Cryptanalysis and Applications of Lattice-based Encryption Schemes

Benjamin R. Curtis

Thesis submitted to the University of London for the degree of Doctor of Philosophy

Information Security Group
Royal Holloway, University of London

2020
Declaration

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled at the Centre for Doctoral Training in Cyber Security in the Department of Information Security 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.

Ben Curtis
Acknowledgements

There are many people that I need to thank for their support throughout the course of my PhD. Firstly, my supervisor Professor Keith Martin. Keith has been a constant source of encouragement and has provided me with many opportunities over the past five years. Thanks for all the wisdom!

Next, my advisor Professor Martin Albrecht. Martin has shown exceptional levels of patience as I have gotten to grips with various aspects of lattice-based cryptography. I have learned a lot from Martin and have appreciated all of his time and effort.

I have been lucky to have had mentorship from many other PhD students, including Dr Rachel Player, Dr Thomas Wunderer, Dr James Alderman, Dr Naomi Farley, and Dr Anamaria Costache. At some point during my PhD, you have provided support and knowledge which has been invaluable to me.

There are many other people who have become good friends over the past five years, especially James, Lydia, Nick, Ivan, Pip, Andreas, Alex, Jason, Torben, Fernando, Giovanni, Joanne, Ela, and Rob. To all of those who I have befriended at conferences, especially Sander, Ryan, and Marcella - it has been a lot of fun.

I also need to thank Professor Johannes Buchmann and Dr Thomas Wunderer for hosting me at TU Darmstadt for a research visit in October 2018, as well as Professor Kurt Rohloff and Professor Yuriy Polyakov for hosting me at New Jersey Institute of Technology for a research visit in April 2019. As well as this, the homomorphiccryption.org community has been welcoming and has provided me with several opportunities.

A special acknowledgement goes to Amit who has taught me many things over the past five years, and still takes the time to explain things to me today. Alongside this you have been a great friend.

Thanks to all of my family members, particularly Mum, Dad, Whoa, and Pop, for providing me with support and encouragement.

Finally, thanks to Eve for all the love, laughs, and for always being there over the past few years. Here’s to many more!
Abstract

The work presented in this thesis is focused around the security of the Learning with Errors (LWE) problem, as well as applications of homomorphic encryption schemes.

In Chapter 1, we provide an overview of the topics discussed in this thesis: lattice-based cryptography, secure computation, cryptanalysis, and standardisation.

In Chapter 2, we introduce necessary background concepts. Specifically, we outline some notions related to lattice-based cryptography and cryptanalysis.

In Chapter 3, we consider trade-offs in “Batch Bounded Distance Decoding”. We consider guess-and-verify decoding (g-v decoding), a porting of the decoding attack on LWE into the case of small and/or sparse secret vectors. This results in a combinatorial trade-off, where components of the secret vector are guessed before batches of BDD instances are solved in a smaller dimension. This attack technique has similarities with the hybrid lattice-reduction and meet-in-the-middle (hybrid-decoding) attack, and we compare and contrast these two techniques throughout. We conclude that, under certain assumptions, our g-v decoding technique outperforms a variant of the hybrid-decoding attack.

In Chapter 4, we analyse submissions to the NIST standardisation process for post-quantum cryptographic algorithms. Specifically, we consider all parameter sets submitted to the first round, for every lattice-based scheme, as well as the cost models used for lattice reduction. We estimate the security of every parameter set, under every cost model, considering both the uSVP and dual attacks (where appropriate). This allows for individual schemes to be compared more easily. As a result of this analysis, we observe that cost models for the BKZ algorithm are not order preserving. That is, if scheme A is “more secure” than scheme B under cost model 1, the same is not necessarily true under cost model 2. Finally we outline the current state of the NIST standardisation process, and provide some estimates for the schemes which have reached the third round.

In Chapter 5, we consider homomorphic encryption-style parameter sets, and explore hybrid attacks. Hybrid attacks are competitive in regimes where the LWE secret is small and/or sparse, so need to be considered for parameter sets used in homomorphic encryption schemes. We consider the effect of secret sparsity on security estimates, and consider the trade-off between bootstrapping complexity and security.

Finally, in Chapter 6, we consider an application of homomorphic encryption: “Private Outsourced Kriging Interpolation”. Kriging is a spatial interpolation algorithm which has
applications in geoscience. We consider the outsourcing of this algorithm using homomorphic encryption, and outline techniques which can be used to protect the sensitive parameters in order to provide an efficient solution.
## Contents

1 Introduction 17

1.1 Lattice-based Cryptography ................................................. 17
1.2 Secure Computation and Applications .................................. 18
1.3 Cryptanalysis ..................................................................... 19
1.4 Standardisation .................................................................. 20
1.5 Organisation and Contributions ........................................... 20
1.6 Source Code ...................................................................... 22

2 Background and Notation 23

2.1 Mathematical Objects ....................................................... 24
2.2 Lattices ............................................................................. 29
2.3 Gram-Schmidt Orthogonalisation ....................................... 34
2.4 Projections ........................................................................ 35
2.5 Hard Problems in Lattices ................................................ 37

2.5.1 Related Problems ........................................................ 43
2.5.2 Small Secrets ............................................................... 44

2.6 SVP Solvers, CVP Solvers, and Lattice Reduction Algorithms . 45

2.6.1 LLL ............................................................................. 46
2.6.2 BKZ ............................................................................ 49

2.7 Babai’s Nearest Plane Algorithm ....................................... 53

2.8 Cryptanalytic Heuristics .................................................... 53

2.9 Solving the Learning with Errors Problem .......................... 56

2.9.1 The Dual Attack .......................................................... 57
2.9.2 The uSVP Attack ........................................................ 57
2.9.3 The Decoding Attack .................................................... 58
2.9.4 Alternative Techniques ................................................ 59

2.10 The Learning with Errors Estimator ................................. 59

2.10.1 Input and Output ......................................................... 59
2.10.2 Implemented Features ................................................ 64

2.11 Public-key Encryption from LWE ...................................... 65

2.12 Homomorphic Encryption ............................................... 67

3 Batch Bounded Distance Decoding 70

3.1 Introduction and Contribution .......................................... 71

3.1.1 An Overview of Decoding Attacks ................................. 72

3.2 A Comparison of Success Conditions ................................. 76

3.3 The Hybrid-decoding Attack .............................................. 78

3.3.1 Common Assumptions in Analyses of the Hybrid-decoding Attack 81
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>An outline of the various attack techniques used to solve the Learning with Errors problem.</td>
</tr>
<tr>
<td>2.1</td>
<td>The vectors ( {b_1, b_2} ) form a basis of the lattice consisting of the blue points, as do the vectors ( {b_3, b_4} ).</td>
</tr>
<tr>
<td>2.2</td>
<td>Two fundamental parallelepipeds, ( P_1 ) and ( P_2 ), of the bases ( B_1 ) and ( B_2 ).</td>
</tr>
<tr>
<td>2.3</td>
<td>An example of ( \lambda_1(L) ) and ( \lambda_2(L) ) in the lattice represented by the blue dots.</td>
</tr>
<tr>
<td>2.4</td>
<td>Code used to generate a random ( q )-ary lattice of dimension 220 with modulus ( q = 2^{15} ) and determinant ( q^{110} ), and perform LLL-reduction.</td>
</tr>
<tr>
<td>2.5</td>
<td>GSO lengths of a random ( q )-ary lattice of dimension 220 with modulus ( q = 2^{15} ) and determinant ( q^{110} ) which has been LLL-reduced using FPYLLL [FPL20].</td>
</tr>
<tr>
<td>2.6</td>
<td>Code used to generate the BKZ-50 reduced lattice basis considered in Figure 2.7.</td>
</tr>
<tr>
<td>2.7</td>
<td>Output GSO lengths of a random ( q )-ary lattice of dimension 220 with modulus ( q = 2^{15} ) and determinant ( q^{110} ) which has been BKZ-( \beta ) reduced for ( \beta \in {10, 30, 50, 70} ) using FPYLLL [FPL20]. For comparison, we also include the LLL basis profile used in Figure 2.5.</td>
</tr>
<tr>
<td>2.8</td>
<td>An example of two custom cost models to be used in the LWE Estimator.</td>
</tr>
<tr>
<td>2.9</td>
<td>Example call to the LWE Estimator with our example parameter set ( n = 256, \alpha = 0.002, ) and ( q = 17500 ).</td>
</tr>
<tr>
<td>2.10</td>
<td>Structure of the top level function \textit{estimate}_\textit{lwe} in the LWE Estimator. Implemented attacks are listed in green, whilst those not implemented are listed in red.</td>
</tr>
<tr>
<td>3.1</td>
<td>Code used to generate the required enumeration dimension which guarantees success in the g-v decoding approach. Note that this code considers the squared norms of the vectors ( |b_i| ), and thus varies slightly from Equation 2.2.</td>
</tr>
<tr>
<td>3.2</td>
<td>A comparison of success conditions. Here, the GSO norms are computed using the Geometric Series Assumption. The dashed blue line represents the expected length of the projected target vector in the projected sublattice ( \pi_1^\perp(L) ). The dashed black line represents the length of the shortest vector in the same projected sublattice ( \pi_1^\perp(L) ) as predicted by the Gaussian Heuristic. Note that we have used the GSA here to produce the GSO norms. This analysis can also be carried out using a BKZ Simulator to generate the GSO norms, and we consider both of these techniques (GSA and BKZ Simulator) throughout this chapter.</td>
</tr>
</tbody>
</table>
3.3 Example of the initial GSO lengths for a $q$-ary lattice of dimension $d = 180$ with $q = 17$ constructed as in Equation (3.5) ........................................ 85
3.4 Example of BKZ-60 reduction on a $q$-ary lattice of dimension $d = 180$ with $q = 17$ and volume $17^{80}$ for bases constructed as in Equation (3.5), along with the output of BKZ simulation and the heuristic from [Wun19] ................. 86
3.5 Example of BKZ-60 reduction on a $q$-ary lattice of dimension $d = 180$ with $q = 17$ and volume $17^{80}$ for a basis constructed as in Equation (3.5), which has been re-randomised via the application of a unimodular matrix. We also plot the output of the BKZ simulator on a basis for the same lattice. .......................... 87

4.1 Enumeration-based cost models used as part of a first round submission to the NIST standardisation process for a lattice of dimension $d = 1024$ with $40 \leq \beta \leq 400$. ................................................................. 112
4.2 Sieving-based cost models, used as part of a first round submission to the NIST standardisation process, for a lattice of dimension $d = 1024$ with $40 \leq \beta \leq 400$. 114
4.3 Estimates of the cost of the primal attack when guessing $\tau$ secret entries for the schemes EMBLEM ($n = 611$) and uRound2.KEM ($n = 500$). ................. 130

5.1 Example LWE (secret) keyspace sizes with $n = 1024$ for binary, ternary, fixed-weight binary, and fixed-weight ternary secrets. ........................................ 150
5.2 Changes applied to the dual attack source code inside the LWE Estimator to produce our hybrid-dual estimates. This replaces lines 1944-1959 in commit 428d6ea of the LWE Estimator. .............................................................. 156
5.3 A comparison of the usvp, dual, hybrid–dual and hybrid–decoding attacks under the BKZ cost model $T(\beta, d) = 2^{0.292 \beta^3 + 16.4 + \log(8d)}$, for the parameter set $n = 1024, q = 2^{40}$ and $\sigma \approx 3.2$ with a sparse ternary secret with a variety of Hamming weights $h \in \{64, 128, 256, 512\}$. .................................................. 158
5.4 Extrapolation to $n = 65536$ and $n = 131072$ using the data from Table 5.5 for the value $\zeta = 1$. Here, we consider the lattice reduction cost model $T_{BKZ}(\beta, d) = 2^{0.292 \beta^3 + 16.4 + \log(8d)}$ and extrapolate using the SageMath function find_fit. Note that the solid lines represent values covered by data points, and the dashed lines represents extrapolation. ................................. 164

6.1 Code used to plot the experimental variogram of a dataset, such as the example dataset presented in Table 6.1 ................................................................. 176
6.2 The experimental variogram for our example dataset, presented in Table 6.1 with $\Delta = 0$ ................................................................................. 177
6.3 A linear model fitted to the experimental variogram using the Pykrige library, for our example dataset presented in Table 6.2 ........................................ 178
6.4 Example interpolation grid using the linear model fitted to the experimental variogram presented in Figure 6.3 ......................................................... 179
6.5 Graphs showing the timing costs of each algorithm......................................... 191
6.6 Graphs showing the timing costs of each algorithm, excluding Outsource......... 192
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A comparison of variants of the Learning with Errors, Learning with Rounding, and NTRU problems.</td>
<td>44</td>
</tr>
<tr>
<td>2.2</td>
<td>The root-Hermite factor for a random $q$-ary lattice of dimension 220 with modulus $q = 2^{35}$ and determinant $q^{10}$ which has been BKZ-$\beta$ reduced for $\beta \in {10, 30, 50, 70}$.</td>
<td>54</td>
</tr>
<tr>
<td>2.3</td>
<td>Input parameters to the LWE Estimator, used to retrieve security estimates for LWE parameter sets.</td>
<td>60</td>
</tr>
<tr>
<td>2.4</td>
<td>Outputs from the LWE Estimator.</td>
<td>63</td>
</tr>
<tr>
<td>3.1</td>
<td>A summary of attacks found in the literature: decoding, drop-and-solve-decoding, and hybrid-decoding, as well as our guess-and-verify decoding technique.</td>
<td>76</td>
</tr>
<tr>
<td>3.2</td>
<td>Complexity estimates for uSVP, dual, and various decoding techniques on our example parameter set with $n = 653, q = 4621, \sigma \approx \sqrt{2 \cdot 3},$ and $\chi_s = B_{100}^{-}$.</td>
<td>92</td>
</tr>
<tr>
<td>3.3</td>
<td>Summary of results for NTRU Prime for a classical guessing (i.e. exhaustive search) approach. Estimates marked with $\dagger$ correspond to standard BDD decoding. Full results can be found in Tables 3.7, 3.8, 3.9, and 3.10.</td>
<td>93</td>
</tr>
<tr>
<td>3.4</td>
<td>Summary of results for Round 5 for a classical guessing (i.e. exhaustive search) approach. Estimates marked with $\dagger$ correspond to standard BDD decoding. Full results can be found in Tables 3.7, 3.8, 3.9, and 3.10.</td>
<td>94</td>
</tr>
<tr>
<td>3.5</td>
<td>Summary of results for Round 5 IoT for a classical guessing (i.e. exhaustive search) approach. Estimates marked with $\dagger$ correspond to standard BDD decoding. Full results can be found in Tables 3.7, 3.8, 3.9, and 3.10.</td>
<td>94</td>
</tr>
<tr>
<td>3.6</td>
<td>Summary of results for HElib for a classical guessing (i.e. exhaustive search) approach. Full results can be found in Tables 3.7, 3.8, 3.9, and 3.10.</td>
<td>95</td>
</tr>
<tr>
<td>3.7</td>
<td>Estimates in the enumeration regime, where BKZ and the BDD solver are instantiated with enumeration algorithms.</td>
<td>96</td>
</tr>
<tr>
<td>3.8</td>
<td>Estimates in the enumeration regime considering a “meet-in-the-middle” approach which does not consider probabilities of failure in the meet-in-the-middle phase. Such an approach considers a square-root speed-up in the guessing phase.</td>
<td>97</td>
</tr>
<tr>
<td>3.9</td>
<td>Estimates in the sieving regime, where BKZ and the BDD solver are instantiated with sieving algorithms. Estimates marked with $\dagger$ correspond to standard BDD decoding.</td>
<td>98</td>
</tr>
<tr>
<td>3.10</td>
<td>Estimates in sieving regime for a “meet-in-the-middle” approach which does not consider probabilities of failure in the meet-in-the-middle phase. Such an approach considers a square-root speed-up in the guessing phase.</td>
<td>99</td>
</tr>
<tr>
<td>3.11</td>
<td>Sets of assumptions considered in this case study.</td>
<td>101</td>
</tr>
</tbody>
</table>
3.12 Enumeration-based estimates, where each section corresponds to a set of assumptions outlined in Table 3.11. “–” denotes a value which is not compatible with our notation (for example, our script considers a simple sqrt speed-up in the search space, the NTRU Prime script considers splitting the search space as in a meet-in-the-middle approach). .................................................. 103

3.13 Sieving-based estimates, where each section corresponds to a set of assumptions outlined in Table 3.11. “–” denotes a value which is not compatible with our notation (for example, our script considers a simple sqrt speed-up in the search space, the NTRU Prime script considers splitting the search space as in a meet-in-the-middle approach). Estimates marked with a † correspond to a uSVP estimate with η,β uncoupled. .................................................. 104

4.1 An example interpretation of the five NIST security levels. ................................................. 108

4.2 Complete and proper lattice-based submissions to the NIST standardisation process. ............. 110

4.3 All cost models proposed as part of a submission to the first round of the NIST standardisation procedure. The name of a model is the log (to the base 2) of its cost. .................................................. 116

4.4 Parameter sets for NTRU-based schemes with secret dimension n, modulo q, small polynomials f and g, and ring \( \mathbb{Z}_q[x]/(\phi) \). The NIST column indicates the NIST security category aimed at. ........................................................................ 117

4.5 LWE parameter sets for NTRU-based schemes, with dimension n, modulo q, standard deviation of the error \( \sigma \), and ring \( \mathbb{Z}_q[x]/(\phi) \). The NIST column indicates the NIST security category aimed at. ........................................................................ 118

4.6 Parameter sets for LWE-based schemes with secret dimension n, Module-LWE rank k (if any), modulo q, standard deviation of the error \( \sigma \). If the LWE samples come from a Ring- or Module-LWE instance, the ring is \( \mathbb{Z}_q[x]/(\phi) \). The NIST column indicates the NIST security category aimed at. *For Titanium \( f_i \in \{-1, 0, 1\} \) and \( f_0 \in \{-1, 1\} \). † For R EMBLEM we list the parameters from the reference implementation since a suitable \( \phi \) could not be found for those proposed in [SPL +17, Table 2]. ........................................................................ 120

4.7 Estimates for the first round submissions EMBLEM, NewHope, NTRU Prime, and uRound2.KEM under the 0.292\( \beta \) and 0.187\( \beta \log(\beta) - 1.019\beta + 16.1 \) cost models. Estimates are taken directly from https://estimate-all-the-lwe-ntru-schemes.github.io/docs/ .................................................. 127

4.8 Security estimates for the first round variants of EMBLEM and uRound2.KEM. Best in class are highlighted in bold, and we can see this is an example of a cost swap: in the 0.292\( \beta \) model uRound2.KEM has a higher security estimate (84-bits vs 76-bits), whereas in the 0.187\( \beta \log(\beta) - 1.019\beta + 16.1 \) model EMBLEM has a higher security estimate (142-bits vs 126-bits). ........................................................................ 128

4.9 Second round lattice-based submissions to the NIST standardisation process. ..................... 132

4.10 Third round lattice-based submissions to the NIST standardisation process. ........................ 132

12
4.11 All cost models proposed as part of a submission to the third round of the NIST standardisation procedure. The name of a model is the log (to the base 2) of its cost. Cost models which were not used as part of our analysis for the first round submissions are marked with a †. The 0.396β model considers the 0.292β model mapped to the AT metric [Ber20]. The 0.3496β and 0.4150β models are used in the NTRU submission under the assumption of a “local” model of computation. The 0.2075β+log(β) model was mentioned in the round one Frodo submission, but wasn’t used to produce estimates.

4.12 Parameter sets for third round NTRU-based schemes with secret dimension n, modulo q, small polynomials f and g, and ring \( \mathbb{Z}_q[x]/(\phi) \). The NIST column indicates the NIST security category aimed at. Each parameter set from SNTRU Prime has been assigned two security levels in the round 3 submission, and we always choose the lowest of the two.

4.13 LWE parameter sets for third round NTRU-based schemes, with dimension n, modulo q, standard deviation of the error \( \sigma \), and ring \( \mathbb{Z}_q[x]/(\phi) \). The NIST column indicates the NIST security category aimed at. Each parameter set from SNTRU Prime has been assigned two security levels in the round 3 submission, and we always choose the lowest of the two.

4.14 Parameter sets for third round LWE-based schemes with secret dimension n, Module-LWE rank k (if any), modulo q, standard deviation of the error \( \sigma \). If the LWE samples come from a Ring- or Module-LWE instance, the ring is \( \mathbb{Z}_q[x]/(\phi) \). The NIST column indicates the NIST security category aimed at. Note that, for the SABER submission, we consider the binomial secret drawn from \( B_{\eta} \) to be uniform over the interval \((-\eta^2, \eta^2)\). Each parameter set from NTRU LPrime has been assigned two security levels in the round 3 submission, and we always choose the lowest of the two.

4.15 “Core-SVP” estimates for third round NTRU-based schemes, with dimension n, modulo q, standard deviation of the error \( \sigma \), ring \( \mathbb{Z}_q[x]/(\phi) \) and with \( m = 2n \) samples. The NIST column indicates the NIST security category aimed at.

4.16 Quantum “Core-SVP” estimates for third round NTRU-based schemes, with dimension n, modulo q, standard deviation of the error \( \sigma \), ring \( \mathbb{Z}_q[x]/(\phi) \), and with \( m = 2n \) samples. The NIST column indicates the NIST security category aimed at.

5.1 A summary of the bootstrapping complexities for CKKS, BGV and BFV.

5.2 Currently standardised LWE parameters at the 128-, 192- and 256-bit security level for uniform ternary secret specified in [ACC+18, Table 1] and estimates of their security against usvp, decoding, and dual attacks under the BKZ cost model \( T(\beta, d) = 2^{0.2925\beta + 16.4 + \log(8d)} \), where \( \beta \) is the blocksize and \( d \) is the dimension. The best performing attack for each parameter set is highlighted in bold.
5.3 Impact of using a sparse ternary secret of Hamming weight $h = 128$, using the currently standardised LWE parameter sets at the target 128-, 192- and 256-bit security level for uniform ternary secret specified in Table 5.2. Estimates of the security of each parameter set against usvp and dual attacks under the BKZ cost model $T(\beta, d) = 2^{0.292\beta + 16.4 + \log(8d)}$ are presented, where $\beta$ is the blocksize and $d$ is the dimension. The best performing attack for each parameter set is highlighted in bold.

5.4 Bit-length $\log q$ of moduli required to provide target security level $\lambda$, for $\lambda \in \{128, 192, 256\}$, for various secret densities. We note that the LWE Estimator treats uniform ternary secrets as ternary fixed-weight secrets with Hamming weight $h = \left\lceil \frac{2n}{3} \right\rceil$. The best performing attack for each parameter set is highlighted in bold.

5.5 The reduction in bit-length $\log q$ of the modulus $q$ required to retain the desired level of security against the dual, usvp, hybrid-dual and hybrid-decoding attacks, under our assumptions, when using a sparse ternary secret parameterised by $\zeta = \frac{\lambda}{h}$ compared to a uniform ternary secret. The lattice reduction cost model is $T(\beta, d) = 2^{0.292\beta + 16.4 + \log(8d)}$, and a (conservative) estimate for both the hybrid-dual and hybrid-decoding attacks are obtained by considering a square-root speed-up in the search space, and ignoring any meet-in-the-middle probabilities.

5.6 Required bit-length $\log q$ of moduli required to attain target security level $\lambda$ under the usvp and dual attacks, with $\lambda \in \{128, 192, 256\}$, for dimension $n = 65536$, under the sieving-based cost model $T(\beta, d) = 2^{0.292\beta + 16.4 + \log(8d)}$.

5.7 Maximal moduli $q$ required to attain target security level $\lambda = 128$ for various values of $\sigma$. In each case, we consider a ring dimension of $n = 8192$ and a fixed weight ternary secret with Hamming weight $h = 128$.

6.1 The toy dataset used throughout this section, as in the ordinary kriging example used in the PyKriging library [PYK20]. The $x_i, y_i$ values are coordinates of the measurement $z_i$.

