124–138, 1993.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Ladner and Dwork [LD08], pages 197–206.

[GR04] Lov Grover and Terry Rudolph. How significant are the known collision and element distinctness quantum algorithms. *Quantum Info. Comput.*, 4(3):201–206, May 2004.

[GR15] Rosario Gennaro and Matthew J. B. Robshaw, editors. *CRYPTO 2015, Part I*, volume 9215 of *LNCS*. Springer, Heidelberg, August 2015.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th ACM STOC*, pages 212–219. ACM Press, May 1996.

[GS02] Antonio Valverde Garcia and Jean-Pierre Seifert. On the implementation of the Advanced Encryption Standard on a public-key crypto-coprocessor. In *Fifth Smart Card Research and Advanced Application Conference, CARDIS '02*, pages 135–146, 2002.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Canetti and Garay [CG13], pages 75–92.

[Gu19] Chunsheng Gu. Integer version of ring-lwe and its applications. In *International Symposium on Security and Privacy in Social Networks and Big Data*, pages 110–122. Springer, 2019.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Boneh et al. [BRF13], pages 545–554.

[Gö16] Florian Göpfert. *Securely Instantiating Cryptographic Schemes Based on the Learning with Errors Assumption.* PhD thesis, Technische Universität Darmstadt, 2016. `http://tuprints.ulb.tu-darmstadt.de/5850/`.

[Hal09] Shai Halevi, editor. *CRYPTO 2009*, volume 5677 of *LNCS*. Springer, Heidelberg, August 2009.

[Ham17] Mike Hamburg. Three Bears. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[Har09] David Harvey. Faster polynomial multiplication via multipoint kronecker substitution. *J. Symb. Comput.*, 44(10):1502–1510, 2009.

[HBB13] Andreas Hülsing, Christoph Busold, and Johannes Buchmann. Forward secure signatures on smart cards. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 66–80. Springer, Heidelberg, August 2013.

[Her50] C. Hermite. Extraits de lettres de m. ch. hermite à m. jacobi sur différents objects de la théorie des nombres. (continuation). *Journal für die reine und angewandte Mathematik*, 40:279–315, 1850.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Heidelberg, November 2017.

[HJN+20] Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. Improved quantum circuits for elliptic curve discrete logarithms. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 425–444. Springer, Heidelberg, 2020.

[HKM17] Gottfried Herold, Elena Kirshanova, and Alexander May. On the asymptotic complexity of solving lwe. *Designs, Codes and Cryptography*, Jan 2017.

[How07] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In Menezes [Men07], pages 150–169.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 447–464. Springer, Heidelberg, August 2011.

[HRS16] Andreas Hülsing, Joost Rijneveld, and Peter Schwabe. ARMed SPHINCS - computing a 41 KB signature in 16 KB of RAM. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 446–470. Springer, Heidelberg, March 2016.

[HS07] Guillaume Hanrot and Damien Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. In Menezes [Men07], pages 170–186.

[HVDH19] David Harvey and Joris Van Der Hoeven. Polynomial multiplication over finite fields in time $O(n \log n)$. working paper or preprint, March 2019.

[IR19] Yuval Ishai and Vincent Rijmen, editors. *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*. Springer, Heidelberg, May 2019.

[IT88] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in GF(2ˆm) using normal bases. *Inf. Comput.*, 78(3):171–177, 1988.

[JCK+20] Kyoungbae Jang, Seungju Choi, Hyeokdong Kwon, Hyunji Kim, Jaehoon Park, and Hwajeong Seo. Grover on Korean block ciphers. *Applied Sciences*, 10(18), 2020.

[JCKS20] Kyungbae Jang, Seungjoo Choi, Hyeokdong Kwon, and Hwajeong Seo. Grover on SPECK: Quantum resource estimates. Cryptology ePrint Archive, Report 2020/640, 2020. `https://eprint.iacr.org/2020/640`.

[JKL10] Yong-Sung Jeon, Young-Jin Kim, and Dong-Ho Lee. A compact memory-free architecture for the AES algorithm using resource sharing methods. *JCSC*, 2010.

[JMPS17] Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. Bit-sliding: A generic technique for bit-serial implementations of SPN-based primitives - applications to AES, PRESENT and SKINNY. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 687–707. Springer, Heidelberg, September 2017.

[JN03] Antoine Joux and Kim Nguyen. Separating decision Diffie-Hellman from computational Diffie-Hellman in cryptographic groups. *Journal of Cryptology*, 16(4):239–247, September 2003.

[JNRV20] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and LowMC. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 280–310. Springer, Heidelberg, May 2020.