6.2 Data protection offered by our private outsourced Kriging scheme.
List of Algorithms

1. The LLL Algorithm ................................................................. 47
2. Babai’s Nearest Plane algorithm ........................................... 53
List of Publications

This thesis is based on the following publications:


Introduction

Chapter 1

Contents

1.1 Lattice-based Cryptography ........................................... 17
1.2 Secure Computation and Applications .............................. 18
1.3 Cryptanalysis ............................................................ 19
1.4 Standardisation .......................................................... 20
1.5 Organisation and Contributions ..................................... 20
1.6 Source Code ............................................................... 22

1.1 Lattice-based Cryptography

Cryptography underpins society as we know it. From secure messaging applications to online payments, cryptography allows us to perform many daily tasks over the internet in a secure manner—often without the knowledge that we are even using it. Currently used cryptographic primitives whose security relies on RSA [RSA78] or discrete logarithm problems [DH76] can be broken by sufficiently large quantum computers using Shor’s algorithm [Sho97]. Thus begins the search for quantum-secure cryptography, known more commonly as post-quantum cryptography. The US National Institute of Standards and Technology (NIST) started a standardisation procedure in 2016 [Nat16], with the goal being to design, analyse, and standardise a portfolio of public-key encryption schemes and digital signature algorithms.

A variety of candidate post-quantum secure algorithms have been proposed, including code-based submissions, lattice-based submissions, isogeny-based submissions, and multivariate-based submissions.

Of the candidate post-quantum secure algorithms submitted to this process, lattice-based cryptographic algorithms represent a strong candidate for standardisation and have been tested in practice [Bra16]. There are three major roadblocks to the wide-scale deployment of lattice-based cryptosystems: usability, confidence in the underlying security assumptions, and
standardisation. Indeed, cryptosystems must be usable, and secure, for wide-scale deployment to be considered and, moreover, standardisation is essential for many forms of industrial or governmental usage. In this thesis, we consider each of these aspects individually.

The hardness of lattice-based cryptographic primitives typically relies on the Learning with Errors (LWE) problem. LWE (and its variants) has given rise to many advanced encryption techniques such as homomorphic encryption, which has lead to a variety of additional interesting areas of research.

1.2 Secure Computation and Applications

Whilst typical encryption techniques allow us to secure data-at-rest and data-in-transit, once data has been encrypted we are no longer able to perform operations on this data. Fully homomorphic encryption [Gen09] allows for computation to occur on data which remains encrypted, giving rise to an abundance of privacy-preserving applications. One of the major bottlenecks with fully homomorphic encryption is the efficiency of the bootstrapping step used to refresh a noisy ciphertext. Solutions used in practice tend to avoid this expensive procedure in favour of somewhat homomorphic encryption, which allows for a limited computation, such as polynomial functions up to some maximal degree $d$, to be evaluated in a secure manner.

There are a variety of other secure computation technologies available for use in the wild today. Garbled circuits [Yao86] allow for the private evaluation of boolean circuits, oblivious RAM [Gol87] allows for access patterns to be protected, and secure multiparty computation allows several parties to jointly compute a function over their private inputs. These techniques can be combined with homomorphic encryption to yield efficient privacy-preserving solutions to interesting problems, e.g. [CCD+19]. We are now in an era where secure computation is being used in commercial products [Env20, Unb20, Dua20, Zam20], which gives an indication of the practicality of these techniques.

In this thesis, we are interested in both the application, and security, of homomorphic encryption encryption schemes. In order to provide increased efficiency, the parameter sets used typically do not come under the umbrella of the various security reductions outlined in the literature. These adaptations come with a concrete security loss\footnote{Here we are referring to both the tightness of reductions, for example the reduction from LWR to LWE,} and it is important
1.3 Cryptanalysis

to quantify this loss of security so that homomorphic encryption-style parameter sets can be used with confidence.

1.3 Cryptanalysis

The security of LWE-based cryptographic constructions is measured in terms of the running times of the best-known attacks. The LWE Estimator of Albrecht et al. [APS15, Est20] is a common tool used for estimating the running time of a subset of these attacks on a given set of LWE parameters \((n, \alpha, q)\). For other variants of the LWE, such as those used in homomorphic encryption schemes where the secret vector is small and/or sparse, hybrid attacks are also competitive.

All of the attacks considered in this thesis involve the usage of lattice-reduction algorithms, which, generically, find short vectors in projected sublattices of size \(\beta\), referred to as the blocksize, in order to find a short basis of a given input lattice. To begin, we reduce the LWE problem to a lattice problem such as the unique Shortest Vector problem (uSVP), the Bounded Distance Decoding problem (BDD) or the Short Integer Solutions problem (SIS) before solving the lattice problem at hand via lattice-reduction techniques.

![Figure 1.1: An outline of the various attack techniques used to solve the Learning with Errors problem.](image)

Of particular interest in this thesis are hybrid attacks, which exploit any smallness and/or sparsity in the LWE secret. Often, it is advantageous to guess components of the LWE secret and the chosen secret distribution.
1.4 Standardisation

(or error) vector before performing lattice reduction in a smaller dimension. This creates a
trade-off between guessing correctly and the reduced cost of performing lattice reduction in a
smaller dimension. Hybrid attacks involve a myriad of trade-offs, and are notoriously difficult
to optimise. To this end, we explore a variety of these trade-offs as well as the assumptions
which are made in analyses of hybrid attacks. As part of our work, we have released open-
source code which allows for the complexity of hybrid attacks to be estimated under a variety
of assumptions\footnote{This code can be found at \url{github.com/bencrts/hybrid_attacks}.} We have also contributed to the LWE estimator, to help keep the output estimates in line with state-of-the-art attacks.

1.4 Standardisation

There are two ongoing standardisation procedures of interest in this thesis. The first, as
discussed in Section 1.1, is the NIST standardisation process for post-quantum algorithms,
and we consider the security of all of the schemes submitted to first and third rounds of this
process.

The second standardisation procedure is the ongoing effort to standardise aspects of homomorphic
encryption, and the homomorphicencryption.org consortium are leading this effort. The
Homomorphic Encryption Security Standard (HE Standard) \cite{ACC+18} recommends secure
parameters for use in homomorphic encryption schemes, and we discuss and analyse some of
these parameter selections in this thesis. We also consider potential future extensions to the
HE Standard, and outline several points for consideration in future work.

1.5 Organisation and Contributions

In Chapter 2 we outline all necessary background content required. This includes preliminary
notions from lattice-based cryptography and cryptanalysis. We then introduce our four
contributions in Chapters 3, 4, 5 and 6.

1. In Chapter 3 we discuss the Batch Bounded Distance Decoding problem and its application
to solving the Small-secret Learning with Errors problem. We compare this technique
1.5 Organisation and Contributions

to Howgrave-Graham’s hybrid lattice-reduction and meet-in-the-middle attack (hybrid-decoding attack), and discuss how important assumptions are in these analyses. We show that, under certain assumptions, our attack technique outperforms a variant of the hybrid-decoding attack and, under other sets of assumptions, the converse is true. This chapter corresponds to the publication *Exploring trade-offs in batch bounded distance decoding* detailed in the List of Publications section of this thesis.

2. In Chapter 4 we discuss the concrete security of the submissions to the NIST post-quantum standardisation process. We highlight the importance of lattice reduction cost models and estimate the security of each parameter set for every scheme under all cost models considered as part of a first round submission. This technique allows for the security of two given schemes to be compared in a fair manner. As part of this work, we observe that lattice-reduction cost models are not *order preserving*, meaning that if scheme A is harder to break than scheme B under cost model 1, the same is not necessarily true under cost model 2. Moreover, we provide an update regarding the current state of the standardisation process. This chapter corresponds to the publication *Estimate all the \{LWE, NTRU\} schemes!* detailed in the List of Publications section of this thesis.

3. In Chapter 5 we discuss homomorphic encryption standardisation, and focus on the security of Sparse-secret LWE parameter sets. We outline the current state of the HE standard, and consider potential extensions to this standard, including sparse secret distributions. We outline parameter sets which balance security and efficiency, with a focus on the cost of the expensive bootstrapping procedure required in fully homomorphic encryption schemes. This chapter corresponds to the publication *On the feasibility and impact of standardising sparse-secret LWE* detailed in the List of Publications section of this thesis.

4. In Chapter 6 we discuss an application of homomorphic encryption: the private outsourcing of Kriging interpolation. Kriging is a spatial interpolation algorithm which provides the best linear unbiased prediction (BLUP) of an observed phenomenon, by taking a weighted average of samples within a specified neighbourhood. Kriging is widely used in areas such as geo-statistics where, as an example, it may be used to predict the quality of mineral deposits at an unobserved location based on previous measurements. In our work, we tweak the underlying algorithms to allow this process to be securely outsourced in an efficient manner. In particular, we build a construction which allows for the Kriging process to carried out when the measurement values are encrypted via a
1.6 Source Code

homomorphic encryption scheme. This chapter corresponds to the publication *Private outsourced kriging interpolation* detailed in the List of Publications section of this thesis.

## 1.6 Source Code

Where possible, source code used as part of this thesis has been made publicly available.

1. The code used in Chapter 2 is available here: [github.com/bencrts/thesis/code/background](https://github.com/bencrts/thesis/code/background)

2. The code used in Chapter 3 is available here: [github.com/bencrts/thesis/code/batchbdd](https://github.com/bencrts/thesis/code/batchbdd)


4. The code used in Chapter 5 is available here: [github.com/bencrts/thesis/code/hestandard](https://github.com/bencrts/thesis/code/hestandard)

5. The code used in Chapter 6 is available here: [github.com/bencrts/thesis/code/kriging](https://github.com/bencrts/thesis/code/kriging)

In the event that any of the above links become broken, please contact the author for a copy of the desired source code.
In this chapter, we introduce all relevant background mathematics required in this thesis. We introduce the notion of a lattice and consider various properties of lattices, as well as heuristics considered in lattice-based cryptography and cryptanalysis.
2.1 Mathematical Objects

Euclidean space of dimension \( n \) is denoted \( \mathbb{R}^n \). All logarithms are to the base two, unless otherwise stated. Column vectors are denoted by lower case bold letters, e.g. \( \mathbf{b} \), and matrices by upper case bold letters, e.g. \( \mathbf{B} \). The transpose of the matrix \( \mathbf{B} \) is denoted by \( \mathbf{B}^\top \). The \( i \)th component of a vector \( \mathbf{b} \) is denoted by \( b_i \), and the \( (i,j) \)th entry of a matrix \( \mathbf{B} \) is denoted by \( B_{i,j} \), where all indices start from one. We write \( \mathbf{B}_i \) for the \( i \)th column of \( \mathbf{B} \). The inner product of two vectors \( \mathbf{b}_1 \) and \( \mathbf{b}_2 \) is written as \( \langle \mathbf{b}_1, \mathbf{b}_2 \rangle \). We write \( (\mathbf{b}_1, \mathbf{b}_2) \) for the vector formed by concatenating the entries of the two vectors \( \mathbf{b}_1 \) and \( \mathbf{b}_2 \). The same notation is used for the concatenation of the vector \( \mathbf{b}_1 \) and the scalar \( c \) as \( (\mathbf{b}_1, c) \). Similarly, we denote the concatenation of \( k \) column vectors, each of length \( d \), into a \((d \times k)\) matrix as \( [\mathbf{b}_1 | \mathbf{b}_2 | \cdots | \mathbf{b}_k] \).

We write \( \mathbf{B}_{(\tau)} \) to represent the \((d \times (k-\tau))\) sub-matrix of \( \mathbf{B} \) constructed via dropping the first \( \tau \) columns of \( \mathbf{B} \), i.e. \( \mathbf{B}_{(\tau)} = [\mathbf{b}_{\tau+1} | \mathbf{b}_{\tau+2} | \cdots | \mathbf{b}_k] \). Similarly we use \( \mathbf{b}_{(\tau)} \) to denote dropping the first \( \tau \) components of the vector \( \mathbf{b} \), i.e. \( \mathbf{b}_{(\tau)} = (b_{\tau+1}, b_{\tau+2}, \ldots, b_k) \). We identify polynomials \( f = \sum_{i=1}^{n} f_i x^{i-1} \) with their coefficient vectors \( \mathbf{f} = (f_1, f_2, \ldots, f_n) \). The Euclidean norm of a vector \( \mathbf{v} \) is defined to be \( \|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2} \).

**Definition 2.1 (Discrete Gaussian Distribution)** A discrete Gaussian distribution centred at \( \mu \) and with width parameter \( r \) samples elements with probability:

\[
\exp \left( -\pi \frac{(x - \mu)^2}{r^2} \right).
\]

Throughout this thesis, we typically consider discrete Gaussian distributions over the integers to be centred at zero (i.e. \( \mu = 0 \)). The standard deviation of such a distribution is \( \sigma = \frac{r}{\sqrt{2\pi}} \), and we denote this distribution by \( D_\sigma \).

Throughout this thesis the term **ring** is reserved for a commutative ring \( R \) with a multiplicative identity element \( 1_R \) (we denote the additive identity element of \( R \) by \( 0_R \)).

**Example 2.1** For an integer \( q \), we consider values in the set \( \mathbb{Z} \cap (-q/2, q/2] \) to be representatives of \( \mathbb{Z}_q \). For example, if \( x \in \mathbb{Z}_2 \), \( x \) can take values in the set:

\[ \mathbb{Z}_2 = \{0, 1\} \]

and, if \( x \in \mathbb{Z}_3 \), \( x \) can take values in the set:

\[ \mathbb{Z}_3 = \{-1, 0, 1\}. \]
2.1 Mathematical Objects

Definition 2.2 (Ideal) The set $I \neq \emptyset$ is an ideal of the ring $R$ if:

1. $I$ is a subgroup of the additive group of $R$, and
2. for each $r \in R$ and $x \in I$ we have that $xr \in I$.

Definition 2.3 (Field) A field $K$ is a ring which also has the property that it is closed under multiplicative inverses, i.e. the non-zero elements of $K$ form a multiplicative group.

Example 2.2 For a ring $R$, an ideal generated by a single ring element $f \in R$ is denoted by $(f)$, and this ideal is made up of elements in the set:

$$(f) = \{ fr \mid r \in R \}.$$

Definition 2.4 (Maximal Ideal) A proper ideal $I \subseteq R$ of a ring $R$ is maximal if for any ideal $J$ satisfying $I \subseteq J$, we have either $J = I$ or $J = R$.

Definition 2.5 (Quotient Ring) Given a ring $R$ and an ideal $I$ of $R$, we can define the quotient ring $R/I$ which is formed of cosets $r + I$, for $r \in R$, of the additive group of $I$ in $R$. We define the addition and multiplication operations in the quotient ring $R/I$ in the following way:

$$(I + r_1) + (I + r_2) = I + (r_1 + r_2)$$

$$(I + r_1)(I + r_2) = (I + r_1r_2),$$

for all $r_1, r_2 \in R$.

We can think of a quotient ring as a set of equivalence classes under the equivalence relation:

$$[x] \sim [y] \text{ if and only if } x - y \in I.$$ 

That is, two elements $x, y \in R/I$ are equivalent if and only if their difference $x - y$ is contained within the ideal $I$. We note that an ideal $I$ in the ring $R$ is maximal if and only if the quotient ring $R/I$ is a field. A ring homomorphism $\phi$ is a structure-preserving map between two rings $R_1$ and $R_2$. 

25
Definition 2.6 (Ring Homomorphism) The map $\phi : R \to S$ is a ring homomorphism from the ring $R$ to the ring $S$ if it satisfies the three conditions:

1. $\phi(1_R) = 1_S$,
2. $\phi(r + s) = \phi(r) + \phi(s)$, and
3. $\phi(rs) = \phi(r)\phi(s)$,

for all $r \in R$ and $s \in S$, where $1_R$ and $1_S$ are the multiplicative identity elements of the rings $R$ and $S$ respectively.

Throughout this thesis we are particularly interested in polynomial rings of the form $\mathbb{Z}_q[X]/(f)$ for some polynomial $f \in \mathbb{Z}_q[X]$. In this case, two elements $g, h \in \mathbb{Z}_q[X]/(f)$ are in the same equivalence class if and only if $g - h \in (f)$, i.e. $g - h = fk$ for some $k \in \mathbb{Z}[X]$. Multiplication in polynomial quotient rings works as in typical polynomial multiplication, with the additional condition that we work modulo the quotient polynomial $f$. For a given ring $R$, we denote the ring $R/qR$ as $R_q$.

When we consider elements $a \in R_q$, we can represent $a$ as a vector $\mathbf{a}$ of $n$ coefficients in $\mathbb{Z}_q$. Therefore, for two ring elements $a, b \in R_q$, we can compute the polynomial sum $a + b$ by considering an element-wise addition of the vectors $\mathbf{a}$ and $\mathbf{b}$. That is:

$$\mathbf{a} + \mathbf{b} = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n),$$

and $\mathbf{a} + \mathbf{b}$ is the coefficient vector of $a + b$. For multiplication, the situation is a little more complicated: multiplication of two ring elements, i.e. $ab$ corresponds to matrix/vector multiplication. That is, there exists a matrix $P_a$ such that, for any ring element $b \in R_q$ we can compute the product $c = ab$ via:

$$c = P_a \mathbf{b}.$$

More specifically, the matrix $P_a$ is of the form:

$$P_a = \begin{pmatrix}
    a_1 & a_2 & \cdots & a_n \\
    (xa)_1 & (xa)_2 & \cdots & (xa)_n \\
    \vdots & \vdots & \ddots & \vdots \\
    (x^{n-1}a)_1 & (x^{n-1}a)_2 & \cdots & (x^{n-1}a)_n
\end{pmatrix},$$

where $(x^i a)_j$ is the $j^{th}$ component of the coefficient vector of the polynomial $x^i a$. 
Example 2.3 Let $R_q = \mathbb{Z}_3[X]/(x^5 - 1)$. This ring can be identified with the set of polynomials with coefficients in $\mathbb{Z}_3$ up to degree 4, i.e. the set:

$$\{a_1 + a_2 x + a_3 x^2 + a_4 x^3 + a_5 x^4 \mid a_i \in \mathbb{Z}_3, 1 \leq i \leq 5\}.$$

Consider the polynomial $g = a_1 + a_2 x + a_3 x^2 + a_4 x^3 + a_5 x^4$. We note that:

$$xg = x(a_1 + a_2 x + a_3 x^2 + a_4 x^3 + a_5 x^4)$$

$$= a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5$$

$$= a_5 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4,$$

since $x^5 \equiv 1 \pmod{x^5 - 1}$. In this case, the matrix $P_g$ is given by:

$$P_g = \begin{pmatrix}
    a_1 & a_2 & a_3 & a_4 & a_5 \\
    a_5 & a_1 & a_2 & a_3 & a_4 \\
    a_4 & a_5 & a_1 & a_2 & a_3 \\
    a_3 & a_4 & a_5 & a_1 & a_2 \\
    a_2 & a_3 & a_4 & a_5 & a_1
\end{pmatrix}.$$

Definition 2.7 (Irreducible Polynomial) Given a field $K$, a non-constant polynomial $f \in K[X]$ is irreducible over $K$ if it cannot be factored into two non-constant polynomials $g, h \in K[X]$.

In typical examples considered in this thesis, the (non-constant) polynomial $f$ is irreducible in $\mathbb{Z}_q[X]$, and we consider rings of the form $\mathbb{Z}_q[X]/(f)$.

Example 2.4 The polynomial $f = X^2 + 1$ is irreducible over $\mathbb{Q}$ since we cannot write:

$$X^2 + 1 = (X + a)(X + b),$$

with $a, b \in \mathbb{Q}$. We note that $f$ is not irreducible over $\mathbb{C}$ since we can write $f$ as:

$$X^2 + 1 = (X + i)(X - i).$$

Example 2.4 motivates the discussion of an extension field.
2.1 Mathematical Objects

**Definition 2.8 (Extension Field)** If $F, K$ are two fields such that $F$ is a subfield of $K$, then we refer to $K$ as an extension field of $F$.

**Example 2.5** Consider the field $\mathbb{Q}$. Clearly $\pm \sqrt{2} \notin \mathbb{Q}$, and so the polynomial $p = x^2 - 2 \in \mathbb{Q}[X]$ is irreducible over $\mathbb{Q}$. Since $p$ is irreducible over $\mathbb{Q}$, the ideal $(p)$ in $\mathbb{Q}[X]$ is maximal and therefore the quotient ring $R = \mathbb{Q}[X]/(x^2 - 2)$ is a field. Since $\mathbb{Q}$ is a subfield of $R$, we have that $R$ is an extension field of $\mathbb{Q}$.

We also consider the notion of a module $M$ over a ring $R$.

**Definition 2.9 (Module)** Given a ring $R$, an abelian group $M$ is called an $R$-Module if there exists an operation $\odot : R \times M \to M$ such that, for all $m, n \in M$ and $r, s \in R$, we have:

1. $r \odot (m + n) = r \odot m + r \odot n$,
2. $(r + s) \odot m = r \odot m + s \odot m$,
3. $(r \cdot s) \odot m = r \odot (s \odot m)$, and
4. $1_R \odot m = m$,

where $\cdot, +$ denote the regular ring operations.

We will consider various cryptographic constructions built over rings and modules. In a similar manner to the notion of a vector space $K^d$ built considering $d$-tuples of field elements from $K$, we can build a module by forming tuples of elements from a ring $R$.

**Example 2.6** Consider the ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$. We define:

$$M := \{(x_1, x_2, \ldots, x_k) \mid x_1, x_2, \ldots, x_k \in R_q\}.$$

The set $M$ is an $R_q$-module since for $s \in R_q$ and $(x_1, x_2, \ldots, x_k) \in M$ we can define a multiplication operation $\odot : R_q \times M \to M$ such that:

$$s \odot (x_1, x_2, \ldots, x_k) = (sx_1, sx_2, \ldots, sx_k),$$

leading to the four conditions in Definition 2.9 being satisfied. In this example, $k$ is referred to as the module rank. Setting $k = 1$, we can see that $R_q$ itself is an $R_q$-module.
2.2 Lattices

Definition 2.10 (Lattice) A lattice $L$ in $\mathbb{R}^d$ is the set:

$$L = \left\{ \sum_{i=1}^{n} v_i b_i \mid v_i \in \mathbb{Z} \right\},$$

of all integer combinations of a set of $n$ linearly independent vectors $\{b_1, b_2, \ldots, b_n\}$ in $\mathbb{R}^d$, where $d \geq n$.

The integer $n$ is known as the rank of the lattice, and the integer $d$ is the dimension of the lattice. If $n = d$ then the lattice is known as full rank. The set of linearly independent column vectors $\{b_1, b_2, \ldots, b_n\}$ is known as a basis of the lattice, and is typically represented in matrix form $B = [b_1 | b_2 | \cdots | b_n] \in \mathbb{R}^{d \times n}$. We write $L(B)$ to denote the lattice generated by the columns of the matrix $B$. In particular, this lattice is made up of the set:

$$L(B) = \{Bv \mid v \in \mathbb{Z}^n\}.$$

In Figure 2.1, we present an example of two different bases of the same lattice.

![Figure 2.1: The vectors $\{b_1, b_2\}$ form a basis of the lattice consisting of the blue points, as do the vectors $\{b_3, b_4\}$.](image)
2.2 Lattices

Definition 2.11 (Span) The span of a set of vectors $S$ over a field $K$ is defined to be:

$$\text{span}(S) = \left\{ \sum_{i=1}^{\ell} c_i s_i \bigg| \ell \in \mathbb{N}, c_i \in K, s_i \in S \right\}.$$ 

That is, the span of the set $S$ is the set of finite linear combinations of elements of $S$, with coefficients drawn from the underlying field $K$. Given a full-rank lattice $L$, the dual lattice $L^*$ is the set of vectors in the span of $L$ which have integer inner product with all lattice points in $L$.

Definition 2.12 (Dual Lattice) Let $L$ be a full rank lattice of dimension $d$. The dual lattice of $L$ is defined to be:

$$L^* = \{ v \in \text{span}(L) \mid \forall w \in L, \langle v, w \rangle \in \mathbb{Z} \}.$$ 

It can be shown that the dual lattice is a lattice.

Proposition 2.1 Let $L$ be a lattice with basis $B$. The dual lattice of $L(B)$, denoted by $L(B)^*$, is a lattice with basis $B(B^\top B)^{-1}$.

To prove this result, we show that $L(B(B^\top B)^{-1}) \subseteq L(B)^*$ and $L(B)^* \subseteq L(B(B^\top B)^{-1})$. We will need to use the fact that:

$$(B(B^\top B)^{-1})^\top B = ((B^\top B)^{-1})^\top B^\top B$$

$$= ((B^\top B)^{-1})B^\top B$$

$$= I$$

Proof. Set $C = B(B^\top B)^{-1}$. Note that:

$$Cx = B(B^\top B)^{-1}x$$

$$= B((B^\top B)^{-1}x)$$

and therefore we have that $Cx \in \text{span}(L(B))$. Moreover, for $y = Cx \in \text{span}(L(B))$ and
2.2 Lattices

$w = Bv \in L(B)$, we have:

$$\langle y, w \rangle = \langle Cx, w \rangle = \langle x, C^\top w \rangle = \langle x, (B(B^\top B)^{-1})^\top Bv \rangle = \langle x, v \rangle \in \mathbb{Z}$$

and, therefore, we have that $Cx \in L(B)^*$ by Definition 2.12. This gives us $L(C) \subseteq L^*(B)$.

To prove that $L(B)^* \subseteq L(C)$, we consider $z \in L(B)^*$ and aim to show that $z \in L(C)$. Since $z \in L(B)^*$, we have that $z \in \text{span}(L(B))$ and, for all $r \in L(B)$, we have $(z, r) \in \mathbb{Z}$ by Definition 2.12. Further, we have that $z = Bs$ for some vector $s \in \mathbb{R}^d$, and, therefore:

$$z = Bs = B(B^\top B)^{-1}(B^\top B)s = C(Bz)$$

and, since each component of $Bz$ is an integer, we have $L(B)^* \subseteq L(C)$. Therefore, we have $L(B)^* \subseteq L(C)$, as required. □

We will also encounter $q$-ary lattices when considering cryptanalytic attacks on lattice-based cryptosystems.

**Definition 2.13 (q-ary Lattice)** A q-ary lattice is a lattice $L$ which satisfies:

$$q\mathbb{Z}^d \subseteq L \subseteq \mathbb{Z}^d,$$

for some integers $q, d$.

**Example 2.7** For a matrix $A \in \mathbb{Z}_q^{n \times m}$, where $q, m, n \in \mathbb{N}$, the lattice:

$$L_q(A) = \{ v \in \mathbb{Z}^m \mid vA = 0 \text{ mod } q \},$$

is q-ary: clearly $L \subseteq \mathbb{Z}^n$, and if $y \in q\mathbb{Z}^m$ then $yA \equiv 0 \text{ mod } q$ meaning that $q\mathbb{Z}^n \subseteq L$. Therefore, we have $q\mathbb{Z}^n \subseteq L \subseteq \mathbb{Z}^n$ as required.
2.2 Lattices

Definition 2.14 (Unimodular Matrix) Let \( U \in \mathbb{Z}^{d \times d} \) be a matrix with all integer entries. The matrix \( U \) is referred to as unimodular if the determinant of \( U \) is in the set \( \{1, -1\} \).

Proposition 2.2 Let \( B_1 \) and \( B_2 \) be the matrix representations of two lattice bases. Then:

\[
L(B_1) = L(B_2),
\]

if and only if \( B_1 = UB_2 \) for some unimodular matrix \( U \).

There are infinitely many unimodular matrices of dimension \( d \geq 2 \). For dimension two we note that:

\[
det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc,
\]

and there are infinitely many solutions to the integer equation \( ad - bc = \pm 1 \). The same holds true for all larger dimensions (by induction). For dimension \( d \geq 2 \), a lattice has infinitely many bases and there are a variety of proofs for this result. Informally, for full rank lattices, this can be seen as a combination of the facts that:

1. there are infinitely many unimodular matrices of dimension \( d \geq 2 \),
2. full rank lattices admit an invertible basis matrix (thus \( U_1B = U_2B \iff U_1 = U_2 \)), and
3. Proposition 2.2

Definition 2.15 (Fundamental Parallelepiped) Given a lattice basis \( B \), we define the set:

\[
P(B) = \{Bx \mid x \in \mathbb{R}^n \text{ such that } \forall i, 0 \leq x_i < 1\}.
\]

as the fundamental parallelepiped of the lattice basis \( B \).

Note that the shape of this parallelepiped depends on the lattice basis \( B \) under consideration. Two such parallelepipeds can be seen in Figure 2.2.

We are also interested in the shifted fundamental parallelepiped.
Definition 2.16 (Shifted Fundamental Parallelepiped) Given a lattice basis $B$, we define the set:

$$P(B) = \{Bx \mid x \in \mathbb{R}^n \text{ such that } \forall i, -\frac{1}{2} \leq x_i < \frac{1}{2}\}.$$ 

as the shifted fundamental parallelepiped of the lattice basis $B$.

Definition 2.17 (Volume of a Lattice) The volume of a lattice $L(B)$ is defined to be:

$$\text{Vol}(L(B)) = \sqrt{\det(B^T B)}.$$

Note that if $B_1 = UB$ for some unimodular matrix $U \in \mathbb{Z}^{(d \times d)}$, then:

$$\text{Vol}(L(B_1)) = \sqrt{\det(B_1^T B_1)}$$

$$= \sqrt{\det(B^T U^T U B)}$$

$$= \sqrt{\det(B^T B)}$$

$$= \text{Vol}(L(B)),$$

since by definition $U^T U = I$. This is as expected, since $B_1$ and $B$ are bases of the same lattice, and volume is a lattice invariant.
2.3 Gram-Schmidt Orthogonalisation

For a lattice basis $\mathbf{B} = [\mathbf{b}_1 \mid \mathbf{b}_2 \mid \cdots \mid \mathbf{b}_d]$, we can define the corresponding Gram-Schmidt orthogonalised (GSO) vectors $\mathbf{B}^* = \{\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_d^*\}$ as:

$$
\begin{align*}
\mathbf{b}_1^* &= \mathbf{b}_1 \\
\mathbf{b}_i^* &= \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*,
\end{align*}
$$

where the Gram-Schmidt coefficients $\mu_{i,j}$ are given by:

$$
\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}.
$$
2.4 Projections

The vector $b_i^*$ is the vector $b_i$ with contributions in the directions of $\{b_1^*, b_2^*, \ldots, b_{i-1}^*\}$ removed, i.e. $b_i^*$ is the component of the vector $b_i$ which is orthogonal to the hyperplane defined by the set of vectors $\{b_1^*, b_2^*, \ldots, b_{i-1}^*\}$. This means that the vectors $b_i^*$ and $b_j^*$, for $i \neq j$, are pairwise orthogonal, i.e. $\langle b_i^*, b_j^* \rangle = 0$.

The volume of the lattice $L(B)$ can be computed as the product of the lengths of the GSO vectors:

$$\text{Vol}(L(B)) = \prod_{i=1}^{d} \|b_i^*\|,$$

and, as outlined earlier, can also be computed as the determinant of the Gram matrix $B^T B$, comprising of the coefficients $\mu_{i,j}$, i.e. $(B^T B)_{i,j} = \mu_{i,j}$. We note that the vectors $B^* = [b_1^* | b_2^* | \cdots | b_d^*]$ in general do not form a basis of the lattice $L(B)$, however these two bases do span the same space i.e. $\text{span}(B) = \text{span}(B^*)$.

When considering cryptanalytic attacks against lattice-based cryptosystems, the lengths of the GSO vectors for a given lattice basis $B$ are an important quantity, which we refer to as the GSO profile.

**Definition 2.19 (GSO Profile)** For a lattice $L$ of rank $n$, we define the GSO Profile of the lattice basis $B$ to be the set of GSO lengths, that is:

$$\{\|b_i^*\| \mid 1 \leq i \leq d\}^T$$

2.4 Projections

**Definition 2.20 (Projection)** Given a vector $v$ and a non-zero vector $u$, we define the projection of $v$ onto the direction of $u$ to be:

$$\pi_u(v) = \frac{\langle u, v \rangle}{\langle u, u \rangle} u.$$

We are particularly interested in projections onto, and orthogonal to, basis vectors of a lattice $L$.

\^1Note that the GSO profile is sometimes defined in terms of the squared GSO lengths.
2.4 Projections

Definition 2.21 (Parallel Projection) Given a lattice with basis $B$ we write $\pi_{i,B}^\parallel(x)$, $1 \leq i \leq d$, to denote the parallel projection of $x$ onto the space spanned by the set of vectors $\{b_1, b_2, \ldots, b_{i-1}\}$, that is:

$$\pi_{i,B}^\parallel(x) = \sum_{j=1}^{i-1} \pi_{b_j^*}(x) = \sum_{j=1}^{i-1} \frac{\langle b_j^*, x \rangle}{\langle b_j^*, b_j^* \rangle} \cdot x.$$ 

Definition 2.22 (Orthogonal Projection) Given a lattice with basis $B$, we write $\pi_{i,B}^\perp(x)$, $1 \leq i \leq d$, to denote the orthogonal projection of $x$ onto the space spanned by the vectors $\{b_1, b_2, \ldots, b_{i-1}\}$, that is:

$$\pi_{i,B}^\perp(x) = x - \pi_{i,B}^\parallel(x) = x - \sum_{j=1}^{i-1} \frac{\langle b_j^*, x \rangle}{\langle b_j^*, b_j^* \rangle} \cdot x.$$ 

Note that $\pi_{1,B}^\perp(x)$ is the identity function, i.e. $\pi_{1,B}^\perp(x) = x$, and $\pi_{1,B}^\parallel(x) = 0$. Moreover, since the lattice basis is clear from context, we usually drop the basis $B$ and write $\pi_i^\perp$ or $\pi_i^\parallel$. Since the orthogonal projection is usually the projection of interest, we sometimes write $\pi_i^\perp := \pi_i$.

Example 2.8 Mapping $x = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$ to the point $y = (x_1, x_2, \ldots, x_{n-1}, 0)^T$ is a parallel projection onto the space spanned by the set of unit vectors $\{u_1, u_2, \ldots, u_{n-1}\}$. Such a projection can be represented by the matrix:

$$P = \begin{pmatrix} I_{n-1} & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{n \times n},$$

so that $y = Px$. The corresponding orthogonal projection onto the space spanned by the set of unit vectors $\{u_1, u_2, \ldots, u_{n-1}\}$, which takes $x = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$ to the point $y = (0, 0, \ldots, 0, x_n)^T$ can be represented by the matrix:

$$Q = \begin{pmatrix} 0 & 0 \\ 0 & I_1 \end{pmatrix} \in \mathbb{R}^{n \times n},$$

so that $y = Qx$.

We are particularly interested in projecting orthogonally to a subset of basis vectors for some lattice $L(B)$, which generates a projected sublattice:

$$\Lambda_i(B) = \{\pi_i^\perp(x) \mid x \in L(B)\},$$
we note that \( \Lambda_i(B) \) is indeed a sublattice of \( L(B) \), with a basis given by:

\[
\{ \pi_i^\perp(b_1), \pi_i^\perp(b_{i+1}), \ldots, \pi_i^\perp(b_d) \}.
\]

Note that \( \pi_i^\perp(B) = L(B) \) as is expected. By considering the projection function outlined in Definition 2.22, we note that we can write the Gram-Schmidt vectors as:

\[
b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^* = b_i - \sum_{j=1}^{i-1} \pi_b^*(b_j) = \pi_i^\perp(b_i).
\]

### 2.5 Hard Problems in Lattices

We are interested in several of the hard problems in lattices which underpin lattice-based cryptography. We first consider the Shortest Vector Problem which challenges us to find the shortest non-zero lattice vector in \( L(B) \), i.e. a vector \( x \in L(B) \) of length \( \|x\| = \lambda_1(L(B)) \).

**Definition 2.23 (Shortest Vector Problem (SVP))** Given a lattice basis \( B \), find a shortest non-zero lattice vector in \( L(B) \), i.e. a vector \( x \in L(B) \) of length \( \|x\| = \lambda_1(L(B)) \).

We can also define a selection of problems closely related to the shortest vector problem. The \( \gamma \)-Approximate Shortest Vector Problem gives us a bound \( \gamma \) and asks us to find a vector whose length is upper-bounded by \( \gamma \lambda_1(L(B)) \).

**Definition 2.24 (\( \gamma \)-Approximate Shortest Vector Problem (Approx-SVP))** Given a lattice basis \( B \) and an approximation factor \( \gamma \), find a non-zero lattice vector in \( x \in L(B) \) of length \( \|x\| \leq \gamma \lambda_1(L(B)) \).

The \( \gamma \)-unique Shortest Vector Problem (uSVP) provides a guarantee that a shortest vector exists, is unique (up to sign), and is significantly shorter (in particular, by a factor of \( \gamma \)) than the second shortest vector \( \lambda_2(L) \) in the lattice \( L \).
2.5 Hard Problems in Lattices

Definition 2.25 (\(\gamma\)-unique Shortest Vector Problem (uSVP)) Given a lattice basis \(B\) guaranteed to contain a non-zero shortest vector satisfying \(\lambda_2(L(B))/\lambda_1(L(B)) = \gamma\), find this shortest vector.

The Bounded Distance Decoding Problem (BDD) provides us with a public lattice basis \(L(B)\), as well as a target point \(t\), and a parameter \(\gamma\). Our goal is to find the closest lattice vector to the given target point, under the guarantee that the target point is very close to the lattice.

Definition 2.26 (Distance Between a Point and a Lattice) We define \(dist(t, L(B))\) to be the distance between the target point \(t\) and the closest point to \(t\) contained within the lattice \(L(B)\), that is:

\[
dist(t, L(B)) = \min\{\|x - t\| \mid x \in L(B)\}.
\]

Definition 2.27 (\(\gamma\)-Bounded Distance Decoding Problem (BDD\(\gamma\))) Given the basis of a lattice \(L(B)\), a vector \(t \notin L(B)\), and a parameter \(0 < \gamma\) such that \(dist(t, L(B)) < \gamma\lambda_1(B)\), find the lattice vector \(v \in L(B)\) which is closest to \(t\).

The Short Integer Solutions Problem (SIS) asks us to find a vector below a certain length, which is in the left kernel of a given, public, matrix.

Definition 2.28 (\(\gamma\)-Shortest Integer Solutions Problem (SIS)) Let \(A \in \mathbb{Z}_{q}^{(m \times n)}\) be a uniformly random matrix. Find a non-zero vector \(x \in \mathbb{Z}_{q}^{m}\) satisfying \(|x| \leq \gamma\) and \(xA \equiv 0 \mod q\).

We are interested in lattice-based cryptography which is built on the hardness of two families of problems: the NTRU problem and the Learning with Errors Problem.

Definition 2.29 (NTRU [HPS96]) Let \(n, q\) be positive integers, \(\phi \in \mathbb{Z}[x]\) be a monic polynomial of degree \(n\), and \(R_q = \mathbb{Z}[x]/(\phi)\). Let \(f, g \in R_q\), with \(f\) invertible, be small polynomials (i.e. having small coefficients) and \(h = g \cdot f^{-1} \in R_q\).
2.5 Hard Problems in Lattices

- Search-NTRU is the problem of recovering \( f \) or \( g \) given \( h \).
- Decision-NTRU is the problem of deciding if \( h = g \cdot f^{-1} \) or uniform.

Typically, the quotient polynomial is chosen as \( \phi = X^n + 1 \). However, other instantiations of the NTRU problem (such as NTRUPrime [BCLv19]) consider alternative quotient polynomials, such as \( \phi = X^n - X - 1 \). An important object is the matrix \( P_h \) as defined in Section 2.1, for a choice of \( \phi = X^n - 1 \) this matrix takes the form:

\[
P_h = \begin{pmatrix}
h_1 & h_2 & \ldots & h_n \\
h_n & h_1 & \ldots & h_{n-1} \\
\vdots & \vdots & \ddots & \vdots \\
h_2 & h_3 & \ldots & h_1
\end{pmatrix},
\]

where \( h_1, \ldots, h_n \) are the coefficients of the degree \((n - 1)\) polynomial \( h \in R_q \). In NTRU-based cryptographic constructions, the ring element \( h \) (and therefore the matrix \( P_h \)) is typically made public. Representing this matrix requires storage of only \( n \) integers modulo \( q \), i.e. \( n \log_2(q) \) bits. Note that, for other choices of \( \phi \), this matrix can take a different form, related to the structure of multiplication by \( x^i \).

We next define variants of the Learning with Errors problem. We begin by outlining the Module Learning with Errors Problem (Module-LWE), before defining the Ring Learning with Errors Problem (Ring-LWE) and the Learning with Errors Problem (LWE), which can both be viewed as special cases of Module-LWE.

**Definition 2.30 (Module Learning with Errors (Module-LWE) [LS15])** Let \( n, q, k \) be positive integers such that \( d = n/k \in \mathbb{Z} \). Define the rings \( R = \mathbb{Z}[X]/(X^d + 1) \) and \( R_q = R/qR \). Let \( \chi \) be a probability distribution on \( R \) and \( s \) be a secret module element in \( R^k_q \).

- We define the Module-LWE Distribution \( ML_{s,\chi,q} \) as the distribution on \( R^k_q \times R_q \) given by choosing \( a_i \in R^k_q \) uniformly at random, choosing \( e_i \in R \) according to \( \chi \) and considering it as an element of \( R_q \), and outputting \( (a_i, \langle a_i, s \rangle + e_i) \in R^k_q \times R_q \).
- Search-Module-LWE is the problem of recovering the ring element \( s \) from a collection \( \{ (a_i, \langle a_i, s \rangle + e_i) \}_{i=1}^m \) of samples drawn according to \( ML_{s,\chi,q} \).
2.5 Hard Problems in Lattices

- **Decision-Module-LWE** is the problem of distinguishing whether samples \( \{(a_i, \langle a_i, s \rangle + e_i)\}_{i=1}^m \) are drawn from the Module-LWE distribution \( ML_{s,\chi,q} \) or uniformly from \( R_q^k \times R_q \).

The distribution \( \chi \) is typically a discrete Gaussian distribution over \( \mathbb{Z} \), as defined in Definition 2.1, centred at zero and with width parameter \( \alpha q \). Here, the value \( \alpha \) is referred to as the LWE error rate, and we recall that a discrete Gaussian distribution with width parameter \( \alpha q \) has standard deviation \( \sigma = \frac{\alpha}{\sqrt{2\pi}} \).

Note that a module element \( \mathbf{v} \in R_q \) corresponds to a vector of \( k \) polynomials of degree \( d \). Therefore, by considering the co-efficient vectors of the module elements, and recalling that \( \mathbf{P}_a \) is the matrix representation of multiplication by the ring element \( a \), we can represent a single Module-LWE sample, consisting of the \( k \) ring elements \( a_1, a_2, \ldots, a_k \in R_q \), by the system of equations:

\[
\mathbf{b} = [\mathbf{P}_{a_1} | \mathbf{P}_{a_2} | \cdots | \mathbf{P}_{a_k}] \mathbf{s} + \mathbf{e}.
\]

More explicitly, for our choice of \( f = X^d + 1 \), we have that each \( \mathbf{P}_{a_j} \) is of the form:

\[
\mathbf{P}_{a_j} = \begin{pmatrix}
    a_1 & a_2 & a_3 & \cdots & a_{d-2} & a_{d-1} & a_d \\
    -a_n & a_1 & a_2 & \cdots & a_{d-3} & a_{d-2} & a_{d-1} \\
    -a_{n-1} & -a_n & a_1 & \cdots & a_{d-4} & a_{d-3} & a_{d-2} \\
    \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
    -a_3 & -a_4 & -a_5 & \cdots & -a_d & a_1 & a_2 \\
    -a_2 & -a_3 & -a_4 & \cdots & -a_{d-1} & -a_d & a_1 \\
\end{pmatrix},
\]

and, therefore, we have:

\[
\mathbf{b} = \begin{pmatrix}
    a_{1,1} & a_{1,2} & \cdots & a_{1,d} & \cdots & a_{k,1} & a_{k,2} & \cdots & a_{k,d} \\
    -a_{1,d} & a_{1,1} & \cdots & a_{1,d-1} & \cdots & -a_{k,d} & a_{k,1} & \cdots & a_{k,d-1} \\
    \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
    -a_{1,2} & -a_{1,3} & \cdots & a_{1,1} & \cdots & -a_{k,2} & -a_{k,3} & \cdots & a_{k,1} \\
\end{pmatrix} \begin{pmatrix}
    s_1 \\
    s_2 \\
    \vdots \\
    s_n \\
\end{pmatrix} + \begin{pmatrix}
    e_1 \\
    e_2 \\
    \vdots \\
    e_n \\
\end{pmatrix}.
\]

Representing the matrix \( \mathbf{A} = [\mathbf{P}_{a_1} | \mathbf{P}_{a_2} | \cdots | \mathbf{P}_{a_k}] \) requires storage of \( n = dk \) integers modulo \( q \), i.e. \( n \log_2(q) \) bits. Setting the module rank \( k = 1 \) gives rise to the **Ring Learning with Errors Problem**.
Definition 2.31 (Ring Learning with Errors (Ring-LWE) \cite{LPR10}) Define the rings 
\[ R = \mathbb{Z}[X]/(X^n + 1) \] and \( R_q = R/qR \). Let \( n, q \) be positive integers, \( \chi \) be a probability 
distribution on \( R \) and \( s \) be a secret ring element in \( R_q \).

- We define the Ring-LWE Distribution \( RL_{s,\chi,q} \) as the distribution on \( R_q \times R_q \) given by 
  choosing \( a_i \in R_q \) uniformly at random, choosing \( e_i \in R \) according to \( \chi \) and considering 
  it as an element of \( R_q \), and outputting \( (a_i, a_is + e_i) \in R_q \times R_q \).

- **Search-Ring-LWE** is the problem of recovering the ring element \( s \) from a collection 
  \( \{(a_i, a_is + e_i)\}_{i=1}^m \) of samples drawn according to \( RL_{s,\chi,q} \).

- **Decision-Ring-LWE** is the problem of distinguishing whether samples \( \{(a_i, a_is + e_i)\}_{i=1}^m \) 
  are drawn from the Ring-LWE distribution \( RL_{s,\chi,q} \) or uniformly from \( R_q^d \times R_q \).

In our definition of Ring-LWE we have considered the quotient ring \( R_q = \mathbb{Z}_q[X]/(X^n + 1) \) as 
a basis for the problem. In fact, we can define Ring-LWE with a variety of rings, although 
this can lead to security issues \cite{Pei16}. A single Ring-LWE sample consists of \( a_is + e_i \) where 
\( a_i, s, e_i \in R_q \) are all ring elements. As in Module-LWE, we can represent Ring-LWE samples 
in matrix/vector form:

\[
b = P_a s + e.\]

More explicitly, for our choice of \( f = X^n + 1 \), we have:

\[
b = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \\ -a_n & a_1 & \cdots & a_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ -a_2 & -a_3 & \cdots & a_1 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}.
\]

Representing the matrix \( P_a \) requires storage of \( n \) integers modulo \( q \), i.e. \( n \log_2(q) \) bits. Finally, 
if we consider the ring elements in \( R_q \) as vectors in \( \mathbb{Z}_q^n \), and ignore the algebraic structure 
induced by the ring \( R_q \), we arrive at the Learning with Errors problem.

Definition 2.32 (Learning with Errors (LWE) \cite{Reg05}) Let \( n, q \) be positive integers, 
\( \chi \) be a probability distribution on \( \mathbb{Z} \) and \( s \) be a secret vector in \( \mathbb{Z}_q^n \).
2.5 Hard Problems in Lattices

- We denote the LWE Distribution $L_{s,\chi,q}$ as the distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ given by choosing $a_i \in \mathbb{Z}_q^n$ uniformly at random, choosing $e_i \in \mathbb{Z}$ according to $\chi$ and considering it as an element of $\mathbb{Z}_q$, and outputting $(a_i, (a_i, s) + e_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.