[Jon13] Cody Jones. Low-overhead constructions for the fault-tolerant Toffoli gate. *Phys. Rev. A*, 87:022328, Feb 2013.

[JS19] Samuel Jaques and John M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In Boldyreva and Micciancio [BM19], pages 32–61.

[Kan87] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, Aug 1987.

[KBMSRV18] Angshuman Karmakar, Jose Maria Bermudo Mera, Sujoy Sinha Roy, and Ingrid Verbauwhede. Saber on ARM: CCA-secure module lattice-based key encapsulation on ARM. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):243–266, Aug. 2018.

[KF15] Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Gennaro and Robshaw [GR15], pages 43–62.

[KHJ18] Panjin Kim, Daewan Han, and Kyung Chul Jeong. Time—space complexity of quantum search algorithms in symmetric cryptanalysis: Applying to AES and SHA-2. *Quantum Information Processing*, 17(12):1–39, December 2018.

[KK51] J.F. Kenney and E.S. Keeping. *Mathematics of Statistics*. Van Nostrand, 1951.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[KLM+07] Phillip Kaye, Raymond Laflamme, Michele Mosca, et al. *An introduction to quantum computing*. Oxford University Press on Demand, 2007.

[KLSW17] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Shorter linear straight-line programs for MDS matrices. *IACR Trans. Symm. Cryptol.*, 2017(4):188–211, 2017.

[Knu76] Donald E Knuth. Big omicron and big omega and big theta. *ACM Sigact News*, 8(2):18–24, 1976.

[KO62] Anatolii A Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, volume 145, pages 293–294. Russian Academy of Sciences, 1962.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO 2010*,

volume 6223 of *LNCS*, pages 631–648. Springer, Heidelberg, August 2010.

[Kva16] Rudolf Kvašňovský. The detailed performance analysis of JavaCard cryptographic smartcards. Bachelor's thesis, Masaryk University, Faculty of Informatics, 2016.

[Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Gennaro and Robshaw [GR15], pages 3–22.

[Lan16] Adam Langley. CECPQ1 results, Nov 2016. `https://www.imperialviolet.org/2016/11/28/cecpq1.html`.

[LD08] Richard E. Ladner and Cynthia Dwork, editors. *40th ACM STOC*. ACM Press, May 2008.

[LDK+17] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[Lei80] Charles E. Leiserson. Area-efficient graph layouts. In *21st Annual Symposium on Foundations of Computer Science (SFCS 1980)*. IEEE, October 1980.

[Li11] S. Li. Concise formulas for the area and volume of a hyperspherical cap. *Asian Journal of Mathematics & Statistics*, 4(1):66–70, Jan 2011.

[LKS18] Guang Hao Low, Vadym Kliuchnikov, and Luke Schaeffer. Trading T-gates for dirty qubits in state preparation and unitary synthesis. *arXiv preprint arXiv:1812.00954*, 2018.

[LLL82] A.K. Lenstra, Jr. Lenstra, H.W., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LLMP93] A. K. Lenstra, H. W. Lenstra, M. S. Manasse, and J. M. Pollard. The number field sieve. In Arjen K. Lenstra and Hendrik W. Lenstra, editors, *The development of the number field sieve*, pages 11–42, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[LLZ+18] Xianhui Lu, Yamin Liu, Zhenfei Zhang, Dingding Jia, Haiyang Xue, Jingnan He, Bao Li, and Kunpeng Wang. LAC: Practical ring-LWE based public-key encryption with byte-level modulus. Cryptology ePrint Archive, Report 2018/1009, 2018. `https://eprint.iacr.org/2018/1009`.

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In Halevi [Hal09], pages 577–594.

[LN13] Mingjie Liu and Phong Q. Nguyen. Solving BDD by enumeration: An update. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 293–309. Springer, Heidelberg, February / March 2013.

[LN20] Jianwei Li and Phong Q. Nguyen. A complete analysis of the BKZ lattice reduction algorithm. Cryptology ePrint Archive, Report 2020/1237, 2020. `https://eprint.iacr.org/2020/1237`.

[LO83] J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. In *24th FOCS*, pages 1–10. IEEE Computer Society Press, November 1983.

[Low19] LowMC. LowMC/lowmc at e847fb160a, 2019. `https://github.com/LowMC/lowmc/tree/e847fb160a`.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, Heidelberg, February 2011.