- **Search-LWE** is the problem of recovering the vector $s$ from a collection $\{(a_i, (a_i, s) + e_i)\}_{i=1}^m$ of samples drawn according to $L_{s,\chi,q}$.

- **Decision-LWE** is the problem of distinguishing whether samples $\{(a_i, (a_i, s) + e_i)\}_{i=1}^m$ are drawn from the LWE distribution $L_{s,\chi,q}$ or uniformly from $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

If the components of the LWE secret follow the error distribution, then this is known as normal-form LWE. A single LWE sample consists of $\langle a_i, s \rangle + e_i$, where the $n$ coefficients of the vector $a_i$ are drawn uniformly at random from $\mathbb{Z}_q$, and the scalar $e_i$ is drawn from the error distribution $\chi$. We can represent $m$ LWE samples by the system of equations $b = As + e$, where:

$$
\begin{bmatrix}
    b_1 \\
    b_2 \\
    \vdots \\
    b_m
\end{bmatrix}
= 
\begin{bmatrix}
    a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\
    a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{m,1} & a_{m,2} & \cdots & a_{m,n}
\end{bmatrix}
\begin{bmatrix}
    s_1 \\
    s_2 \\
    \vdots \\
    s_n
\end{bmatrix}
+ 
\begin{bmatrix}
    e_1 \\
    e_2 \\
    \vdots \\
    e_m
\end{bmatrix}
$$

In LWE-based cryptographic constructions the matrix $A$ and the vector $b$ are typically made public, whereas the secret $s$ and the error $e$ remain private. To represent the matrix $A$ requires storage of $n \times m$ integers modulo $q$, i.e. $nm \log_2(q)$ bits.

To summarise, Module-LWE considers vectors of $k$ polynomials over the ring $\mathbb{Z}[X]/(X^{n/k} + 1)$. Setting the module rank $k = 1$, and considering elements of the ring $\mathbb{Z}[X]/(X^n + 1)$, gives rise to the Ring-LWE problem and, moreover, ignoring any structure induced by this ring, we can retrieve the Learning with Errors problem over $\mathbb{Z}_q^n$. Therefore, one can represent:

- a single Module-LWE sample by $k$ Ring-LWE samples of the appropriate dimension,
- a single Ring-LWE sample by $n$ LWE samples, and
- a single Module-LWE sample as $nk$ LWE samples.

Throughout this thesis, we will always view Ring-LWE and Module-LWE samples simply as LWE samples of the appropriate dimension. Based on current knowledge, this approach is...
reasonable since there are no known approaches to solve Ring-LWE and Module-LWE which exploits the additional algebraic structure in the ring.

### 2.5.1 Related Problems

There are a variety of problems closely related to the Learning with Errors problem. One variant of interest is the *Learning with Rounding Problem* [BPR12], which replaces the addition of a random error term $e_i$ with a deterministic rounding process. Informally, this deterministic rounding process splits the space $\mathbb{Z}_q$ into $p$ sections, each of size approximately $\frac{q}{p}$. Specifically, the operation:

$$\lfloor x \rfloor_p = \left\lfloor \frac{p}{q} x \right\rfloor,$$

is used.

**Definition 2.33 (Learning with Rounding (LWR) [BPR12])** Let $n, q, p < q$ be positive integers and $s$ be a secret vector in $\mathbb{Z}_q^n$. We define the LWR Distribution $L_{s,q,p}$ as the distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ given by choosing $a_i \in \mathbb{Z}_q^n$ uniformly at random and outputting $(a_i, \lfloor \langle a_i, s \rangle \rfloor_p) \in \mathbb{Z}_q^n \times \mathbb{Z}_p$.

- **Search-LWR** is the problem of recovering the vector $s$ from a collection $\{(a_i, \lfloor \langle a_i, s \rangle \rfloor_p)\}_{i=1}^m$ of samples drawn according to $L_{s,q,p}$.
- **Decision-LWR** is the problem of distinguishing whether samples $\{(a_i, \lfloor \langle a_i, s \rangle \rfloor_p)\}_{i=1}^m$ are drawn from the LWR distribution $L_{s,q,p}$ or uniformly from $\mathbb{Z}_q^n \times \mathbb{Z}_p$.

We note that we can similarly define the *Ring-Learning with Rounding Problem* (Ring-LWR), and the *Module-Learning with Rounding Problem* (Module-LWR) in a similar manner to LWR and LWE. We can also view Learning with Rounding samples as LWE samples, where the error term is generated according to the parameters used in the rounding process as outlined above:

$$\lfloor \langle a_i, s \rangle \rfloor_p = \langle a_i, s \rangle + e_i,$$

for an appropriately chosen error term $e_i$. In Table 2.1, we provide a comparison of the LWE, Ring-LWE, Module-LWE, LWR, and NTRU problems.
2.5 Hard Problems in Lattices

2.5.2 Small Secrets

In many cases, for efficiency purposes, it can be useful to restrict the secret distribution of $s$ from uniformly random over $\mathbb{Z}_n^q$ to some other distribution with a smaller support e.g. $\{0, 1\}^n$. Several typical examples can be found in Definition 2.34 following the notation of [Alb17].

**Definition 2.34 (Small Secret Distributions)** Let $n, q$ be positive integers.

- $B$ is any probability distribution on $\mathbb{Z}_q^n$ where each component is $\leq 1$ in absolute value.
- $B^+$ is the probability distribution on $\mathbb{Z}_q^n$ where each component is independently sampled uniformly at random from $\{0, 1\}$.
- $B^-$ is the probability distribution on $\mathbb{Z}_q^n$ where each component is independently sampled uniformly at random from $\{-1, 0, 1\}$.
- $B^+_h$ is the probability distribution on $\mathbb{Z}_q^n$ where components are sampled uniformly at random from $\{0, 1\}$ with the additional guarantee that at most $h$ components are non-zero.
- $B^-_h$ is the probability distribution on $\mathbb{Z}_q^n$ where components are sampled uniformly at random from $\{-1, 0, 1\}$ with the additional guarantee that exactly $h$ components are non-zero.

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Secret</th>
<th>Error</th>
<th>Public Coefficients</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTRU</td>
<td>$f \in \mathbb{R}_q$ invertible</td>
<td>$g \in \mathbb{R}_q$</td>
<td>$h \in \mathbb{R}_q$</td>
<td>$(h, g \cdot f^{-1}) \in \mathbb{R}_q \times \mathbb{R}_q$</td>
</tr>
<tr>
<td>LWE</td>
<td>$s \in \mathbb{Z}_q^n$</td>
<td>$e_i \in \mathbb{Z}$</td>
<td>$a_i \in \mathbb{Z}_q^n$</td>
<td>$(a_i, (a_i, s) + e_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$</td>
</tr>
<tr>
<td>Ring-LWE</td>
<td>$s \in \mathbb{R}_q$</td>
<td>$e_i \in \mathbb{R}$</td>
<td>$a_i \in \mathbb{R}_q$</td>
<td>$(a_i, a_i s + e_i) \in \mathbb{R}_q \times \mathbb{R}_q$</td>
</tr>
<tr>
<td>Module-LWE</td>
<td>$s \in \mathbb{R}_q^k$</td>
<td>$e_i \in \mathbb{R}$</td>
<td>$a_i \in \mathbb{R}_q^k$</td>
<td>$(a_i, (a_i, s) + e_i) \in (\mathbb{R}_q)^k \times \mathbb{R}_q$</td>
</tr>
<tr>
<td>LWR</td>
<td>$s \in \mathbb{Z}_q^n$</td>
<td>n/a</td>
<td>$a_i \in \mathbb{Z}_q^n$</td>
<td>$(a_i, [a_i, s]_p) \in \mathbb{Z}_q^n \times \mathbb{Z}_p$</td>
</tr>
<tr>
<td>Ring-LWR</td>
<td>$s \in \mathbb{R}_q$</td>
<td>n/a</td>
<td>$a_i \in \mathbb{R}_q$</td>
<td>$(a_i, [a_i, s]_p) \in \mathbb{R}_q \times \mathbb{R}_p$</td>
</tr>
<tr>
<td>Module-LWR</td>
<td>$s \in \mathbb{R}_q^k$</td>
<td>n/a</td>
<td>$a_i \in \mathbb{R}_q^k$</td>
<td>$(a_i, [a_i, s]_p) \in \mathbb{R}_q^k \times \mathbb{R}_p$</td>
</tr>
</tbody>
</table>

Table 2.1: A comparison of variants of the Learning with Errors, Learning with Rounding, and NTRU problems.
• $B_{(h_1,h_2)}$ is the probability distribution on $\mathbb{Z}_q^n$ where components are sampled uniformly at random from \{-1, 0, 1\} with the additional guarantee that exactly $h_1$ components are equal to $-1$ and exactly $h_2$ components are equal to 1.

We define the Small-secret Learning with Errors problem to be the same as in Definition 2.32 except the secret is drawn from one of the distributions defined in Definition 2.34. The hardness of Small-secret LWE has been studied in [BLP+13] and more recently in [Mic18]. The concrete loss of security which arises from the use of small secrets has been studied in [BGPW16, Alb17, ACD+18, Wun19, CP19]. Note that we can consider small-secrets in the context of any of the Module-LWE/R, Ring-LWE/R, or LWE/R problems.

### 2.6 SVP Solvers, CVP Solvers, and Lattice Reduction Algorithms

The security of Learning with Errors-based cryptosystems depends on the hardness of finding short vectors within $q$-ary lattices. There are two main families of algorithms used to find short vectors. The first are sieving algorithms [AKS01, LMvdP15, BDGL16], which use large lists of short vectors and iteratively generate new vectors by checking for shorter linear combinations within this list, requiring the use of exponential memory. The second are enumeration algorithms [Kan83, FP85, MW15], which perform a lattice-point search around a given target point and only require the usage of polynomial memory. There are a variety of estimates in the literature considering the running time of these SVP solvers, which are surveyed in Chapter 4.

Informally, a lattice reduction algorithm takes as input a public, ‘bad’, basis $B$ of some lattice $L$ and attempts to find a shorter and more orthogonal basis $B'$ of the same lattice. As an example, the BKZ algorithm [SE94] does this by calling an SVP solver on projected sublattices of dimension $\beta$ at a cost of:

$$T_{BKZ}(\beta, d) = c \cdot T_{SVP}(\beta).$$

Where $T_{SVP}(\beta)$ is the cost of an SVP solver in dimension $\beta$, and $c$ denotes the number of required calls to this SVP solver. The FPYLLL library [FPL20] contains implementations of the lattice reduction algorithms outlined in this section, and we will use this library throughout to provide examples.
2.6.1 LLL

The *Lenstra, Lenstra and Lovász (LLL) algorithm* \cite{LLL82} was originally designed as an algorithm to perform polynomial factorisation. This algorithm has since been used to perform lattice reduction and, indeed, is the first polynomial time lattice reduction algorithm. In a lattice of dimension \( d \), LLL finds an (approximate) shortest vector \( v \) whose length is at most \( \gamma \lambda_1(L) \) where \( \gamma = 2^{O(d)} \). Let \( L(B) \) be the lattice with basis matrix \([b_1 \mid b_2 \mid \cdots \mid b_d] \), GSO vectors \( B^* = \{b_1^*, b_2^*, \cdots, b_d^*\} \), and Gram-Schmidt coefficients \( \mu_{i,j} \).

**Definition 2.35 (LLL-reduced Basis)** We say that the basis \( B \) is LLL-reduced if the following two conditions are satisfied:

1. for \( 1 \leq j < i \leq d \), we have that \( |\mu_{i,j}| \leq \frac{1}{2} \), and
2. for \( t = 1, 2, \ldots, d-1 \), and some \( \kappa \in \left(\frac{1}{4}, 1\right] \), each pair of vectors indexed \((t, t+1)\) satisfies:
   \[
   \kappa \|b_t^*\|^2 \leq \|b_{t+1}^*\|^2 + \mu_{t+1,t}^2 \|b_t\|^2.
   \]

The first condition (size reduction) ensures that the lengths of the GSO vectors are in descending order i.e. \( \|b_t\| \geq \|b_{t+1}^*\| \). The second (Lovász) condition can be re-arranged to:

\[
(\kappa - \mu_{t+1,t}^2)\|b_t^*\|^2 \leq \|b_{t+1}^*\|^2,
\]

and by the first condition we have that \( |\mu_{t,t+1}| \leq \frac{1}{2} \), which gives us \( \mu_{t+1,t}^2 \leq \frac{1}{4} \), and, since \( \kappa \in \left(\frac{1}{4}, 1\right] \), this leads us to \((\kappa - \mu_{t+1,t}^2) \in \left(0, \frac{3}{4}\right]\). This provides a bound on the difference in length between successive Gram-Schmidt vectors, i.e. \( b_t^* \) and \( b_{t+1}^* \), for \( 1 \leq t \leq d-1 \). Note that we can write:

\[
\|b_t^*\|^2 \leq \frac{1}{(\kappa - \mu_{t+1,t}^2)}\|b_{t-1}^*\|^2 \leq \frac{1}{(\kappa - \frac{3}{4})}\|b_{t-1}^*\|^2,
\]

and, choosing e.g. \( \kappa = \frac{1}{2} \), we can note the relation:

\[
\|b_t^*\|^2 \leq 4\|b_{t-1}^*\|^2.
\]

Intuitively, LLL works in the following way: we perform size reduction, i.e. iteratively compute:

\[
b_t \leftarrow b_t - \sum_{j=1}^{t-1} |\mu_{t,j}|b_j,
\]
and then we check that the Lovász condition is satisfied. If this condition is not satisfied at e.g. index $\ell$, we swap the position of the vector $b_\ell$ with the previous vector $b_{\ell-1}$ and restart the process. We outline the LLL algorithm in Algorithm 1.

**Algorithm 1: The LLL Algorithm.**

We discuss heuristics related to the LLL algorithm, such as the expected shape of the output basis, in Section 2.8.

### 2.6.1.1 Performing LLL in FPYLLL

We can use the FPYLLL library [FPL20] to perform LLL reduction in practice. Let us consider a $q$-ary lattice of dimension $d = 220$, with $q = 2^{15}$. This lattice has basis matrix given by:

$$B = \begin{pmatrix} I_{110} & 0 \\ A & 2^{15} \cdot I_{110} \end{pmatrix},$$

where the components of $A$ are chosen uniformly at random from $\mathbb{Z}_q$. We perform LLL-reduction on such a lattice basis using the SageMath code in Figure 2.4 which generates an output GSO Profile as in Figure 2.5. Clearly, by considering Algorithm 1, we expect that the lengths of the GSO vectors will decrease as the index $i$ increases, i.e. $\|b_i\| \geq \|b_{i+1}\|$ for $1 \leq i < d$. 
2.6 SVP Solvers, CVP Solvers, and Lattice Reduction Algorithms

```python
from fpylll import *

A = IntegerMatrix.random(220, "qary", k=110, q=2**15)[::1]
A = LLL.reduction(A)
M = GSO.Mat(A)
M.update_gso()
N = []

for i in range(len(M.r())):
    N.append((i, log(M.r()[i], 2**15).n()))

for i in N:
    print(i)
```

Figure 2.4: Code used to generate a random $q$-ary lattice of dimension 220 with modulus $q = 2^{15}$ and determinant $q^{110}$, and perform LLL-reduction.

![Figure 2.4: Code used to generate a random q-ary lattice of dimension 220 with modulus q = 2^{15} and determinant q^{110}, and perform LLL-reduction.](image)

Figure 2.5: GSO lengths of a random $q$-ary lattice of dimension 220 with modulus $q = 2^{15}$ and determinant $q^{110}$ which has been LLL-reduced using FPYLLL [FPL20].
2.6 SVP Solvers, CVP Solvers, and Lattice Reduction Algorithms

2.6.2 BKZ

For the majority of this thesis the lattice reduction algorithm under consideration will be the Block Korkine Zolotarev algorithm (BKZ) \cite{CN11}. The high level idea of BKZ is to produce a shorter lattice basis by finding the short vectors in projected sublattices of dimension $\beta$, referred to as the blocksize, as a subroutine. Let $L(B)$ be the lattice with basis matrix $B = [b_1 | b_2 | \cdots | b_n]$, and corresponding GSO vectors $B^* = \{b_1^*, b_2^*, \cdots, b_n^*\}$.

**Definition 2.36 (HKZ-reduced Basis)** We say that the basis $B$ is HKZ-reduced if the following two conditions are satisfied:

1. The lattice basis is size reduced, and
2. for $i \in \{1, \ldots, d\}$ we have that $b_i = \lambda_1(L(B))$.

Recall that the function $\pi_1(\cdot)$ is the identity function, meaning that for $i = 1$ the second condition states that $b_1 = \lambda_1(L(B))$. We write $B_{[i,j]}$ to denote the projected sublattice with basis matrix:

$$[\pi_i(b_i) | \pi_i(b_{i+1}) | \cdots | \pi_i(b_j)].$$

**Definition 2.37 (BKZ-$\beta$-reduced Basis)** The lattice basis $B$ is BKZ-$\beta$-reduced, for some $\beta \geq 2$, if the following two conditions are satisfied:

1. The lattice basis is size reduced, and
2. For $i \in \{1, \ldots, d\}$ we have that $b_i = \lambda_1(L(B_{[i,\min(i+\beta-1,d)]}))$.

The BKZ algorithm is complex, and can be difficult to present concisely. We provide an intuition as to how BKZ works, and point to the literature \cite{CN11} for full details. In this thesis use BKZ as a black-box: an algorithm which, given as input a lattice basis, provides us with another lattice basis which satisfies certain properties. These properties can then be used to produce running times for cryptanalytic attacks. Informally, the BKZ algorithm executes the following set of steps:
2.6 SVP Solvers, CVP Solvers, and Lattice Reduction Algorithms

1. The process begins by considering the first “block” of $\beta$ vectors $\mathbf{b}_i, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_{i+\beta-1}$ for $i = 1$. We find a new, shorter, vector $\mathbf{b}_\psi$, which is a linear combination of the vectors under consideration, by solving SVP in this “block”.

2. We then insert this vector $\mathbf{b}_\psi$ into the lattice basis for $L(\mathbf{B})$ (in position $i$), and, since we now have a set of $d+1$ vectors in $d$-dimensional space, we use LLL to remove the resulting linear dependency, that is, we remove the resulting “zero” vector to retrieve a new basis $\mathbf{B}''$ for the full lattice $L$.

3. We increase the index $i$, iterating through the set $i \in \{1, 2, \ldots, d - \beta + 1\}$.

4. We repeat this process $c$ times, halting when either (a) an upper bound on $c$ is achieved, or (b) the output basis satisfies a set of pre-defined conditions (which may include, for example, halting when the quality of the basis is no longer improving at a predetermined rate).

Each cycle through the index set $i \in \{1, 2, \ldots, d - \beta + 1\}$ is referred to as a tour. We note that the number of tours considered inside BKZ affects both the running time and the output quality.

2.6.2.1 Performing BKZ in FPYLLL

We can use the FPYLLL library [FPL20] to perform BKZ reduction for a variety of blocksizes $\beta$. Using the code outlined in Figure 2.6, we can generate GSO profiles for a random $q$-ary matrix as in Section 2.6.1.1 i.e. with $d = 220, q = 2^{15}$, and $m = n = 110$. This code generates basis profiles using blocksizes $\beta \in \{10, 30, 50, 70\}$, and the output can be seen in Figure 2.7. In a similar manner to Figure 2.5, we expect the lengths of the GSO vectors to be decreasing, i.e. $\|\mathbf{b}^*_i\| \geq \|\mathbf{b}^*_{i+1}\|$ for $1 \leq i < d$. 

50
from fpylll import *
from fpylll.algorithms.bkz2 import BKZReduction as BKZ2

FPLLL.set_precision(128)
A = IntegerMatrix.random(220, "qary", k=110, q=2**15)[:,:-1]
M = GSO.Mat(A, float_type = "mpfr")
M.update_gso()
params = BKZ.EasyParam(block_size = 50, flags=BKZ.VERBOSE)
bkz = BKZ2(M)
bkz(params)
N = []

for i in range(len(M.r())):
    M.append((i, log(M.r()[i],2**15).n()))

for i in N:
    print(i)

Figure 2.6: Code used to generate the BKZ-50 reduced lattice basis considered in Figure 2.7
Figure 2.7: Output GSO lengths of a random $q$-ary lattice of dimension 220 with modulus $q = 2^{15}$ and determinant $q^{110}$ which has been BKZ-$\beta$ reduced for $\beta \in \{10, 30, 50, 70\}$ using FPYLLL [FPL20]. For comparison, we also include the LLL basis profile used in Figure 2.5.
2.7 Babai’s Nearest Plane Algorithm

Babai’s Nearest Plane algorithm \cite{Bab86} is a polynomial time algorithm which (approximately) solves the CVP problem. That is, given a lattice $L(B)$ with $B \in \mathbb{R}^{d \times n}$ and target point $t \in \mathbb{R}^d$, Babai’s Nearest Plane algorithm outputs a vector $v \in L(B)$ which satisfies:

$$\|v - t\| \leq 2^{\frac{n}{2}} \text{dist}(t, L(B)).$$

We describe Babai’s Nearest Plane algorithm in Algorithm 2.

\begin{algorithm}
\KwIn{A basis for the lattice $L(B)$ and a target point $t \in \mathbb{R}^m$
\KwResult{A vector $v$ which satisfies $\|v - t\| \leq 2^{\frac{n}{2}} \text{dist}(t, L(B))$
Run LLL on $B$ with $\delta = \frac{3}{4}$. Set $b = t$
for $i = n$ to 1 do
    $b = b - \left\lfloor \frac{(b, b^*_i)}{\langle b^*_i, b^*_i \rangle} \right\rfloor b_i$
end
Output $t - b$

\textbf{Algorithm 2:} Babai’s Nearest Plane algorithm.
\end{algorithm}

If we consider the lattice $L(B)$ tiled by the shifted fundamental parallelepiped defined in Definition 2.16 then, for a given target vector $t$, Babai’s Nearest Plane algorithm returns the lattice point which is contained in the same fundamental region of the target vector $t$ (here, we assume that the parallelepiped is centred on the vector $t$). Although Babai’s Nearest Plane algorithm is deterministic, in the case of decoding attacks on NTRU and LWE, we can define a success probability with respect to the target point $t$ \cite{LN13}, and we outline approaches for computing this probability in Chapter 3.

2.8 Cryptanalytic Heuristics

In order to provide concrete security estimates for lattice-based cryptosystems, we use the running times of the best attacks. To retrieve these running times, we require usage of many heuristics. Throughout this section we outline the cryptanalytic heuristics used in this thesis. We begin with the root-Hermite factor, a quantity which captures the quality of a lattice basis after lattice reduction has been performed.

\begin{definition}[Root-Hermite Factor] For a basis $B$ of a lattice $L$ of dimension $d$, the...
2.8 Cryptanalytic Heuristics

**root-Hermite factor** is defined to be:

\[
\delta = \left( \frac{\|b_1\|}{\text{Vol}(L)^{1/d}} \right)^{1/d}.
\]

For bases produced by the BKZ algorithm with \( \beta \geq 40 \), this value is well approximated by

\[
\delta^{2(\beta-1)} = \frac{\beta}{2\pi e} (\beta \pi)^{1/\beta} \quad [\text{Che13}].
\]

As the value of \( \beta \) increases, the value \( \delta \) decreases towards one.

**Example 2.9** Consider the BKZ-reduced lattice bases in Figure 2.7. In Table 2.2, for each value of the blocksize \( \beta \), we compute the root-Hermite Factor \( \delta \) and also calculate the corresponding approximation \( \frac{\beta}{2\pi e} (\beta \pi)^{1/\beta} \).

<table>
<thead>
<tr>
<th>( \beta )</th>
<th>( |b_1| )</th>
<th>( \delta )</th>
<th>( \frac{\beta}{2\pi e} (\beta \pi)^{1/\beta} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5616.891</td>
<td>1.01574</td>
<td>0.98947</td>
</tr>
<tr>
<td>30</td>
<td>3486.508</td>
<td>1.01353</td>
<td>1.01240</td>
</tr>
<tr>
<td>50</td>
<td>2501.535</td>
<td>1.01201</td>
<td>1.01206</td>
</tr>
<tr>
<td>70</td>
<td>1944.753</td>
<td>1.01085</td>
<td>1.01084</td>
</tr>
</tbody>
</table>

Table 2.2: The root-Hermite factor for a random \( q \)-ary lattice of dimension 220 with modulus \( q = 2^{15} \) and determinant \( q^{110} \) which has been BKZ-\( \beta \) reduced for \( \beta \in \{10, 30, 50, 70\} \).

The **Geometric Series Assumption** gives a prediction for the output shape of the Gram-Schmidt basis vectors.

**Definition 2.39 (Geometric Series Assumption [Sch03])** Let \( \{b_1, b_2, \ldots, b_d\} \) be a basis, of quality \( \delta \), of a lattice \( L \), output by some lattice reduction algorithm. The **Geometric Series Assumption** states that the lengths \( \|b_i^*\| \) for \( 1 \leq i \leq d \) of the Gram-Schmidt vectors of this basis are approximated by:

\[
\|b_i^*\| \approx \psi^{i-1} \|b_1\|,
\]

for some \( 0 < \psi < 1 \).

We consider the formula for the volume of a lattice, as defined in Definition 2.17, in order to
determine the value of $\psi$:

$$
\text{Vol}(L)^{1/d} = \prod_{i=1}^{d} \|b_i^*\|
= \left( \prod_{i=1}^{d} \psi^i \|b_i\| \right)^{1/d}
= \left( \psi^{\frac{1}{2}d(d-1)} \|b_1\| \right)^{1/d}
= \psi^{\frac{1}{2}(d-1)} \|b_1\|
= \sqrt{\psi} d^{-1} \|b_1\|.
$$

Making use of $\|b_1\| = \delta^d \text{Vol}(L)^{1/d}$, we can conclude that $\delta^{-d} = \sqrt{\psi} d^{-1}$. This leads to $\psi = \delta^{-2d/(d+1)}$. Typically, this value is approximated as $\psi \approx \delta^{-2}$ and therefore we have:

$$
\|b_i^*\| = \psi^{i-1} \|b_1\|
\approx (\delta^{-2})^{i-1} \|b_1\|
= \delta^{-2i+2} \delta^d \text{Vol}(L)^{1/d}
= \delta^{-2i+2} d \text{Vol}(L)^{1/d}.
$$

We note that, for certain blocksizes $\beta$ (e.g. $\beta \geq 40$ [APS15]), the GSA appears to predict the output shape of lattice reduction well. Wunderer [Wun19] suggests a ‘q-ary’ GSA, which takes into consideration the case where the length of the first GSO vector is shorter than predicted by the GSA (in this case, since the vector is already shorter than predicted by the GSA, and lattice reduction will not increase the length of the input vectors, the GSA does not fit well). There are also a variety of BKZ simulators [CN11, BSW18, FPL20], which attempt to capture the concavity (i.e. non-linearity) of the final block of GSO vectors [YD17]. This non-linearity can be observed in Figure 2.7. We discuss both the ‘q-ary’ GSA and BKZ simulators in more detail in Chapter 3.

The Gaussian Heuristic gives us a simple method of predicting how many lattice points are within a “regular” region of a given volume. We can then use this heuristic to predict the length of the shortest vector within a lattice.

**Definition 2.40 (Gaussian Heuristic)** Let $L$ be a $d$-dimensional lattice. The Gaussian Heuristic states that if $K$ is a measurable set satisfying $K \subseteq \text{Span}(L)$, then the number of lattice points in $K \cap L$ satisfies:

$$
|K \cap L| \approx \frac{\text{Vol}(K)}{\text{Vol}(L)}.
$$
We can use the Gaussian Heuristic to predict the length of the shortest vector, \( \lambda_1(L) \), in a random lattice \( L \). To do this we suppose that \( K \) is an \( n \) dimensional ball, centred at the origin, with radius \( R = \lambda_1(L) \). Clearly, this means that the set \( K \cap L \) contains only the shortest vector in the lattice, i.e. we have \(|K \cap L| = 1\), yielding \( \text{Vol}(K) = \text{Vol}(L) \). Using the well-known formula for the volume of a \( n \)-sphere of radius \( R \), this leads us to:

\[
\frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2} + 1\right)} \lambda_1(L)^d \approx \text{Vol}(L).
\]

Re-arranging leads to the equation for an approximation of the length of the shortest vector in \( L \):

\[
\lambda_1(L) \approx \text{Vol}(L)^{1/d} \frac{\Gamma\left(\frac{d}{2} + 1\right)^{1/d}}{\sqrt{\pi}}.
\]

Assuming that Equation 2.1 is in fact an equality, and setting \( V_d(1) = \frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2} + 1\right)} \), we can take logarithms on both sides to produce:

\[
\log(\lambda_1(L)) = \log\left(\frac{\text{Vol}(L)^{1/d}}{V_d(1)^{1/d}}\right)
\]

\[
= \frac{1}{d} \log(\text{Vol}(L)) - \log(V_d(1))
\]

\[
= \frac{1}{d} \log\left(\prod_{i=1}^{d} \|b_i^*\|\right) - \log\left(\frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2} + 1\right)}\right)
\]

\[
= \frac{1}{d} \log\left(\prod_{i=1}^{d} \|b_i^*\|\right) - \frac{d}{2} \log(\pi) - \log\left(\Gamma\left(\frac{d}{2} + 1\right)\right)
\]

\[
= \frac{1}{d} \sum_{i=1}^{d} \log(\|b_i^*\|) - \frac{d}{2} \log(\pi) - \log\left(\Gamma\left(\frac{d}{2} + 1\right)\right).
\]

In Chapter 3 we will use Equation 2.2 to predict the length of the shortest vector in a variety of different lattices, in order to guarantee success for a specific attack technique.

## 2.9 Solving the Learning with Errors Problem

Throughout this thesis, we are interested in considering techniques used to solve variants of the Learning with Errors problem. In particular, in this section, we outline the dual, \( uSVP \), and \( decoding \) attacks. All three of these attacks require the usage of a lattice reduction algorithm, typically BKZ, parametrised by a blocksize \( \beta \). Variants of these attacks specialised to the presence of a small secret are discussed in Chapter 4.
2.9 Solving the Learning with Errors Problem

2.9.1 The Dual Attack

The Dual (or Distinguishing) attack [MR09, Alb17] is a method of solving Decision-LWE. Specifically, this problem distinguishes between LWE samples of the form \( \mathbf{b} = \mathbf{A}s + \mathbf{e} \), as defined in Definition 2.32, and uniformly random input. This method solves Decision-LWE via finding short vectors in the \( q \)-ary lattice considered in the SIS problem outlined in Section 2.5:

\[
L^* = \{ \mathbf{x} \in \mathbb{Z}^m \mid \mathbf{xA} \equiv \mathbf{0} \mod q \}.
\]

Upon retrieving a short vector \( \mathbf{v} \in L^* \), we then consider the inner product \( \langle \mathbf{v}, \mathbf{b} \rangle \). We have:

\[
\langle \mathbf{v}, \mathbf{b} \rangle = \langle \mathbf{v}, \mathbf{A}s + \mathbf{e} \rangle = \langle \mathbf{vA}, \mathbf{s} \rangle + \langle \mathbf{v}, \mathbf{e} \rangle = \langle \mathbf{v}, \mathbf{e} \rangle \mod q,
\]

since \( \mathbf{vA} \equiv \mathbf{0} \mod q \). If \( \mathbf{b} \) is formed of LWE samples, then the inner product \( \langle \mathbf{v}, \mathbf{e} \rangle \) is small [ACF+15]. If \( \mathbf{b} \) is uniformly random modulo \( q \), then we would expect the inner product \( \langle \mathbf{v}, \mathbf{e} \rangle \) to also be uniformly random. This observation can be used to distinguish LWE from random and thus solve Decision-LWE. We note that the success probability of this attack depends on the length of the vector \( \mathbf{v} \), and thus the BKZ blocksize \( \beta \). This can be seen by considering the output distribution of the inner products \( \langle \mathbf{v}, \mathbf{b} \rangle \): as the length of \( \mathbf{v} \) grows, this distribution becomes closer to uniformly random. In particular, the result from [LP11] gives the distinguishing advantage as close to:

\[
\exp(-\pi(\|\mathbf{v}\|\alpha)^2),
\]

where \( \alpha \) is the LWE error rate, as defined in Definition 2.30 i.e. \( \alpha = \frac{\sigma\sqrt{2\pi}}{q} \).

2.9.2 The uSVP Attack

The uSVP attack [Kan87, ADPS16, AGVW17] is a method of solving Search-LWE via the \( \gamma \)-unique shortest vector problem (uSVP). This approach consists of embedding the LWE error vector \( \mathbf{e} \) into a lattice \( L \) with a uSVP structure. It is known that the recovery of this vector via lattice reduction can be guaranteed under certain conditions [ADPS10]. In more detail, to solve LWE via uSVP we construct the \( q \)-ary lattice:

\[
L_{\text{uSVP}} = \{ \mathbf{y} \in \mathbb{Z}^m \mid \mathbf{y} \equiv \mathbf{Ax} \mod q \text{ for some } \mathbf{x} \in \mathbb{Z}^n \},
\]

which has basis matrix:

\[
\mathbf{B} = \begin{pmatrix}
\mathbf{A} & \mathbf{b} \\
0 & t
\end{pmatrix}.
\]
2.9 Solving the Learning with Errors Problem

Here $t$ is the embedding factor which denotes the distance between $b$ and the lattice $L(A)$, and this value is typically set to be e.g. $t = 1$ or $t = \|e\|$ \footnote{Although a choice of $t = \|e\|$ makes this problem more balanced, in practice, we typically choose $t = 1$ for efficiency, see [AFG13].}. We can see that:

\[
\begin{pmatrix}
A & b \\
0 & t
\end{pmatrix}
\begin{pmatrix}
-s \\
1
\end{pmatrix}
= \begin{pmatrix}
-As + b \\
t
\end{pmatrix}
\mod q,
\]

and, setting $t = 1$, we see that the lattice $L(B)$ contains the short vector $(e, 1)$. To recover this short vector, we choose the BKZ blocksize $\beta$ which satisfies the success condition from [ADPS16] (and this success condition was experimentally verified in [AGVW17]). Specifically, we choose $\beta$ to satisfy the [ADPS16] success condition:

\[
\sqrt{\frac{\beta}{d}} \cdot \lambda_1(L) \leq \delta^{2\beta - d} \cdot \text{Vol}(L)^{1/d},
\]

where $d$ is the dimension of the lattice $L(B)$. The overall cost of this approach, constrained by the [ADPS16] success condition, is:

\[
T_{uSVP} = \min_{\beta, d} \left\{ T_{BKZ}(\beta, d) \right\}.
\]

2.9.3 The Decoding Attack

The decoding attack [LP11, LN13, BG14] is a method of solving Search-LWE via the bounded distance decoding problem (BDD). This approach consists of constructing a lattice in which the LWE error vector $e$ is the offset between this lattice and a known target point. Specifically, we construct a lattice with basis matrix:

\[
B = \begin{pmatrix}
A & qI_m
\end{pmatrix},
\]

and we note that:

\[
\begin{pmatrix}
A & qI_m
\end{pmatrix}
\begin{pmatrix}
s \\
*
\end{pmatrix}
= As + q* = As \mod q.
\]

This tells us that the vector $As$ is contained in the lattice $L(B)$. Moreover, since $As = b - e$, we have that the publicly known point $b$ is of distance $\|e\|$ from the lattice point $As \mod q$. Since the vector $e$ is short, we can use a BDD solver to recover the lattice vector $As \mod q$ and, therefore, the LWE error vector $e$ (since $b$ is publicly known). Overall, the cost of this approach is:

\[
T_{Dec} = \min_{\beta, d} \left\{ \frac{T_{BKZ}(\beta, d) + T_{BDD}(\beta, d)}{p_{BDD}} \right\},
\]
where $T_{\text{BDD}}(\beta, d)$ is the cost of running the chosen BDD solver on a BKZ-$\beta$ reduced lattice of dimension $d$, and $p_{\text{BDD}}$ is the associated success probability of the BDD solver.

### 2.9.4 Alternative Techniques

There are many other methods to solve LWE, such as Coded-BKW [GJS15], Meet-in-the-middle [APS15, CHHS19], and Arora-Ge [AG11, ACFP14]. A summary of these techniques can be found in [APS15, Pla19]. For the parameter sets we are interested in, these techniques are not competitive with the uSVP, dual, and decoding attacks, as noted in [ACD+18], and we therefore do not consider them in this thesis.

### 2.10 The Learning with Errors Estimator

The Learning with Errors Estimator (LWE Estimator [APS15]) outputs concrete security estimates for a given set of LWE parameters $(n, \alpha, q)$. By default, the running times of the uSVP, decoding, and dual approaches are given as output. Estimates for several other techniques are also available, although we will not consider them in this thesis. In this section we outline the LWE Estimator.

#### 2.10.1 Input and Output

There are many input and output options in the LWE Estimator. In Table 2.3 we outline the input variables to the top level function `estimate_lwe`, which is used to retrieve security estimates.
### 2.10 The Learning with Errors Estimator

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>The LWE dimension</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>The LWE error rate</td>
</tr>
<tr>
<td>( q )</td>
<td>The LWE modulus</td>
</tr>
<tr>
<td>( m )</td>
<td>The number of LWE samples available</td>
</tr>
<tr>
<td>secret_distribution</td>
<td>The LWE secret distribution</td>
</tr>
<tr>
<td>reduction_cost_model</td>
<td>The cost model considered for BKZ</td>
</tr>
<tr>
<td>skip</td>
<td>List of attacks not considered in the output</td>
</tr>
</tbody>
</table>

Table 2.3: Input parameters to the LWE Estimator, used to retrieve security estimates for LWE parameter sets.

The two most interesting inputs are `secret_distribution` and `reduction_cost_model`. These enable the user to specify a specific LWE secret distribution (such as \( \{0,1\}^n \) or \( D_\sigma \)), as well as a custom cost model for lattice reduction.

#### 2.10.1.1 The LWE Secret Distribution: secret_distribution

By default, the secret distribution used in the estimator is \( D_\sigma \), where \( \sigma \) is the width of the error distribution, i.e. the LWE Estimator assumes *normal form LWE* by default. Available inputs are:

- **True**: each component is chosen uniformly at random from \( D_\sigma \) (default)
- **False**: each component is chosen uniformly at random from \( \mathbb{Z}_q \)
- \((a,b)\) : each component is chosen from the uniform distribution between \( a \) and \( b \)
- \((a,b),h\) : vectors with a fixed Hamming weight \( h \), where the non-zero components are chosen from the uniform distribution between \( a \) and \( b \)

It should be noted that, whilst there are small-secret variants of the uSVP, dual, and decoding attacks (outlined in Chapter 4), the LWE estimator does not trigger these variants for *all* small-secret distributions. It is therefore important to consult the source code [Est20] to determine the attack strategies being used.
There are four in-built cost models for the BKZ algorithm in the LWE Estimator: `sieve`, `qsieve`, `enum`, and `lp`. Each of those cost models corresponds to a result in the literature. The `lp` model corresponds to the runtime estimation of Linder and Peikert [LP11], which is known to be outdated, as discussed in [Alb17]. The `enum` model corresponds to the recent result of [ABF+20], which represents state-of-the-art enumeration-based lattice reduction. The `sieve` model corresponds to results in [BDGL16], which applies improvements in nearest-neighbour algorithms to sieving for short vectors to retrieve a complexity of $2^{0.292n}$ in dimension $n$, and the model used here assumes $8d$ calls to such an SVP oracle for a dimension $d$ lattice. The `qsieve` result corresponds to the results in [LMv14], which considers the application of quantum search techniques to sieving, resulting in a complexity of $2^{0.265n}$ in dimension $n$, and the estimator again assumes $8d$ calls for lattices of dimension $d$. These cost models correspond to the following formulae:

- `sieve`: $8d \cdot 2^{0.292\beta+16.4}$,
- `qsieve`: $8d \cdot 2^{0.265\beta+16.4}$,
- `enum`: \[
\begin{cases}
20.1839\beta \log(\beta) - 0.995\beta + 16.25 + \log(100) & \text{if } \frac{3}{2}\beta \geq d \text{ or } \beta \leq 92 \\
20.125\beta \log(\beta) - 0.547\beta + 10.4 + \log(100) & \text{otherwise}
\end{cases}
\]

and
- `lp`: $2^{\frac{1.8}{\log(100)} - 110 + \log(2.3 \cdot 10^9)}$.

Note that the model `enum` is considered as the default option in the current variant of the Estimator [Est20]. When retrieving security estimates, it may be desirable to use a custom cost model. We consider how to define the cost models:

- `example-model-1`: $T_{BKZ}(\beta, d) = 2^{0.292\beta}$, and
- `example-model-2`: $T_{BKZ}(\beta, d) = 4d \cdot 2^{0.265\beta+16.4}$,

in Figure 2.8.

```python
element-model-1 = lambda beta, d, B: ZZ(2)**RR(0.292*beta)
element-model-2 = lambda beta, d, B: ZZ(2)**RR(0.265*beta + 16.4 + log(4*d, 2))
```

Figure 2.8: An example of two custom cost models to be used in the LWE Estimator.

To retrieve an estimate with the cost model $T_{BKZ}(\beta, d) = 2^{0.292\beta}$, we call the top-level function `estimate_lwe` with the input `reduction_cost_model = example-model-1`. Such

\footnote{As of commit 8daf3f7, this model replaces $2^{0.270\beta \log(\beta) - 1.019\beta + 16.103 + \log(100)}$.}
2.10 The Learning with Errors Estimator

load("https://bitbucket.org/malb/lwe-estimator/raw/HEAD/estimator.py")

n = 256
alpha = 0.002
q = 17500
r = estimate_lwe(n, alpha, q)

Figure 2.9: Example call to the LWE Estimator with our example parameter set $n = 256$, $\alpha = 0.002$, and $q = 17500$.

a functionality allows for a custom cost models to be considered when computing security estimates. In Chapter 4, we consider further lattice reduction cost models used in the literature.

2.10.1.3 Outputs

When the LWE Estimator outputs security estimates, it also provides optimal attack parameters. To help illustrate the output of the LWE Estimator, we consider an example.

**Example 2.10** We call the top-level function `estimate_lwe` for the parameters $n = 256$, $\alpha = 0.002$ and $q = 17500$ in Figure 2.9.

This code will return complexity estimates and optimal attack parameters for the uSVP, decoding, and dual attacks. For the uSVP attack, the output is:

\[
\begin{align*}
rop & \approx 2^{159.5}, \quad red \approx 2^{159.5}, \quad \delta_0 : 1.005093, \quad \beta : 278, \quad d : 677, \quad m : 420,
\end{align*}
\]

for the decoding attack, the output is:

\[
\begin{align*}
rop & \approx 2^{171.2}, \quad m : 438, \quad red \approx 2^{171.2}, \quad \delta_0 : 1.005181, \quad \beta : 271, \quad d : 694, \quad babai : \approx 2^{157.0} \\
& \quad babai\_op:2^{172.1}, \quad repeat:2^{16.2}, \quad \epsilon : 2^{-14.0}
\end{align*}
\]

and for the dual attack, the output is:

\[
\begin{align*}
rop & \approx 2^{180.8}, \quad m : 471, \quad red \approx 2^{180.8}, \quad \delta_0 : 1.004745, \quad \beta : 310, \quad repeat:2^{141.0}, \quad d : 727, \quad c : 1
\end{align*}
\]

In Table 2.4, we explain each of the output values for the usvp, dual, and decoding attacks.
<table>
<thead>
<tr>
<th>Quantity</th>
<th>Attack/s</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>rop</td>
<td>all</td>
<td>The cost of the attack in ring operations (in the ring $\mathbb{Z}_q$)</td>
</tr>
<tr>
<td>$\beta$</td>
<td>all</td>
<td>The optimal blocksize used inside BKZ</td>
</tr>
<tr>
<td>$\delta_0$</td>
<td>all</td>
<td>The root-Hermite factor associated to the choice of $\beta^6$</td>
</tr>
<tr>
<td>$m$</td>
<td>all</td>
<td>The optimal number of LWE samples used in the attack</td>
</tr>
<tr>
<td>$d$</td>
<td>all</td>
<td>The dimension of the lattice reduced via BKZ</td>
</tr>
<tr>
<td>red</td>
<td>all</td>
<td>The (total) cost of lattice reduction</td>
</tr>
<tr>
<td>repeat</td>
<td>dual, dec</td>
<td>The number of times the attack needs to be repeated</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>dec</td>
<td>Overall success probability of the algorithm</td>
</tr>
<tr>
<td>babai</td>
<td>dec</td>
<td>The (total) number of nodes enumerated inside the decoding algorithm</td>
</tr>
<tr>
<td>babai_op</td>
<td>dec</td>
<td>The (total) cost of the decoding algorithm</td>
</tr>
<tr>
<td>$k$</td>
<td>usvp, dec, dual</td>
<td>The number of guessed components of the LWE secret</td>
</tr>
<tr>
<td>postprocess</td>
<td>usvp, dec, dual</td>
<td>The maximal hamming weight considered in the search space</td>
</tr>
<tr>
<td>$c$</td>
<td>dual</td>
<td>The scaling factor considered in the dual lattice</td>
</tr>
</tbody>
</table>

Table 2.4: Outputs from the LWE Estimator.

*We use the notation $\delta$ for this quantity.*
2.10 The Learning with Errors Estimator

The attack parameters $k$ and postprocess only appear in the case that certain small-secret distributions are used, and these attack parameters will be discussed in full detail in Chapter 4.

2.10.2 Implemented Features

There are a variety of attack techniques considered in the LWE Estimator and, as new attack techniques surface, these techniques may or may be supported. Moreover, there may be attack techniques which can leverage (for example) small secret distributions, which may be implemented in the case of sparse ternary secrets, but not in the case of uniform ternary secrets. Although the code is open source, it may not be immediately obvious when a new attack technique has been implemented. To help with this issue, we present in Figure 2.10 a diagram of attack techniques considered, and identify which attack techniques are considered for which secret distributions. We comment on each of the attacks listed in Figure 2.10.

- **classic uSVP** refers to the uSVP attack outlined in Section 2.9.2.
- **guess + uSVP** refers to the small-secret variant of the uSVP attack outlined in Chapter 4.
- **LP11 decoding** corresponds to the decoding attack outlined in section 2.9.3, implemented using Linder and Peikert’s Nearest Planes algorithm as the BDD solver [LP11].
- **guess + LP11 decoding** corresponds to the small-secret variant of the decoding attack outlined in Chapter 3 (implemented using Linder and Peikert’s Nearest Planes algorithm as the BDD solver [LP11]).
- **LN13 decoding** refers to both the decoding attack outlined in section 2.9.3 (implemented using pruned enumeration as the BDD solver [LN13]).
- **guess + LN13 decoding** corresponds to the small-secret variant of this attack as outlined in Chapter 3 (implemented using pruned enumeration as the BDD solver [LN13]).
- **g-v decoding** refers to the guess-and-verify decoding technique outlined in Chapter 3.
- **hybrid-decoding** refers to the hybrid lattice reduction and meet-in-the-middle attack [How07], discussed in Chapters 3 and 4.
- **classic dual** corresponds to the dual attack as outlined in Section 2.9.1.
- **SILKE** refers to the techniques outlined in [Alb17], as discussed in Chapter 4.
• hybrid-dual corresponds to the hybrid dual and meet-in-the-middle attack [CHHS19].