[LPO+17] Zhe Liu, Thomas Pöppelmann, Tobias Oder, Hwajeong Seo, Sujoy Sinha Roy, Tim Güneysu, Johann Großschädl, Howon

Kim, and Ingrid Verbauwhede. High-performance ideal lattice-based cryptography on 8-bit AVR microcontrollers. *ACM Trans. Embedded Comput. Syst.*, 16(4):117:1–117:24, 2017.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Gilbert [Gil10], pages 1–23.

[LPS20] Brandon Langenberg, Hai Pham, and Rainer Steinwandt. Reducing the cost of implementing the Advanced Encryption Standard as a quantum circuit. *IEEE Transactions on Quantum Engineering*, 1:1–12, 2020.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, June 2015.

[LY20] Hui Liu and Li Yang. Quantum key recovery attack of SIMON 32/64. *arXiv preprint arXiv:2012.08321*, 2020.

[Mar03] Jacques Martinet. *Perfect Lattices in Euclidean Spaces*. Springer Berlin Heidelberg, 2003.

[Mau94] Ueli M. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete algorithms. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 271–281. Springer, Heidelberg, August 1994.

[Max19] Alexander Maximov. AES MixColumn with 92 XOR gates. Cryptology ePrint Archive, Report 2019/833, 2019. `https://eprint.iacr.org/2019/833`.

[Men07] Alfred Menezes, editor. *CRYPTO 2007*, volume 4622 of *LNCS*. Springer, Heidelberg, August 2007.

[Mic19a] Microsoft. Getting started with Python and Q# — Microsoft Docs, 2019. `https://docs.microsoft.com/en-us/quantum/install-guide/python`.

[Mic19b] Microsoft. microsoft/iqsharp: Microsoft's IQ# server., 2019. `https://github.com/microsoft/iqsharp`.

[MN18] Samuel K. Moore and Amy Nordrum. Intel's new path to quantum computing. *IEEE Spectrum*, 2018.

[Mon05] Peter L. Montgomery. Five, six, and seven-term karatsuba-like formulae. *IEEE Transactions on Computers*, 54(3):362–369, 2005.

[Mor44] Louis J Mordell. Observation on the minimum of a positive quadratic form in eight variables. *Journal of the London Mathematical Society*, 19(73_Part_1):3–6, 1944.

[MQT18] Microsoft Quantum Team. Developing a topological qubit. *Cloud Perspectives Blog*, 2018.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer, Heidelberg, Berlin, Heidelberg, New York, 2009.

[MR20] Daniele Micciancio and Thomas Ristenpart, editors. *CRYPTO 2020, Part II*, volume 12171 of *LNCS*. Springer, Heidelberg, August 2020.

[MSC+19] Giulia Meuli, Mathias Soeken, Earl Campbell, Martin Roetteler, and Giovanni De Micheli. The role of multiplicative complexity in compiling low t-count oracle circuits. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[MSR+19] Giulia Meuli, Mathias Soeken, Martin Roetteler, Nikolaj Bjorner, and Giovanni De Micheli. Reversible pebbling game for quantum memory management. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 288–291. IEEE, 2019.

[MTSB13] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo SLM Barreto. Mdpc-mceliece: New mceliece variants from moderate density parity-check codes. In *2013 IEEE international symposium on information theory*, pages 2069–2073. IEEE, 2013.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., 1st edition, 1996.

[MW99] Ueli M Maurer and Stefan Wolf. The relationship between breaking the diffie–hellman protocol and computing discrete logarithms. *SIAM Journal on Computing*, 28(5):1689–1721, 1999.

[MW16] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In Fischlin and Coron [FC16], pages 820–849.

[MY80] I Manin Yu. *Vychislimoe i nevychislimoe (Computable and Noncomputable).* Sov. Radio, Moscow, 1980.

[Nat16] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the Post-Quantum Cryptography standardization process. `http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf`, December 2016.

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, 2010.

[NNT+10] Yasuyuki Nogami, Kenta Nekado, Tetsumi Toyota, Naoto Hongo, and Yoshitaka Morikawa. Mixed bases for efficient inversion in $\mathbb{F}_{((2^2)^2)^2}$ and conversion matrices of subbytes of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 234–247. Springer, Heidelberg, August 2010.

[NS06] Phong Q. Nguyen and Damien Stehlé. Lll on the average. In Florian Hess, Sebastian Pauli, and Michael Pohst, editors, *Algorithmic Number Theory*, pages 238–256, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[NSS+17] Matús Nemec, Marek Sýs, Petr Svenda, Dusan Klinec, and Vashek Matyas. The return of coppersmith's attack: Practical factorization of widely used RSA moduli. In Thuraisingham et al. [TEMX17], pages 1631–1648.

[Nus80] H. Nussbaumer. Fast polynomial transform algorithms for digital convolution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(2):205–215, Apr 1980.

[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.

[NV10] Phong Q. Nguyen and Brigitte Vallée, editors. *The LLL Algorithm - Survey and Applications*. Information Security and Cryptography. Springer, 2010.

[OF15] Elisabeth Oswald and Marc Fischlin, editors. *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*. Springer, Heidelberg, April 2015.

[OPG14] Tobias Oder, Thomas Pöppelmann, and Tim Güneysu. Beyond ECDSA and RSA: lattice-based digital signatures on constrained devices. In *The 51st Annual Design Automation Conference 2014, DAC '14*, pages 110:1–110:6, 2014.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure masked Ring-LWE implementations. *IACR TCHES*, 2018(1):142–174, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/836`.

[PAA+17] Thomas Poppelmann, Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, and Douglas Stebila. NewHope. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzen-

macher, editor, *41st ACM STOC*, pages 333–342. ACM Press, May / June 2009.

[Pei16] Chris Peikert. How (not) to instantiate ring-LWE. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 411–430. Springer, Heidelberg, August / September 2016.

[PHS19] Alice Pellet-Mary, Guillaume Hanrot, and Damien Stehlé. Approx-SVP in ideal lattices with pre-processing. In Ishai and Rijmen [IR19], pages 685–716.

[Pol78] John M Pollard. Monte carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978.

[PS08] Xavier Pujol and Damien Stehlé. Rigorous and efficient short lattice vectors enumeration. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 390–405. Springer, Heidelberg, December 2008.

[PV21] Eamonn W. Postlethwaite and Fernando Virdia. On the success probability of solving unique SVP via BKZ. In *Public-Key Cryptography – PKC 2021*, Cham, 2021. Springer International Publishing.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[PyC19] PyCryptodome. Welcome to PyCryptodome's documentation — PyCryptodome 3.8.2 documentation, 2019. `https://pycryptodome.readthedocs.io/en/stable/index.html`.

[RdCR+16] Oscar Reparaz, Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Additively homomorphic ring-LWE masking. In Takagi [Tak16], pages 233–244.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), September 2009.

[Rij00] Vincent Rijmen. Efficient implementation of the Rijndael S-box. *Katholieke Universiteit Leuven, Dept. ESAT. Belgium*, 2000.

[RMTA18] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. New area record for the AES combined S-box/inverse S-box. In *ARITH*. IEEE, 2018.

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

[RTA18] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. Smashing the implementation records of AES s-box. *IACR TCHES*, 2018(2):298–336, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/884`.

[S+17] William Stein et al. *Sage Mathematics Software Version 8.0*. The Sage Development Team, 2017. `http://www.sagemath.org`.

[SAB+17] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[SAB+19] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[SAL+17] Nigel P. Smart, Martin R. Albrecht, Yehuda Lindell, Emmanuela Orsini, Valery Osheter, Kenny Paterson, and Guy Peer. LIMA. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[SBPV07] Kazuo Sakiyama, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. HW/SW co-design for public-key cryptosystems on the 8051 micro-controller. *Computers & Electrical Engineering*, 33(5-6):324–332, 2007.

[Sch77] Arnold Schönhage. Schnelle multiplikation von polynomen über körpern der charakteristik 2. *Acta Informatica*, 7(4):395–398, Dec 1977.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

[Sch03] Claus-Peter Schnorr. Lattice reduction by random sampling and birthday methods. In Helmut Alt and Michel Habib, editors, *STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, Germany, February 27 - March 1, 2003, Proceedings*, volume 2607 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2003.

[Sch20] Lars Schlieper. In-place implementation of quantum-Gimli. *arXiv preprint arXiv:2007.06319*, 2020.

[SE91] Claus-Peter Schnorr and M Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In *FCT*, 1991.

[SE94] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 1994.

[Sel13] Peter Selinger. Quantum circuits of T-depth one. *Phys. Rev. A*, 87:042302, Apr 2013.

[SGT+18] Krysta M. Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher E. Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling scalable quantum computing and development with a high-level DSL. In *RWDSL@CGO 2018*, 2018.

[Sho94] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[SHT18] Damian S. Steiger, Thomas Häner, and Matthias Troyer. ProjectQ: an open source software framework for quantum computing. *Quantum*, 2:49, January 2018.

[SMTM01] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A compact Rijndael hardware architecture with S-box optimization. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 239–254. Springer, Heidelberg, December 2001.