2.11 Public-key Encryption from LWE

Public-key encryption [DH76, RSA78] provides a way for two parties to communicate over an insecure channel, without the need to have agreed on a pre-shared secret. We begin this section by informally defining the notion of a public-key encryption scheme.

**Definition 2.41 (Public-key Encryption Scheme)** A public-key encryption scheme is made up of three algorithms (KGen, Enc, Dec) satisfying:

1. \((sk, pk) \leftarrow KGen(1^n)\). The key generation algorithm \(KGen\) takes as input the security parameter, and outputs a pair of keys \((pk, sk)\), where \(pk\) is the public key and \(sk\) is the private key.

2. \(c \leftarrow Enc(pk, m)\). The encryption algorithm \(Enc\) takes as input the public key \(pk\) and message \(m\), and returns a ciphertext \(c\).

3. \(m \leftarrow Dec(sk, c)\) The decryption algorithm \(Dec\) takes as input the secret key \(sk\) a ciphertext \(c\), and returns a message \(m\).

We also require correctness, which ensures that a ciphertext decrypts to the correct underlying message.

**Definition 2.42 (Correctness of Public-key Encryption)** A public-key encryption scheme is referred to as correct if for any \(m \in M\), we have:

\[
\Pr[Dec(sk, c) = m \mid (sk, pk) \leftarrow KGen(1^n), c \leftarrow Enc(pk, m)] = 1.
\]

**Example 2.11** We present the Ring-LWE-based public-key encryption scheme outlined by Regev [Reg05], and show how it can be used to encrypt a bit \(b \in \{0, 1\}\). Note that the cryptosystem parameters need to be correctly tuned in order to guarantee security. Let \(n, m, q\)
Figure 2.10: Structure of the top level function `estimate_lwe` in the LWE Estimator. Implemented attacks are listed in green, whilst those not implemented are listed in red. Supported secret distributions are listed below implemented attacks. Here we assume that $a \leq 0$ and $b > 0$.

---

This refers to all secret distributions not contained within the set \{uniform, normal, $((-1,1),h)$\}. 
be integers and let \( \chi \) be a probability distribution\(^4\) on \( \mathbb{Z}_q \). The secret key \( sk \) is a uniform random vector \( s \in \mathbb{Z}_n^q \), i.e \( sk = s \). The public key is selected as follows: we draw several samples \((a_i, \langle a, s \rangle + e_i)\) from the LWE distribution \( L_{s, \chi, q} \). To encrypt a bit \( b \in \{0, 1\} \), we choose a random set \( S \) from the set of all \( 2^m \) bit strings of length \( m \) and the encryption function outputs:

\[
\text{Enc}_{pk}(b) = \begin{cases} 
(\sum_{i \in S} a_i, \sum_{i \in S} b) & \text{if } b = 0 \\
(\sum_{i \in S} a_i, \lfloor \frac{q}{2} \rfloor + \sum_{i \in S} b) & \text{if } b = 1
\end{cases}
\]

The decryption algorithm takes as input some vector of the form \((a, b)\), and returns 0 if \( b - \langle a, s \rangle \) is closer to 0 than to \( \lfloor \frac{q}{2} \rfloor \), and 1 otherwise.

### 2.12 Homomorphic Encryption

The search for Fully Homomorphic Encryption (FHE) dates back to 1978 [RAD78] and was an open question until 2009, when the problem was solved by Gentry [Gen09]. A variety of intermediary steps were taken [Pai99, BGN05] prior to Gentry’s discovery, but none of the schemes prior to Gentry’s work were fully homomorphic. Since then, there have been a variety of schemes [FV12, BGV14, CGGI16] and several libraries [HEA20, Mic20, Pal20, TFH20, Con20], and there are numerous interesting applications [ACC+17], including privacy-preserving machine learning [GBDL+16, BCL+18, JKLS18, CCD+19, CKR+20]. The iDash competition [Ida19] runs each year, challenging participants to produce a privacy-preserving solution using homomorphic encryption. As an example, Track 2 in the 2019 iDash competition was entitled Secure Genotype Imputation using Homomorphic Encryption. Currently, a standardisation effort [BDH+17, ACC+17, ACC+18] is taking place, and companies are beginning to commercialise homomorphic encryption techniques.

We begin this section with some definitions, following the notation of [Vai11].

**Definition 2.43 (Homomorphic Encryption Scheme)** A homomorphic encryption scheme is made up of four algorithms (KGen, Enc, Dec, Eval) satisfying:

1. \((pk, ek, sk) \leftarrow \text{KGen}(1^n)\). The key generation algorithm KGen takes as input the security parameter, and outputs a tuple of keys \((pk, ek, sk)\), where \( pk \) is the public encryption key, \( ek \) is a public evaluation key, and \( sk \) is the private key.

\(^4\)We note that Regev considers a Gaussian distribution in order to ensure security.
2.12 Homomorphic Encryption

2. \( c \leftarrow \text{Enc}(pk,m) \). The encryption algorithm Enc takes as input the public key \( pk \) and message \( m \), and returns a ciphertext \( c \).

3. \( m \leftarrow \text{Dec}(sk,c) \). The decryption algorithm Dec takes as input the secret key \( sk \) a ciphertext \( c \), and returns a message \( m \).

4. \( c_f \leftarrow \text{Eval}(ek,f,c_1,c_2,\ldots,c_k) \). The evaluation algorithm Eval takes as input the public evaluation key \( ek \), a function \( f \), typically represented as an algebraic circuit, and \( k \) ciphertexts \( c_1,c_2,\ldots,c_k \), with each \( c_i = \text{Enc}(pk,m_i) \), and outputs a ciphertext \( c_f \).

Definition 2.44 (\( C \)-homomorphism) Denote by \( C \) a class of functions. A homomorphic encryption scheme \((KGen,\text{Enc},\text{Dec},\text{Eval})\) is \( C \)-homomorphic if for any function \( f \in C \), we have:

\[
\Pr[\text{Dec}(sk, (\text{Eval}(ek,f,c_1,c_2,\ldots,c_k)))) \neq f(m_1,m_2,\ldots,m_k)] = \text{negl}(\lambda)
\]

where \((pk,ek,sk) \leftarrow KGen(1^\lambda)\), and \( c_i \leftarrow \text{Enc}(pk,m_i) \), for \( 1 \leq i \leq k \).

Definition 2.45 (Compactness) A homomorphic encryption scheme \((KGen,\text{Enc},\text{Dec},\text{Eval})\) is compact if there exists a polynomial \( s = s(\lambda) \) such that the output length of \( \text{Eval}(\cdot) \) is at most \( s \)-bits long.

The definition of compactness tells us two things. Firstly, that the length of a ciphertext output by the evaluate algorithm is “not too long” (i.e. bounded by some polynomial \( s(\lambda) \)) and, secondly, that the length of this output is independent of both the input function \( f \) and the number of ciphertexts \( c_1,c_2,\ldots,c_k \).

Definition 2.46 (Fully Homomorphic Encryption Scheme) A homomorphic encryption scheme is fully homomorphic if it is both compact and homomorphic for the class of all arithmetic circuits over \( GF(q) \).
2.12 Homomorphic Encryption

That is, a fully homomorphic encryption scheme can evaluate any arithmetic circuit. As a basic example of a homomorphic encryption scheme, we consider the RSA algorithm.

**Example 2.12** The ‘Textbook RSA’ algorithm is multiplicatively homomorphic. Let $p,q$ be two $k$-bit primes, $n = pq$, and choose $e$ such that $\gcd(e, \phi(n)) = 1$ (here $\phi$ is Euler’s Totient function $[OLBC10]$). We choose $d$ such that $ed \equiv 1 \mod \phi(n)$, and set $pk = e, sk = d$. Encryption is performed on a message $m$ via computing the function:

$$\text{Enc}(e, m) = m^e \mod n,$$

and decryption is performed on $c$ via the computing the function:

$$\text{Dec}(d, c) = c^d \mod n.$$

We now highlight the multiplicatively homomorphic nature of this scheme. Let $c_1 = m_1^e \mod n$ and $c_2 = m_2^e \mod n$, we have:

$$c_1 c_2 = (m_1^e \mod \phi(n)) (m_2^e \mod \phi(n))$$

$$= (m_1 m_2)^e \mod \phi(n)$$

$$= \text{Enc}(e, m_1 m_2).$$

That is, by performing multiplication on two ciphertexts, we retrieve an encryption of the multiplication of the underlying plaintexts, i.e we have multiplicatively homomorphic encryption.

In this thesis we cryptanalyse LWE-based homomorphic encryption-style parameter sets in Chapters 3 and 5. Also in Chapter 5 we outline in detail three lattice-based homomorphic encryption schemes: BGV [BGV12], BFV [FV12], and CKKS [CKKS17]. In Chapter 6 we make use of the Paillier homomorphic encryption scheme [Pai99].
Chapter 3

Batch Bounded Distance Decoding

Contents

3.1 Introduction and Contribution ............................................. 71
  3.1.1 An Overview of Decoding Attacks ................................ 72
3.2 A Comparison of Success Conditions .................................... 76
3.3 The Hybrid-decoding Attack .............................................. 78
  3.3.1 Common Assumptions in Analyses of the Hybrid-decoding Attack 81
  3.3.2 Modelling Lattice Reduction for $q$-ary Bases .................. 84
3.4 A Spectrum of Decoding Approaches for Solving Small-secret LWE 87
  3.4.1 Small-secret Decoding ............................................ 89
  3.4.2 Drop-and-solve Decoding ........................................ 89
  3.4.3 Guess-and-verify Decoding ...................................... 90
3.5 Target Parameter Sets and Estimates ................................... 92
  3.5.1 NTRU Prime ..................................................... 92
  3.5.2 Round5 ......................................................... 93
  3.5.3 HElib ......................................................... 94
  3.5.4 Results and Notation ........................................... 95
  3.5.5 Results in the Enumeration Regime ............................ 95
  3.5.6 Results in the Sieving Regime ................................ 100
3.6 Assumptions Case Study: NTRU Prime .................................. 100
3.7 Conclusion .............................................................. 105

This chapter is based on the following publication: Martin R. Albrecht, Benjamin R. Curtis and Thomas Wunderer. Exploring Trade-offs in Batch Bounded Distance Decoding. Selected Areas of Cryptography 2019 (pp. 467-491). Springer, volume 11959 of Lecture Notes in Computer science, 2019. Additional details have been added in this thesis.

In this chapter, we explore trade-offs in Howgrave-Graham’s hybrid lattice-reduction and meet-in-the-middle attack (hybrid-decoding attack) on NTRU and LWE. Specifically, we consider...
3.1 Introduction and Contribution

The effect of using a BDD solver with a higher probability of success on the landscape of trade-offs.

The author of this thesis contributed towards (a) the writing of the paper, (b) the design of the g-v decoding algorithm, (c) the implementation of the attack scripts, and (d) the running of all experiments.

3.1 Introduction and Contribution

As defined in Chapter 2, the Bounded Distance Decoding problem (BDD) with parameter $\alpha > 0$ asks a challenger to find the closest vector in a lattice $L \subset \mathbb{R}^d$ to some given target vector $t \in \mathbb{R}^d$, under the guarantee that the distance between the target vector $t$ and the lattice $\Lambda$ is at most $\alpha \lambda_1(L)$. We can use algorithms which solve BDD to solve variants of the Learning with Errors problem and, therefore, we can estimate the security of lattice-based cryptographic primitives based on the running times of these algorithms. In this chapter, we are particularly interested in solving the small-secret variant of the Learning with Errors problem, as defined in Section 2.6.2, where the components $s_i$ of the secret vector are drawn from some distribution over the set $\{-1, 0, 1\}$.

Due to the shortness and/or sparsity of the secret vector $s$, batches of BDD instances emerge from a combinatorial approach where several components of the target vector are guessed before decoding. In this chapter, we explore trade-offs in solving batches of (candidate) BDD instances, which we refer to as “Batch BDD” throughout, and apply our techniques to the NTRU Prime \cite{BCLv19} and Round5 \cite{GZB19} schemes submitted to the NIST standardisation process \cite{Nat16}, as well as a sparse-secret parameter set previously used for the homomorphic encryption library HElib \cite{HElib20}.

We compare our techniques to a variant of Howgrave-Graham’s hybrid meet-in-the-middle and lattice reduction attack (hybrid-decoding attack) \cite{How07} under a variety of different assumptions. Specifically, we consider Howgrave-Graham’s attack when a “classical” guessing approach is used, and also when a meet-in-the-middle guessing approach is used (under conservative assumptions).
3.1 Introduction and Contribution

3.1.1 An Overview of Decoding Attacks

In this section, we outline the relationships between four techniques to solve LWE via BDD:

1. the decoding attack,
2. the drop-and-solve decoding attack, which combines the decoding attack with zero-forcing techniques,
3. the hybrid-decoding attack, which solves batches of candidate BDD instances using Babai’s Nearest Plane algorithm, and
4. the g-v decoding attack, outlined in this chapter, which solves batches of candidate BDD instances using pruned enumeration.

The purpose of this section is to explain the relationship between these four techniques via high level descriptions. The technical details of each of these four techniques can be found later in this chapter.

3.1.1.1 The Decoding Attack

To solve Small-secret LWE via BDD we use a variation of the decoding attack outlined in Section 2.9.3. For a collection of LWE samples \( \{(a_i, \langle a_i, s \rangle + e_i)\}_{i=1}^{m} \) of dimension \( n \), we begin by finding a basis of sufficient quality for the dimension \( (n + m) \) lattice:

\[
\{(x, Ax \mod q) \mid x \in \mathbb{Z}^n\},
\]

followed by usage of a BDD solver to solve the resulting BDD instance, of which a typical choice would be pruned enumeration \([\text{LP11, LN13}]\).

The decoding attack can be summarised in two phases:

1. a lattice reduction phase producing a lattice basis of a sufficient quality, and
2. a decoding phase where a BDD solver is run against the reduced lattice from step 1, using a known target vector as input.
3.1 Introduction and Contribution

3.1.1.2 The Drop-and-solve Decoding Attack

We can combine the decoding attack with dimension reduction techniques by guessing (typically as zero, and referred to as “zero-forcing” [MS01]) e.g. the first $\kappa$ components of the secret vector $s$. This approach begins by finding a basis of sufficient quality for the dimension $(n + m - \kappa)$ lattice:

$$\{ (y, A_{(\kappa)} y \mod q) \mid y \in \mathbb{Z}^{n-\kappa} \},$$

followed, again, by usage of a BDD solver to solve the resulting dimension-reduced BDD instance. The LWE secret can be recovered (with some probability, associated to the BDD solver) via solving the BDD instance in the case that the guessed $\kappa$ components of the secret are all zero. This technique is referred to as the drop-and-solve decoding attack in the literature, and follows the mantra of dimension reduction outlined in [MS01, Alb17, ACD+18].

The drop-and-solve decoding attack can be summarised in three phases:

1. a zero-forcing phase which selects a zero-forcing dimension $\kappa \leq n$, and generates the resulting $(n + m - \kappa)$ dimension lattice basis to be reduced,
2. a lattice reduction phase producing a lattice basis of a sufficient quality, and
3. a decoding phase where a BDD solver is run against the reduced lattice from step 1, using a known target vector as input.

3.1.1.3 The Hybrid-decoding Attack

The hybrid-decoding attack [How07, BGPW16, GvW17, Wun19] admits batches of BDD instances which emerge from guessing some components of the target vector $(s, e)$, as above, combined with an algebraic observation which allows several points to be decoded against the same lattice. This approach begins by finding a basis of sufficient quality for the dimension $(n + m - \tau)$ lattice:

$$\{ (y, A_{(\tau)} y \mod q) \mid y \in \mathbb{Z}^{n-\tau} \},$$

followed by multiple calls to a decoding algorithm (Babai’s Nearest Plane algorithm), the cost of which is polynomial in the lattice dimension. This makes Babai’s algorithm an appropriate choice of BDD solver to be employed in the hybrid-decoding attack, since the adversary has to call the BDD solver many times: once for each guess.
3.1 Introduction and Contribution

The hybrid-decoding attack can be summarised in three phases:

1. a *dimension reduction phase* which selects a guessing dimension $\tau \leq n$, and generates the resulting $(n + m - \tau)$ dimension lattice basis to be reduced,
2. a *lattice reduction phase* producing a lattice basis of a sufficient quality, and
3. a *guess-and-verify phase* where guesses are verified by running Babai’s Nearest Plane against the reduced lattice basis from step 1, and a target vector derived from the particular guess under consideration.

Therefore, unlike the decoding and drop-and-solve decoding techniques, the hybrid-decoding attack permits *multiple decodings per lattice reduction step*. That is, in phase three we are solving batches of (candidate) BDD instances: one for each guess. This guess-and-verify step is usually considered to be realised using a meet-in-the-middle (mitm) approach.

3.1.1.4 The Guess-and-verify Decoding Attack

The uSVP approach outlined in Section 2.9.2 and the hybrid-decoding approach can be considered as the endpoints of a continuum of strategies for solving Batch-BDD. The final enumeration is either (essentially) as expensive as the initial lattice reduction algorithm (as in uSVP) or optimised to be as cheap as possible, in order to decode a large number of points (as in the hybrid-decoding attack). In this chapter, we will explore this continuum of strategies for solving Small-secret LWE instances.

Specifically, we present a *guess-and-verify* decoding approach (g-v decoding) which, like the hybrid-decoding attack, makes use of a guessing approach to reduce the dimension of the associated BDD problem. However, in our g-v decoding attack, we employ a more expensive BDD solver than Babai’s Nearest Plane. Here, as in the hybrid-decoding attack, this approach begins by finding a basis of sufficient quality for the dimension $(n + m - \tau)$ lattice:

\[ \{(y, A_\tau y \mod q) \mid y \in \mathbb{Z}^{n-\tau}\}, \]

followed by multiple calls to a decoding algorithm. However, the decoding algorithm considered here is pruned enumeration. To establish the dimension of the projected sublattice in which we perform enumeration (i.e. the *enumeration dimension*, denoted $\eta$), we deploy (a slight
3.1 Introduction and Contribution

variant of) the success condition for uSVP from [ADPS16], outlined in Section 2.9.2 and, in Section 3.2 we explain and compare these success conditions with a concrete example.

The g-v decoding attack can be summarised in three phases:

1. a dimension reduction phase which selects a guessing dimension $\tau \leq n$, and generates the resulting $(n + m - \tau)$ dimension lattice basis to be reduced,

2. a lattice reduction phase producing a lattice basis of a sufficient quality, and

3. a guess-and-verify phase where guesses are verified by running pruned enumeration in dimension $\eta$ against the reduced lattice basis from step 1, and a target vector derived from the particular guess under consideration.

Therefore, as opposed to applying a low probability BDD solver on a large number of (candidate) BDD instances (e.g. Babai’s Nearest Plane algorithm, as in the hybrid-decoding attack), our g-v decoding technique applies a heavier enumeration process, with a higher probability of success, to a (typically) smaller number of (candidate) BDD instances. In doing so, we hope to achieve a positive trade-off in the overall running time of the attack when considered in comparison to the hybrid-decoding attack.

3.1.1.5 Summary

These four attacks (decoding, drop-and-solve decoding, hybrid-decoding, and guess-and-verify decoding) all share the same genealogy. However, it is not the case that one of these techniques always outperforms the others. Indeed, the ordering of these attack techniques depends on the assumptions being used, and the parameter set under consideration.

---

1. This is because the cost of performing pruned enumeration in some dimension $\eta$ is typically more expensive than Babai’s Nearest Plane algorithm. Therefore, for the same “enumeration budget”, we can perform less BDD calls when using pruned enumeration compared to the Babai’s Nearest Plane algorithm.
3.2 A Comparison of Success Conditions

<table>
<thead>
<tr>
<th>Attack</th>
<th>Guessing?</th>
<th>Multiple Guesses?</th>
<th>Enumeration algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoding</td>
<td>No</td>
<td>No</td>
<td>Pruned enumeration</td>
</tr>
<tr>
<td>Drop-and-solve decoding</td>
<td>Yes</td>
<td>No</td>
<td>Pruned enumeration</td>
</tr>
<tr>
<td>Hybrid-decoding</td>
<td>Yes</td>
<td>Yes</td>
<td>Babai’s Nearest Plane</td>
</tr>
<tr>
<td>Guess-and-verify decoding</td>
<td>Yes</td>
<td>Yes</td>
<td>Pruned Enumeration</td>
</tr>
</tbody>
</table>

Table 3.1: A summary of attacks found in the literature: decoding, drop-and-solve-decoding, and hybrid-decoding, as well as our guess-and-verify decoding technique.

3.2 A Comparison of Success Conditions

The success condition from [ADPS16] for uSVP considers the length of the GSO vectors in the context of the uSVP attack. As outlined in Section 2.9.2 for a given uSVP lattice $L$ of dimension $d$, the unusually short vector in which the LWE secret is embedded can be recovered by BKZ using blocksize $\beta$, if the condition:

$$\sqrt{\frac{\beta}{d}} \cdot \lambda_1(\Lambda) \approx \sqrt{\beta} \cdot \sigma \leq \delta^{2\beta-d} \cdot \det(L)^{1/d},$$

is satisfied. The variant of this condition used in this work considers the Gaussian Heuristic directly, and the code used to compute the dimension $\eta$, in which we perform enumeration, can be seen in Figure 3.1. This condition is exactly that the length of the projection of the shortest vector (in the lattice $\pi_i(L)$, as predicted by the Gaussian Heuristic) is less than the expected length of the target, offset, vector, i.e. $\sigma \sqrt{d-i}$. Specifically, the condition states:

$$\lambda_1(\pi_i(L)) \approx \text{Vol}(\pi_i(L))^{1/(d-i)} \cdot \frac{\Gamma(d-i+1)^{1/(d-i)}}{\sqrt{\pi}} \leq \sigma \sqrt{d-i}.$$  \hspace{1cm} (3.2)

For the first index $j$ for which this condition is satisfied, we set the enumeration dimension $\eta$ as $\eta = d - (j - 1)$ and, if this condition is never satisfied, the code reverts to considering an enumeration dimension of $\eta = 2$. The choice of $\eta = 2$ corresponds to performing lattice reduction such that Babai’s Nearest Plane algorithm succeeds with probability close to 1 (which is typically not optimal).

Since the ADPS16 success condition is defined such that $\eta = \beta$, i.e. requires the final block of the lattice basis to be HKZ reduced, we consider this case in an example. We consider a $q$-ary lattice of dimension $d = 1522$, with $q = 4591$ and determinant $q^{d/2}$. For blocksize $\beta = 561$
3.2 A Comparison of Success Conditions

```python
def ball_log_vol(n):
    Return volume of ‘n’-dimensional unit ball
    :param n: dimension
    
    return (n/2.) * log(pi) - lgamma(n/2. + 1)

def gaussian_heuristic(r):
    Return squared norm of shortest vector as predicted by the Gaussian heuristic.
    :param r: vector of squared Gram-Schmidt norms
    
    n = len(list(r))
    log_vol = sum([log(x) for x in r])
    log_gh = 1./n * (log_vol - 2 * ball_log_vol(n))

    return exp(log_gh)

def _enum_dim(r, alpha, q):
    Return eta for a given lattice shape and LWE noise.
    :param r: squared GSO norms
    :param alpha: LWE noise rate
    :param q: LWE modulus
    
    stddev = est.stddevf(alpha*q)
    d = len(r)
    for i, ri in enumerate(r):
        if gaussian_heuristic(r[i:]) < stddev**2 * (d-i):
            return ZZ(d-(i-1))

    return ZZ(2)
```

Figure 3.1: Code used to generate the required enumeration dimension which guarantees success in the g-v decoding approach. Note that this code considers the squared norms of the vectors $\|b_i\|$, and thus varies slightly from Equation 2.2. The functions `gaussian_heuristic()` and `ball_log_vol()` are taken from the FPLLL library [FPL20].
3.3 The Hybrid-decoding Attack

satisfying Equation 3.1 we have \( \eta_{\text{ADPS}} = \beta = 561 \). Using Equation 3.2, we can determine that \( \eta_{\text{gh}} = 562 \). The success condition considered in our work, i.e. Equation 3.2, explicitly constructs a \( \text{BDD}_\alpha \) instance with \( \alpha = 1 \). Note that we are not recovering the short vector directly via lattice reduction, but instead via the usage of an enumeration algorithm with an associated probability of success. The success condition we consider essentially states the following:

\[
\text{When lattice reduction is conducted with blocksize} \ \beta, \ \text{the short vector can be recovered via solving} \ \text{BDD}_{\alpha=1} \ \text{in dimension} \ \eta, \ \text{where} \ \eta \ \text{satisfies Equation} \ 3.2.
\]

whereas the \([\text{ADPS16}]\) success condition essentially states:

\[
\text{When lattice reduction is conducted with blocksize} \ \beta \ \text{satisfying the} \ [\text{ADPS16}] \ \text{success condition, i.e. Equation} \ 3.1, \ \text{the short vector is recovered directly, and is contained within the output basis provided by BKZ.}
\]

We note that, however, the condition used in our work allows the parameters \( \beta \) and \( \eta \) to be uncoupled. That is, we no longer require the condition that \( \eta = \beta \) as in Equation 3.1.