[SNS+16] Petr Svenda, Matús Nemec, Peter Sekan, Rudolf Kvasnovský, David Formánek, David Komárek, and Vashek Matyás. The million-key question - investigating the origins of RSA public keys. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 893–910, 2016.

[SS71] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3):281–292, Sep 1971.

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 617–635. Springer, Heidelberg, December 2009.

[Tak16] Tsuyoshi Takagi, editor. *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*. Springer, Heidelberg, 2016.

[TEMX17] Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors. *ACM CCS 2017*. ACM Press, October / November 2017.

[Ter15] Barbara M Terhal. Quantum error correction for quantum memories. *Reviews of Modern Physics*, 87(2):307, 2015.

[Too63] Andrei L Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. In *Soviet Mathematics Doklady*, volume 3, pages 714–716, 1963.

[TP17] Tsuyoshi Takagi and Thomas Peyrin, editors. *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*. Springer, Heidelberg, December 2017.

[TP19] Quan Quan Tan and Thomas Peyrin. Improved heuristics for short linear programs. *IACR TCHES*, 2020(1):203–230, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/8398`.

[UHS+15] Rei Ueno, Naofumi Homma, Yukihiro Sugawara, Yasuyuki Nogami, and Takafumi Aoki. Highly efficient GF(2^8) inversion circuit based on redundant GF arithmetic and its application to AES design. In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 63–80. Springer, Heidelberg, September 2015.

[va18] various authors. Post-quantum crypto library for the ARM Cortex-M4. Website, 2018. accessed April 2018, see `https://github.com/mupq/pqm4`.

[Vas21] Dmitry Vasilevsky. Computing width and depth of a quantum program in the Tracer. Technical report, Microsoft Research, January 2021. `https://github.com/microsoft/qsharp-runtime/blob/`

649a3ee8acd27f19aa525c7b9bfb35699fa4e586/src/Simulation/Simulators/QCTraceSimulator/Docs/Width%20and%20Depth%20in%20the%20Tracer.docx.

[vMOG15] Ingo von Maurich, Tobias Oder, and Tim Güneysu. Implementing QC-MDPC McEliece encryption. *ACM Trans. Embedded Comput. Syst.*, 14(3):44:1–44:27, 2015.

[VZGG13] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.

[Wen13] Erich Wenger. A lightweight atmega-based application-specific instruction-set processor for elliptic curve cryptography. In *Lightweight Cryptography for Security and Privacy - Second International Workshop, LightSec 2013*, pages 1–15, 2013.

[WFG21] Karl Wehden, Ismael Faro, and Jay Gambetta. IBM's roadmap for building an open quantum software ecosystem. *IBM Research Blog*, 2021.

[WGY20] Bin Wang, Xiaozhuo Gu, and Yingshan Yang. Saber on ESP32. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *ACNS 20, Part I*, volume 12146 of *LNCS*, pages 421–440. Springer, Heidelberg, October 2020.

[WSH+19] Zihao Wei, Siwei Sun, Lei Hu, Man Wei, Joan Boyar, and Rene Peralta. Scrutinizing the tower field implementation of the $\mathbb{F}_{2^8}$ inverter – with applications to AES, Camellia, and SM4. Cryptology ePrint Archive, Report 2019/738, 2019.

[Wun18] Thomas Wunderer. *On the Security of Lattice-Based Cryptography Against Lattice Reduction and Hybrid Attacks*. PhD thesis, Technische Universität, Darmstadt, 2018. http://tuprints.ulb.tu-darmstadt.de/8082/.

[WZ82] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, Oct 1982.

[YI00] Akihiro Yamamura and Hirokazu Ishizuka. Quantum cryptanalysis of block ciphers (algebraic systems, formal languages and computations). *RIMS Kokyuroku*, 1166:235–243, August 2000.

[Zal99] Christof Zalka. Grover's quantum searching algorithm is optimal. *Phys. Rev. A*, 60:2746–2751, Oct 1999.

[ZC19] Fang Zhang and Jianxin Chen. Optimizing T gates in Clifford+T circuit as $\pi/4$ rotations around paulis. *arXiv preprint arXiv:1903.12456*, 2019.

[ZCD+17] Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, and Daniel Slamanig. Picnic. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[ZCH+19] Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, William Whyte, John M. Schanck, Andreas Hulsing, Joost Rijneveld, Peter Schwabe, and Oussama Danba. NTRUEncrypt. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[ZWS+20] Jian Zou, Zihao Wei, Siwei Sun, Ximeng Liu, and Wenling Wu. Quantum circuit implementations of aes with fewer qubits. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 697–726, Cham, 2020. Springer International Publishing.

This thesis has 256 pages.