3.3 The Hybrid-decoding Attack

The hybrid meet-in-the-middle and lattice reduction (hybrid-decoding) attack was introduced by Howgrave-Graham [How07], and has been considered in the context of LWE with binary error [BGPW16], with a quantum search [GvW17], and is typically used to set parameters for encryption schemes based on the NTRU problem [HPS+15]. This attack combines the ideas of May and Silverman’s dimension reduction techniques [MS01] against the NTRU cryptosystem, with an algebraic observation which allows multiple (candidate) BDD instances to be solved per lattice reduction step. As opposed to solving BDD in the dimension \( d = m + n \) lattice with basis:

\[
B = \begin{pmatrix}
qI_m & A \\
0 & I_n
\end{pmatrix},
\]

78
3.3 The Hybrid-decoding Attack

The hybrid-decoding attack sets a guessing dimension $\tau$, carries out lattice reduction in dimension $(d - \tau)$ and solves BDD on a dimension $(d - \tau)$ lattice by decoding on various points associated to guesses in the $\tau$-dimensional guessing space. In more detail: the first phase of the hybrid-decoding attack involves choosing a guessing dimension $\tau$ and generating the lattice basis determined by the matrix:

$$B = \begin{pmatrix} qI_m & A_{(\tau)} \\ 0 & I_{n-\tau} \end{pmatrix},$$

recall that $A_{(\tau)}$ denotes the matrix $A$ where (without loss of generality) the first $\tau$ columns have been dropped. The second phase involves performing lattice reduction on this basis to retrieve $\text{BKZ}_\beta(B)$, and the third phase involves solving a batch of candidate (BDD) instances.

Figure 3.2: A comparison of success conditions. Here, the GSO norms are computed using the Geometric Series Assumption. The dashed blue line represents the expected length of the projected target vector in the projected sublattice $\pi_i^*(L)$. The dashed black line represents the length of the shortest vector in the same projected sublattice $\pi_i^*(L)$ as predicted by the Gaussian Heuristic. Note that we have used the GSA here to produce the GSO norms. This analysis can also be carried out using a BKZ Simulator to generate the GSO norms, and we consider both of these techniques (GSA and BKZ Simulator) throughout this chapter.
using Babai’s Nearest Plane algorithm [Bab86] on target points related to vectors in the \( \tau \)-dimensional guessing space. We have:

\[
\begin{pmatrix}
q I_m & A_{(\tau)} \\
0 & I_{n-\tau}
\end{pmatrix}
\begin{pmatrix}
\ast \\
\ast
\end{pmatrix}
= 
\begin{pmatrix}
q^* + A_{(\tau)} s_{(\tau)} \\
\ast
\end{pmatrix}.
\]

If the first \( \tau \) components of \( s \) are all zero, then it is the case that \( A_{(\tau)} s_{(\tau)} = A s \), which allows recovery of the LWE secret (with some probability, determined by the BDD solver) by decoding on the point \((b, 0)\). If the first \( \tau \) components of the LWE secret \( s \) are not all equal to zero, then we are required to start searching for the first \( \tau \) components of the secret (e.g. over the space \([-1, 0, 1]^\tau\)). When \( s \leftarrow B^- \), the coefficients \( s_i \) of the secret vector are contained within the set \([-1, 0, 1]\), and we observe that:

\[
As = \sum_{\{i|s_i=1\}} A_i - \sum_{\{j|s_j=-1\}} A_j.
\]

(3.3)

Recalling that the unit vectors are represented by \( u_k \), we can make a guess:

\[
v_g = \sum_{k=1}^\tau c_k u_k,
\]

for some values \( c_k \in \{-1, 0, 1\} \). For this new guess, we can decode on the point \((b - \sum_{k=1}^\tau c_k A_k, 0)\). This can be seen to be the correct point to attempt decoding on since, for the correct guess \( v = \sum_{k=1}^\tau s_k u_k \), we have:

\[
b - \tau \sum_{k=1}^\tau s_k A_k = As - \tau \sum_{k=1}^\tau s_k A_k + e
= A_{(\tau)} s_{(\tau)} + e \mod q.
\]

Therefore, we have that \((b - \sum_{i=1}^\tau s_k A_k, 0)\) is separated from the lattice point \((A_{(\tau)} s_{(\tau)} \mod q, s_{(\tau)})\) by the vector \((-e, s_{(\tau)})\), as is required. This means that we can consider a batch of BDD instances, by decoding on several target points of the form \((b - \sum_{k=1}^\tau c_k A_k, 0)\).

Typically, this guessing is realised via the usage of a time-memory trade-off approach outlined in [Wum19]. That is, we can consider Odlyzko’s meet-in-the-middle technique described in [HGSW03] for the guessing phase of the attack. Specifically, the guessed part of the secret \( v_g \) is split into two sub-guesses \( v'_g \) and \( v''_g \) satisfying \( v_g = v'_g + v''_g \), and we in turn have two applications of Babai’s Nearest Plane: one for each “half” of the original guess:

\[
v'_l = NP_B (b - \sum_{k=1}^\tau (v'_{g(k)} A_k), 0),
\]
3.3 The Hybrid-decoding Attack

\[ v''_l = \text{NP}_B \left( - \sum_{k=1}^{\tau} (v''_g)_{(k)} A_k \right, 0 \).

We hope that the BDD solver is additively homomorphic, i.e. we hope that if \( v_g = v'_g + v''_g \) then we have \( v_l = v'_l + v''_l \). We store each decoded vector in a hash table using a locality sensitive hash function [Wun19] which permits collision finding. A collision involves detecting pairs \( (v'_g, v''_g) \) which satisfy \( v_g = v'_g + v''_g \) and \( v_l = v'_l + v''_l \). Wunderer [Wun19] gives a concrete example of hash tables which guarantee that collisions can be detected. The probability that the BDD solver is additively homomorphic has been analysed in the case that the BDD solver is Babai’s Nearest Plane in [Wun19]. However, this refined model is not employed e.g. in submissions to the NIST PQC process [BCLv17, SHRS17, ZCHW17a]. To summarise, there are two probabilities at play when considering a meet-in-the-middle approach:

1. the probability that the BDD solver is homomorphic, and

2. the probability that collisions (as outlined above) are detected; which we can set to 1 by assuming the hash tables of Wunderer [Wun19].

3.3.1 Common Assumptions in Analyses of the Hybrid-decoding Attack

Throughout any analysis of the hybrid-decoding attack, there are several stages in which assumptions are required in order to generate concrete security estimates. In this section, we outline locations within an analysis in which assumptions are required, and we also outline common choices which are made in the literature.

1. The lattice reduction cost model.

The cost of lattice reduction is an important assumption in an analysis of the hybrid-decoding attack. Here we are referring explicitly to the cost of the BKZ algorithm, that is:

\[ T_{BKZ}(\beta, d), \]

where an example cost is \( T_{BKZ}(\beta, d) = 8d \cdot 2^{0.292\beta+16.4} \).

2. The shape of the basis output by lattice reduction.
3.3 The Hybrid-decoding Attack

As discussed in Section 3.3.2, typical choices are to consider the Geometric Series Assumption [Sch03], the adapted $q$-ary Geometric Series Assumption [Wun19], or a BKZ simulator e.g. [CN11]. The output shape has an effect on the success probability of Babai’s Nearest Plane algorithm (note the inclusion of the GSO lengths in the $p_{\text{babai}}$ terms in 3.), and thus the overall running time of the attack.

3. The success probability of Babai’s Nearest Plane, $p_{\text{babai}}$.

Typical choices are either to (conservatively) set this probability to be 1, or to use the formula from Wunderer [Wun19] which approximates the probability as:

$$p_{\text{babai}} \approx \prod_{1 \leq i \leq d} \left(1 - \frac{2}{B\left(\frac{d-1}{2}, \frac{1}{2}\right)} \int_{\min(r_i, 1)}^{1} (1 - t^2)^{(d-3)/2} dt \right).$$

Here $B(\cdot)$ denotes the Beta function [OLBC10], and $r_i = \frac{\|b_i^*\|}{\|b_i^*\|_2}$. Previous works [LP11] have also assumed that Babai’s Nearest Plane algorithm succeeds with probability:

$$p_{\text{babai}} \approx \prod_{1 \leq i \leq d} \text{erf} \left( \frac{\|b_i^*\|_2 \sqrt{\pi}}{2 \alpha q} \right).$$

under the assumption of a continuous LWE error distribution. In our work, we use Wunderer’s formula, as is standard with state-of-the-art analyses of the hybrid-decoding attack.

4. The running time of Babai’s Nearest Plane algorithm, $T_{\text{babai}}$.

Typical choices are either to (conservatively) set this running time to be a single operation, i.e. 1, or to consider a cost which is either linear in the lattice dimension, such as $d/2^{1.06}$, or quadratic in the lattice dimension, such as $d^2/2^{1.06}$. In our work, we consider the quadratic cost, i.e. $d^2/2^{1.06}$.

5. The search strategy under consideration.

Typical choices are to consider a meet-in-the-middle approach, a quantum search, or a classical guessing (exhaustive search) approach. As an example, all three techniques are considered in the NTRU Prime submission to the second round of the NIST standardisation procedure [BCLv19].
3.3 The Hybrid-decoding Attack

6. The meet-in-the-middle probability, $p_{mitm}$.

Typical choices are to (conservatively) set this probability to be 1, or to consider the formula from [BGPW16, Wun19]:

$$p_{mitm} = \prod_{i=1}^{d} \left( 1 - \frac{1}{r_i B \left( \frac{d-1}{2}, \frac{1}{2} \right)} \int_{-r_i}^{-z+r_i} \int_{\max(-1,z-r_i)}^{z+r_i} \left( 1 - \frac{t^2}{2} \right) dt \, dz \right),$$

where the values $r_i$ and function $B(\cdot)$ are as in 3. This probability can be evaluated in Sage [S+20] using Wunderer’s scripts [Wun20]. An alternative formula was recently provided in [SCI19]:

$$p_s = \prod_{i=1}^{d} \left( \text{erf} \left( \frac{R_i \sqrt{\pi}}{\alpha q} \right) + \frac{\alpha q e^{-\left( \frac{R_i \sqrt{\pi}}{\alpha q} \right)^2}}{R_i \pi} - 1 \right),$$

under the assumption of a continuous LWE error distribution, as in [LPT11].

7. Whether or not to consider the cost of memory.

Most analyses of the hybrid-decoding attack only consider the time cost. That is, they do not consider memory costs of (a) lattice reduction (especially in the sieving regime, where the memory cost is exponential in the blocksize $\beta$), and (b) the meet-in-the-middle step. The second round NTRU Prime submission to the NIST standardisation procedure [BCLv19], considers different memory scenarios. That is, they consider an analysis which includes memory costs (referred to as “real” memory) and also consider an analysis which ignores memory costs (referred to as “free” memory).

8. The length of the target, offset, vector.

There is also a decision to be made about the length of the target vector. Suppose we set our guessing dimension to $\tau$, then our target vector is:

$$(s_{\tau}, e) = (s_{\tau+1}, \ldots, s_n, e).$$

Suppose further that $s$ is a ternary vector of Hamming weight $h$, so that $\|s\| = h$ and $\|e\| = m\sigma^2$, then we can assume that:

$$\| (s_{\tau+1}, \ldots, s_n, e) \| = \sqrt{h \frac{n - \tau}{n} + m \sigma^2}.$$
3.3 The Hybrid-decoding Attack

That is, we consider the \textit{expected length} of the target offset. We note that for any given instance of the attack, the target vector may be longer, or shorter, than this assumed length, and this has an impact on the overall running time of the attack.

In Section 3.6 we consider these assumptions for both the hybrid-decoding attack as well as our g-v decoding approach. We begin by making an initial set of assumptions, which match the assumptions made in the NTRU Prime submission to the NIST standardisation procedure, and swap out one assumption at a time until we reach the assumptions considered in our work. At each intermediate stage we consider the complexities of both attacks under the given assumptions, and we present the results in Tables 3.12 and 3.13. This highlights the differences in attack complexity that certain assumptions can make, and shows that it is vital to clearly state assumptions when analysing the hybrid-decoding attack.

3.3.2 Modelling Lattice Reduction for $q$-ary Bases

For lattice reduction we consider the BKZ algorithm \cite{SE94} parametrised by a block size $\beta$ which determines the running time (at least exponential in $\beta$) and output quality. In this chapter, when BKZ is instantiated with an enumeration algorithm \cite{FP85,Kan83}, we consider the cost of lattice reduction using blocksize $\beta$ on a lattice of dimension $d$ to be:

$$T_{BKZ}(\beta,d) = 8d \cdot 2^{0.18728 \cdot \beta \cdot \log(\beta) - 1.019 \beta + 16.1} \text{ enum. nodes},$$

which is taken from \cite{APS15} based on experiments from \cite{CN11}. To translate from the number of nodes visited during enumeration to CPU cycles, the literature typically assumes one node \approx 100 CPU cycles \cite{FPL20}. This means that in terms of ring-operations the cost of lattice reduction using blocksize $\beta$ on a lattice of dimension $d$ is:

$$T_{BKZ}(\beta,d) = 8d \cdot 2^{0.18728 \cdot \beta \cdot \log(\beta) - 1.019 \beta + 16.1 + \log(100)}.$$ 

When BKZ is instantiated with a sieving algorithm \cite{AKS01,BDGL16}, we consider the cost of lattice reduction using blocksize $\beta$ on a lattice of dimension $d$ to be:

$$T_{BKZ}(\beta,d) = 8d \cdot 2^{0.292 \beta + 16.4},$$

where the constant term is arbitrarily picked as in \cite{APS15}.

There are several models in the literature for the behaviour of the BKZ algorithm on $q$-ary lattices. In most of the literature on solving LWE via BDD, we use the public LWE matrix
3.3 The Hybrid-decoding Attack

\( A \in \mathbb{Z}_q^{(m \times n)} \) to construct a lattice basis of the form:

\[
\begin{pmatrix}
I_n & 0 \\
A & qI_m \\
\end{pmatrix},
\]

for which it is commonly assumed that the Geometric Series Assumption, defined in Definition 2.39, is relatively accurate after running BKZ-\( \beta \) with \( \beta \ll m + n \). The literature on analysing the hybrid-decoding attack considers lattice bases of the form:

\[
\begin{pmatrix}
qI_m & A \\
0 & I_n \\
\end{pmatrix},
\]

where the \( q \)s are in the top left hand corner of the basis matrix, and the 1s are located in the bottom right of the basis matrix. In Figure 3.3, we plot the initial GSO vector lengths from the basis as in Equation 3.5.

![Figure 3.3: Example of the initial GSO lengths for a \( q \)-ary lattice of dimension \( d = 180 \) with \( q = 17 \) constructed as in Equation 3.5.](image)

Wunderer [Wun19] notes that writing the basis matrix in the form of Equation 3.5 suggests that the GSA does not hold. Indeed, when the GSA predicts that \( \|b_1^*\| > q \) we notice that this is longer than the first vector already contained within the GSO basis. Since lattice reduction does not make the GSO vectors longer, we therefore will obtain \( \|b_1^*\| = q \). As a consequence, lattice reduction is expected to produce a “Z-shaped” basis [How07], comprised of leading \( q \)s, trailing ones and a middle part approximated by the GSA. In Figure 3.4 we give an illustrative example of this phenomena of the output of lattice reduction as implemented in FPLLL [FPL20] which clearly illustrates the Z-shape.

The structure of this Z-shape has been modelled by Wunderer [Wun19], via insisting on the
condition $\| b_i^* \| \leq q$ for all $1 \leq i \leq d$. Let $k$ be the number of lattice vectors which follow a GSA-style behaviour, meaning that the remaining $(d - k)$ vectors have length $q$. Explicitly, we have:

$$\| b_i^* \| = \begin{cases} q & \text{if } i \leq d - k \\ \delta^{2(i-(d-k)-1)+k} \cdot \frac{k^n}{q} & \text{otherwise} \end{cases}$$

If we make the additional assumption that the first vector in the “GSA-block”, i.e. $b_{(d-k+1)}$, satisfies $\| b_{(d-k+1)}^* \| \approx q$ [Wun19], we can compute a value for $k$, i.e. $k = \min \left( \left\lfloor \frac{n}{\log_q(\delta)} \right\rfloor, d \right)$. Wunderer refers to this shape as the “$q$-ary GSA”, and we follow this notation throughout. In Figure 3.4 we observe that, for this example, the number of leading $q$’s predicted by Wunderer closely resembles that produced by fpylll and the BKZ simulator. However, [Wun19] makes no attempt to model the number of trailing ones. Some works fix the number of trailing ones by choosing a sublattice to reduce [HPS +15], although there is currently no work in the literature that offers a way to predict the number of trailing ones which occur when BKZ is run on the full lattice.

![Figure 3.4: Example of BKZ-60 reduction on a $q$-ary lattice of dimension $d = 180$ with $q = 17$ and volume $17^{80}$ for bases constructed as in Equation (3.5), along with the output of BKZ simulation and the heuristic from [Wun19].](image)

We define $\gamma_i = \alpha^{i-1} \cdot \delta^d \cdot \det(\Lambda)^{1/d}$ and so the GSA corresponds to the line at $y = 0$.

Figure 3.4: Example of BKZ-60 reduction on a $q$-ary lattice of dimension $d = 180$ with $q = 17$ and volume $17^{80}$ for bases constructed as in Equation (3.5), along with the output of BKZ simulation and the heuristic from [Wun19].

For the remainder of this chapter, we do not consider the $q$-ary GSA. That is, we make the assumption that the $q$-ary structure of the lattice bases on which we perform lattice reduction does not impact the output basis shape. Explicitly, we do not assume that leading $q$s or training ones occur. This assumption can be made to hold by re-randomising the input basis for lattice reduction. Instead of reducing a lattice basis $B$ directly, we reduce the lattice basis given by $U_i B$ for some unimodular matrix $U_i$. Considering the techniques in this work in a regime exploiting the $q$-ary structure is an interesting area for future work. In Figure 3.5...
we consider the GSA against such a randomised basis:

\[ \mathbf{B'} = U\mathbf{B}, \]

to determine how closely this example follows the GSA and/or a BKZ simulator. In Figure 3.5, we plot the output GSO lengths against the GSA and the BKZ simulator. Here, we can see that both the GSA and the BKZ simulator provide a good approximation to the GSO lengths, although the BKZ simulator captures the HKZ-reduced behaviour in the last block more accurately.

![Figure 3.5: Example of BKZ-60 reduction on a \(q\)-ary lattice of dimension \(d = 180\) with \(q = 17\) and volume \(17^{80}\) for a basis constructed as in Equation 3.5, which has been re-randomised via the application of a unimodular matrix. We also plot the output of the BKZ simulator on a basis for the same lattice.](image)

We define \(\gamma_i = \alpha^{i-1} \cdot \delta^d \cdot \operatorname{det}(\Lambda)^{1/d}\) and so the GSA corresponds to the line at \(y = 0\).

Figure 3.5: Example of BKZ-60 reduction on a \(q\)-ary lattice of dimension \(d = 180\) with \(q = 17\) and volume \(17^{80}\) for a basis constructed as in Equation 3.5, which has been re-randomised via the application of a unimodular matrix. We also plot the output of the BKZ simulator on a basis for the same lattice.

### 3.4 A Spectrum of Decoding Approaches for Solving Small-secret LWE

In this section we outline the expected costs of:

1. the classical “decoding” approach in the LWE literature,
2. the “drop-and-solve decoding” approach, which is a combination of zero-guessing and solving a single BDD instance in a reduced dimension, and
3. our guess-and-verify decoding approach, where multiple BDD instances are solved per lattice reduction step.
In our guess-and-verify decoding approach, the attack parameters are chosen such that $p_{\text{bdd}} \approx 1$ and $p_{\text{babai}} \approx 1^2$, although for clarity we include these probabilities in the running times presented in this chapter. Note that the probability $p_{\text{bdd}} \approx 1$ ensures that the BDD algorithm succeeds when called on the correct target point. Exploring trade-offs which arise by varying these probabilities is interesting future work.

In order to produce these estimates, we make use of both enumeration-based, and sieving-based, BDD solvers. When using enumeration to solve BDD in dimension $\eta$, we assume a cost of:

$$T_{\text{bdd}}(\eta) = 2^{0.18728\eta \log(\eta) - 1.019\eta + 16.1} \text{ enum. nodes},$$

where again we assume that one node $\approx 100$ CPU cycles \cite{FPL20}. Such an enumeration is assumed to succeed with probability close to one, i.e. $p_{\text{bdd}} \approx 1$. When using sieving to solve BDD in dimension $\eta$, we assume a cost of:

$$T_{\text{bdd}}(\eta) = 2^{0.292\eta + 16.4},$$

based on the results of \cite{Laa16}, which suggest that sieving for short vectors has the same asymptotic cost as sieving for close vectors. We assume that this sieving process succeeds with probability close to one, i.e. $p_{\text{bdd}} \approx 1$. We note that it is always clear from context whether an enumeration-based, or a sieving-based, BDD solver is being deployed.

To highlight the differences in the three decoding approaches discussed, we make use of a running example to illustrate the behaviour of the approaches under consideration. We consider the small-secret LWE parameter set:

$$n = 653, q = 4621, \sigma \approx \sqrt{\frac{2}{3}}, \chi_s = B^{-100},$$

and use this parameter set as a reference throughout. Assuming the GSA, a combinatorial dual attack costs $2^{214.4}$ ring-operations ($\beta = 198$), and a combinatorial uSVP attack costs $2^{209.6}$ ring-operations ($\beta = 223$), according to the LWE Estimator \cite{APS15} under the enumeration-based BKZ cost model mentioned in Section 3.3.2.

\footnote{For g-v decoding, the probability $p_{\text{babai}}$ is the probability that the solution found by the BDD algorithm is lifted to the full lattice successfully.}

\footnote{Using the LWE Estimator as of commit 428d6ea.}

88
3.4 A Spectrum of Decoding Approaches for Solving Small-secret LWE

3.4.1 Small-secret Decoding

As discussed in Section 3.1, the decoding approach for small-secret LWE \[ b = A \cdot s + e \] constructs a lattice for which the vector \((b, 0)\) is separated by the short vector \((-e, s)\) to the lattice point \((A \cdot s \mod q, s)\). The basis of this lattice is given by the columns of the matrix \(B\), where:

\[
B = \begin{pmatrix}
q \cdot I_m & A \\
0 & I_n
\end{pmatrix}.
\]

In more detail, we have that:

\[
\begin{pmatrix}
q \\
0
\end{pmatrix} \cdot \begin{pmatrix}
A \\
I_n
\end{pmatrix} \cdot \begin{pmatrix}
s \\
\ast
\end{pmatrix} = \begin{pmatrix}
q \ast + A \\
0
\end{pmatrix} = \begin{pmatrix}
b \\
0
\end{pmatrix} + \begin{pmatrix}
-e \\
s
\end{pmatrix} \mod q.
\]

After lattice reduction on the lattice \(L(B)\), we perform enumeration around the target point \((b, 0)\) which, with some probability, will return the lattice point separated by \((-e, s)\), allowing for recovery of the LWE secret.

Typically, the cost of lattice reduction and decoding are balanced, and the output BDD probability determines the number of times the algorithm is repeated. The total expected running time is:

\[
T_{\text{Dec}} = T_{\text{BKZ}}(\beta_{\text{Dec}}, d) + T_{\text{bdd}}(\eta_{\text{Dec}}) \cdot p_{\text{babai} \cdot p_{\text{bdd}}}.
\]

Lattice reduction is carried out on the full lattice with block size \(\beta_{\text{Dec}}\), and a BDD solver is used on a projected sublattice of dimension \(\eta_{\text{Dec}}\), where \(\eta_{\text{Dec}}\) can be determined using Equation 3.2. Here \(p_{\text{babai}}\) is the probability of lifting the candidate solution from \(\pi_{d-\eta_{\text{Dec}}+1}(L(B))\) to the full lattice. Since \(\eta_{\text{Dec}}\) is determined using our variant of the condition from [ADPS16] we have \(p_{\text{babai}} \approx 1\) [AGVW17]. For our running example parameter set, assuming the GSA, this approach has a cost of \(2^{293.0}\) ring-operations with an optimal blocksize \(\beta_{\text{Dec}} = 419\), where \(\eta_{\text{Dec}} = 429\).

3.4.2 Drop-and-solve Decoding

The drop-and-solve decoding approach for small-secret LWE combines the decoding approach with dimension reduction techniques [MS01]. The attack begins by constructing a lattice for which the vector \((b, 0)\) is hopefully separated by the short vector \((-e, s_{(\tau)})\) to the lattice...
3.4 A Spectrum of Decoding Approaches for Solving Small-secret LWE

point \((A_{(\tau)}s_{(\tau)} \mod q, s_{(\tau)})\). The basis of this lattice is given by the columns of the matrix \(B\), where:

\[
B = \begin{pmatrix}
qI_m & A_{(\tau)} \\
0 & I_{n-\tau}
\end{pmatrix}.
\]

Assuming that we have correctly guessed \(\tau\) zero components, we have that:

\[
\begin{pmatrix}
qI_m & A_{(\tau)} \\
0 & I_r
\end{pmatrix} \begin{pmatrix}
* \\
s_{(\tau)}
\end{pmatrix} = \begin{pmatrix}
q^* + A_{(\tau)}s_{(\tau)} \\
s_{(\tau)}
\end{pmatrix} = \begin{pmatrix}
b \\
0
\end{pmatrix} + \begin{pmatrix}
-e \\
s_{(\tau)}
\end{pmatrix} \mod q.
\]

In this approach, we guess \(\tau\) zero components of \(s\) and then run the decoding attack in dimension \((d - \tau)\) \([MS01, Alb17, ACD+18]\). If we are unsuccessful, we restart with a fresh guess for the positions of zeros.

The core idea is that the lower running time of the dimension-reduced problem will trade-off positively against the probability of guessing zeros. If we correctly guess, for example, that the first \(\tau\) components of \(s\) are all zeros, then \((s_{\tau+1}, \ldots, s_n, e)\) can be found via solving BDD in the dimension-reduced lattice. The total expected running time of this strategy is:

\[
T_{dsDec} = \frac{T_{BKZ}(\beta_{dsDec}, d - \tau) + T_{bdd}(\eta_{dsDec})}{p_{babai} \cdot p_{bdd} \cdot p_0},
\]

where \(p_0\) is the probability of correctly guessing \(\tau\) zero components in the LWE secret \(s\). Lattice reduction is carried out on a lattice of dimension \((d - \tau)\) with block size \(\beta_{dsDec}\), and enumeration is carried out in the projected sublattice in dimension \(\eta_{dsDec}\), as outlined in Section 3.2. For our running example parameter set, assuming the GSA, this attack returns a complexity of \(2^{208.2}\) ring-operations with optimal values of \(\beta_{dsDec} = 170\) and \(\tau = 315\).

3.4.3 Guess-and-verify Decoding

As in the drop-and-solve decoding approach, our guess-and-verify approach begins by performing lattice reduction on a lattice basis of the form:

\[
B = \begin{pmatrix}
qI_m & A_{(\tau)} \\
0 & I_{n-\tau}
\end{pmatrix},
\]

and combines this technique with the following algebraic observation: if our initial zero guess is incorrect, we can account for this by decoding on multiple points against the same lattice.
basis. For example, after decoding on the point \((b, 0)\) with unsuccessful verification, we can begin checking all Hamming weight one guesses by decoding on the \(2 \cdot \binom{\tau}{1}\) points of the form:

\[(b \pm A_i, 0)\text{ for all } 1 \leq i \leq \tau,\]

and then all Hamming weight two guesses by decoding on the \(2^2 \cdot \binom{\tau}{2}\) points of the form:

\[(b \pm A_i \pm A_j, 0)\text{ for all } 1 \leq i, j \leq \tau, i \neq j,\]

and repeating this process up to Hamming weight \(\psi\) for some optimal value \(\psi \leq \min(h, \tau)\). This allows us to make multiple guesses per lattice reduction step, exactly as in the hybrid-decoding attack. There is no obvious reason to restrict the BDD solver in any guess-and-verify decoding attack to Babai’s Nearest Plane algorithm (as in the hybrid-decoding attack). Instead we may employ a stronger BDD solver, which in turn permits a reduction in the cost of preprocessing or the usage of a lower guessing dimension.

In this \textit{g-v} decoding attack, we consider a BDD dimension as defined by a variant of the success condition from [ADPS16], as outlined in Section 3.2. The overall expected cost of this approach is:

\[T_{gv\text{Dec}} = T_{\text{BKZ}}(\beta_{gv\text{Dec}}, d - \tau) + \|S_{tgt\text{Dec}}\| \cdot T_{\text{BDD}}(\eta_{gv\text{Dec}}) \cdot \frac{p_{babai} \cdot PBDD \cdot \left(\sum_{i=0}^{t_{gv\text{Dec}}} p_i\right)}{p_{babai}},\]

where \(p_i\) is the probability that the guessed sub-vector of the secret \(s\) has Hamming weight \(i\), and \(\|S_{tgt\text{Dec}}\|\) is the size of the guessing set, i.e. the number of target points decoded against. For our running example parameter set, assuming the GSA, this attack returns a complexity of \(2^{186.1}\) ring-operations with \(\beta_{gv\text{Dec}} = 225\) and \(\tau = 335\), with optimal choices of \(\eta_{gv\text{Dec}} = 49\) \(t_{gv\text{Dec}} = 16\) so that \(\|S_{tgt\text{Dec}}\| = \sum_{i=0}^{16} \binom{335}{i} \cdot 2^i \approx 2^{105.5}\).

We note that “guess-and-verify” decoding encompasses the usual decoding strategy \((\tau = 0, \|S_{tgt\text{Dec}}\| = 1, \sum p_i = 1)\) and the “drop-and-solve” strategy \((\tau > 0, \|S_{tgt\text{Dec}}\| = 1, t_{gv\text{Dec}} = 0)\). On the other hand, as specified here, it does not encompass the hybrid-decoding attack, (even without time-memory trade-offs) since we insist on picking \((\beta, \eta)\) such that \(p_{babai} \approx 1\), which is typically not the case for optimal hybrid-decoding attack parameters.

To summarise, we present a table of results outlining the complexity and optimal parameters for the attack techniques considered in this section.
3.5 Target Parameter Sets and Estimates

<table>
<thead>
<tr>
<th>Technique</th>
<th>β</th>
<th>τ</th>
<th>Ring operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>uSVP</td>
<td>198</td>
<td>297</td>
<td>$2^{214.4}$</td>
</tr>
<tr>
<td>dual</td>
<td>223</td>
<td>315</td>
<td>$2^{209.6}$</td>
</tr>
<tr>
<td>decoding</td>
<td>419</td>
<td>n/a</td>
<td>$2^{293.0}$</td>
</tr>
<tr>
<td>drop-and-solve decoding</td>
<td>170</td>
<td>315</td>
<td>$2^{208.2}$</td>
</tr>
<tr>
<td>guess-and-verify decoding</td>
<td>225</td>
<td>335</td>
<td>$2^{186.1}$</td>
</tr>
</tbody>
</table>

Table 3.2: Complexity estimates for uSVP, dual, and various decoding techniques on our example parameter set with $n = 653, q = 4621, \sigma \approx \sqrt{\frac{2}{3}},$ and $\chi_s = B_{100}^-.$

3.5 Target Parameter Sets and Estimates

In this section we apply our techniques to parameter sets for the NTRU Prime [BCLv17] and Round5 [BGL+18] submissions to the NIST PQC standardisation process, and, in order to compare with previous works, we also target a sparse-secret parameter set, we consider an old parameter set previously used in the homomorphic encryption library HElib [HEl20], which was studied in [Alb17]. We compare our results against the LWE Estimator under the same assumptions, i.e. considering the cost models in Section 3.3.2 and the Geometric Series Assumption. We also present our results considering usage of the BKZ simulator. Full results can be seen in Tables 3.7, 3.8, 3.9 and 3.10. In each section, we present a subset of the results for a classical guessing approach (i.e. exhaustive search).

3.5.1 NTRU Prime

We consider the NTRU LPrime parameter set from [BCLv19]. The construction is based on LWE with a fixed Hamming weight ternary secret and a random ternary error. Specifically, the parameter set considered is:

$$n = 761, q = 4591, \sigma \approx \sqrt{2/3}, \chi_s = B_{250}^-.$$

In the enumeration regime, when the output basis shape is determined by the BKZ simulator, g-v decoding outperforms the non-mitm hybrid-decoding attack by approximately 23-bits, as

\footnote{For up-to-date information on parameter selection in the homomorphic encryption library HElib, we refer the reader to [HS20].}
3.5 Target Parameter Sets and Estimates

can be seen in Table 3.3. In the sieving regime, solving Batch-BDD is less efficient than solving BDD (i.e. combinatorics do not improve the running time). Our techniques are thus not interesting in this regime, for this parameter set.

<table>
<thead>
<tr>
<th>Attack</th>
<th>GSO Profile</th>
<th>Enumeration</th>
<th>Sieving</th>
</tr>
</thead>
<tbody>
<tr>
<td>(non-mitm) hybrid-decoding</td>
<td>GSA</td>
<td>325.7</td>
<td>220.8</td>
</tr>
<tr>
<td>g-v decoding</td>
<td>GSA</td>
<td>337.6</td>
<td>181.0†</td>
</tr>
<tr>
<td>(non-mitm) hybrid-decoding</td>
<td>Simulator</td>
<td>385.3</td>
<td>303.3</td>
</tr>
<tr>
<td>g-v decoding</td>
<td>Simulator</td>
<td>362.1</td>
<td>185.3†</td>
</tr>
</tbody>
</table>

Table 3.3: Summary of results for NTRU Prime for a classical guessing (i.e. exhaustive search) approach. Estimates marked with † correspond to standard BDD decoding. Full results can be found in Tables 3.7, 3.8, 3.9, and 3.10.

3.5.2 Round5

For Round5, we consider the NIST level 3 parameter set from [GZB+19]. Round 5 is based on the Learning with Rounding problem (LWR) with a ternary, fixed hamming weight, secret. In the case of LWR, we have another parameter $p$ which is an additional modulus considered in the deterministic rounding process. In this case, we can set $\sigma \approx \sqrt{(q/p)^2 - 1}$ as in [ACD+18]. We therefore model this parameter set as LWE with:

$$n = 756, \sigma \approx 4.61, q = 2^{12}, p = 2^8, \chi_s = B_{242}^-.$$  

We also consider the IoT specific use-case parameter set from [GZB+19]. We can model this parameter set as LWE with:

$$n = 372, \sigma \approx 4.61, q = 2^{11}, p = 2^7, \chi_s = B_{178}^-.$$  

In the enumeration regime, when the output basis shape is determined by the BKZ Simulator, g-v decoding outperforms the non-mitm hybrid-decoding attack for the NIST level 3 parameter set by approximately 16-bits, and the IoT parameter set by approximately 19-bits, as can be seen in Tables 3.3 and 3.5. In the sieving regime, for both parameter sets, solving Batch-BDD is less efficient than solving BDD (i.e. combinatorics do not improve the running time). Our techniques are thus not interesting in this regime, for this parameter set.
3.5 Target Parameter Sets and Estimates

<table>
<thead>
<tr>
<th>Attack</th>
<th>GSO Profile</th>
<th>Enumeration</th>
<th>Sieving</th>
</tr>
</thead>
<tbody>
<tr>
<td>(non-mitm) hybrid-decoding</td>
<td>GSA</td>
<td>392.5</td>
<td>275.2</td>
</tr>
<tr>
<td>g-v decoding</td>
<td>GSA</td>
<td>416.9</td>
<td>218.9†</td>
</tr>
<tr>
<td>(non-mitm) hybrid-decoding</td>
<td>Simulator</td>
<td>466.6</td>
<td>394.9</td>
</tr>
<tr>
<td>g-v decoding</td>
<td>Simulator</td>
<td>449.6</td>
<td>224.1†</td>
</tr>
</tbody>
</table>

Table 3.4: Summary of results for Round 5 for a classical guessing (i.e. exhaustive search) approach. Estimates marked with † correspond to standard BDD decoding. Full results can be found in Tables 3.7, 3.8, 3.9, and 3.10.

<table>
<thead>
<tr>
<th>Attack</th>
<th>GSO Profile</th>
<th>Enumeration</th>
<th>Sieving</th>
</tr>
</thead>
<tbody>
<tr>
<td>(non-mitm) hybrid-decoding</td>
<td>GSA</td>
<td>205.5</td>
<td>156.9</td>
</tr>
<tr>
<td>g-v decoding</td>
<td>GSA</td>
<td>214.3</td>
<td>122.4†</td>
</tr>
<tr>
<td>(non-mitm) hybrid-decoding</td>
<td>Simulator</td>
<td>240.6</td>
<td>205.6</td>
</tr>
<tr>
<td>g-v decoding</td>
<td>Simulator</td>
<td>221.4</td>
<td>124.3†</td>
</tr>
</tbody>
</table>

Table 3.5: Summary of results for Round 5 IoT for a classical guessing (i.e. exhaustive search) approach. Estimates marked with † correspond to standard BDD decoding. Full results can be found in Tables 3.7, 3.8, 3.9, and 3.10.

3.5.3 HElib

We also consider our approach in the context of a sparse LWE parameter set, in order to gauge the performance of g-v decoding for sparse secrets. To compare with previous works, we consider the sparse-secret parameter set outlined in [Alb17]. Specifically, the parameter set we consider is:

\[ n = 1024, q = 2^{47}, \sigma \approx 3.19, \chi_s = B_{64}^- \]

Since the optimal blocksize \( \beta \) satisfies \( \beta \ll m + n \), the output of the BKZ simulator is very close to the GSA and thus the results in each case are similar, as can be seen in Table 3.6. We can see in Tables 3.7 and 3.9 that the results for g-v decoding and non-mitm hybrid-decoding are essentially the same.
3.5 Target Parameter Sets and Estimates

<table>
<thead>
<tr>
<th>Attack</th>
<th>GSO Profile</th>
<th>Enumeration</th>
<th>Sieving</th>
</tr>
</thead>
<tbody>
<tr>
<td>(non-mitm) hybrid-decoding</td>
<td>GSA</td>
<td>69.9</td>
<td>69.8</td>
</tr>
<tr>
<td>g-v decoding</td>
<td>GSA</td>
<td>69.1</td>
<td>69.8</td>
</tr>
</tbody>
</table>

Table 3.6: Summary of results for HElib for a classical guessing (i.e. exhaustive search) approach. Full results can be found in Tables 3.7, 3.8, 3.9, and 3.10.

3.5.4 Results and Notation

In the Tables 3.7, 3.8, 3.9, and 3.10, \( \tau \) is the (fixed) optimal guessing dimension, \( \beta \) is optimal the blocksize used in lattice reduction, \( \eta \) is the enumeration dimension considered, BDD cost is the total cost of solving the candidate BDD instances in the dimension \( \eta \) projected sublattice, \(|S|\) is the size of the search space considered, i.e. the number of points on which we decode, \( d \) is the dimension of the lattice considered, \#pp denotes the maximal Hamming weight considered in the search space, and rop is the cost of running the algorithm in ring-operations. Note that “g-v decoding” is the technique described in this work. Where meaningful, we highlight the “best in class” values in bold. Finally, we note that the “\( \lambda \)” values outline the security claims of each scheme, considering similar (pre-quantum) cost models and (pre-quantum) attacks; we note that such values of \( \lambda \) can be generated using vastly different assumptions.

3.5.5 Results in the Enumeration Regime

Tables 3.7 and 3.8 represent results in the enumeration regime, for a classical guessing strategy and a (square-root) meet-in-the-middle guessing strategy, respectively. Here, if the GSA is assumed, then the hybrid-decoding attack (typically) outperforms g-v decoding. Considering NTRU LPrime in Table 3.7, we see a complexity of 325.7-bits for the non-mitm hybrid attack and a complexity of 337.6-bits for the g-v decoding attack. If the BKZ simulator is assumed, then g-v decoding (typically) outperforms the hybrid-decoding attack in the case of a “classical” guessing approach, but not in the case of a square-root meet-in-the-middle approach. Considering NTRU LPrime in Table 3.7, we see a complexity of 385.3-bits for the non-mitm hybrid attack, and a complexity of 362.1-bits for the g-v decoding attack.

This tells us that, under certain assumptions, g-v decoding outperforms a variant of the
### 3.5 Target Parameter Sets and Estimates

| attack                  | \( \tau \) | \( \beta \) | \( \eta \) | BDD cost | \( |S| \) | repeats | \( d \) | \#pp | \( \log_2(\text{rop}) \) |
|-------------------------|----------|----------|----------|---------|-------|--------|-----|-----|-----------------------------|
| **NTRU LPrime**: \( n = 761, q = 4591, \sigma = \sqrt{2/3}, h = 250 \) \( \lambda = 222 \) |           |          |          |         |       |        |     |     |                             |
| uSVP (GSA)              | 92       | 458      | 458      | n/a     | 1     | 2\text{57.2} | 1220 | n/a | 384.6                         |
| Dual (GSA)              | 69       | 495      | n/a      | n/a     | n/a   | 2\text{320.9} | 1281 | 11  | 374.0                         |
| g-v decoding (GSA)      | 285      | 430      | 102      | 2\text{336.1} | 2\text{252.8} | 2\text{34.1} | 1026 | 55  | 337.6                         |
| non-mitm hybrid (GSA)   | 275      | 400      | n/a      | 2\text{324.1} | 2\text{255.6} | 2\text{49.6} | 1036 | 57  | 325.7                         |
| g-v decoding            | 225      | 435      | 272      | 2\text{360.9} | 2\text{146.4} | 2\text{53.9} | 1086 | 28  | 362.1                         |
| non-mitm hybrid         | 305      | 395      | n/a      | 2\text{384.3} | 2\text{252.3} | 2\text{113.1} | 1006 | 53  | 385.3                         |
| **Round5**: \( n = 756, q = 2^{12}, \sigma \approx 4.61, h = 242 \) \( \lambda = 270 \) |           |          |          |         |       |        |     |     |                             |
| uSVP (GSA)              | 230      | 449      | 449      | n/a     | 1     | 2\text{160.1} | 936  | n/a | 478.9                         |
| Dual (GSA)              | 63       | 626      | n/a      | n/a     | n/a   | 2\text{113.5} | 1227 | 19  | 489.2                         |
| g-v decoding (GSA)      | 365      | 490      | 117      | 2\text{415.2} | 2\text{297.9} | 2\text{60.2} | 814  | 62  | 416.9                         |
| non-mitm hybrid (GSA)   | 335      | 445      | n/a      | 2\text{391.0} | 2\text{295.7} | 2\text{76.8} | 844  | 64  | 392.5                         |
| g-v decoding            | 290      | 490      | 320      | 2\text{448.2} | 2\text{157.2} | 2\text{92.6} | 889  | 28  | 449.6                         |
| non-mitm hybrid         | 365      | 420      | n/a      | 2\text{465.5} | 2\text{274.2} | 2\text{172.9} | 814  | 55  | 466.6                         |
| **Round5 (IoT)**: \( n = 372, q = 2^{11}, \sigma \approx 4.61, h = 178 \) \( \lambda = 129 \) |           |          |          |         |       |        |     |     |                             |
| uSVP (GSA)              | 0        | 335      | 335      | n/a     | n/a   | 1     | 682  | n/a | 220.0                         |
| Dual (GSA)              | 32       | 334      | n/a      | n/a     | n/a   | 2\text{174.7} | 661  | 14  | 221.7                         |
| g-v decoding (GSA)      | 65       | 315      | 224      | 2\text{213.1} | 2\text{79.2} | 2\text{9.0} | 616  | 22  | 214.3                         |
| non-mitm hybrid (GSA)   | 115      | 270      | n/a      | 2\text{203.6} | 2\text{149.5} | 2\text{36.9} | 566  | 43  | 205.5                         |
| g-v decoding            | 50       | 320      | 266      | 2\text{220.4} | 2\text{51.6} | 2\text{12.8} | 631  | 13  | 221.4                         |
| non-mitm hybrid         | 120      | 270      | n/a      | 2\text{239.6} | 2\text{150.8} | 2\text{71.5} | 561  | 42  | 240.6                         |
| **HElib-1024**: \( n = 1024, q = 2^{47}, \sigma \approx 3.19, h = 64 \) |           |          |          |         |       |        |     |     |                             |
| uSVP (GSA)              | 140      | 105      | 105      | n/a     | 1     | 2\text{14.0} | 1670 | n/a | 75.5                          |
| Dual (GSA)              | 189      | 107      | n/a      | n/a     | n/a   | 2\text{223.3} | 1680 | 7   | 68.4                          |
| g-v decoding (GSA)      | 185      | 100      | 48       | 2\text{66.7} | 2\text{29.5} | 2\text{9.9} | 1624 | 4   | 69.1                          |
| non-mitm hybrid (GSA)   | 210      | 100      | n/a      | 2\text{67.5} | 2\text{36.6} | 2\text{10.7} | 1599 | 5   | 69.9                          |

Table 3.7: Estimates in the enumeration regime, where BKZ and the BDD solver are instantiated with enumeration algorithms.
### 3.5 Target Parameter Sets and Estimates

| Attack | $\tau$ | $\beta$ | $\eta$ | BDD cost | $|S|$ | repeats | $d$ | #pp | $\log_2(\text{rop})$ |
|--------|--------|--------|--------|----------|------|--------|----|-----|------------------|
| NTRU LPrime: $n = 761$, $q = 4591$, $\sigma = \sqrt{2/3}$, $h = 250$ |         |        |        |          |      |        |    |     |                  |
| sqrt g-v decoding (GSA) | 370 | 350 | 43 | $2^{243.5}$ | $2^{412.2}$ | $2^{11.5}$ | 941 | 102 | 245.0 |
| sqrt hybrid (GSA) | 360 | 335 | n/a | $2^{240.1}$ | $2^{402.8}$ | $2^{20.0}$ | 951 | 100 | 241.3 |
| sqrt g-v decoding | 370 | 380 | 119 | $2^{273.6}$ | $2^{400.0}$ | $2^{15.5}$ | 941 | 97  | 274.7 |
| sqrt hybrid | 395 | 350 | n/a | $2^{274.9}$ | $2^{428.3}$ | $2^{42.1}$ | 916 | 104 | 275.9 |
| Round5: $n = 756$, $q = 2^{12}$, $\sigma \approx 4.61$, $h = 242$ |         |        |        |          |      |        |    |     |                  |
| sqrt g-v decoding (GSA) | 445 | 395 | 37 | $2^{283.9}$ | $2^{490.0}$ | $2^{14.1}$ | 734 | 120 | 285.5 |
| sqrt hybrid (GSA) | 425 | 365 | n/a | $2^{277.0}$ | $2^{454.9}$ | $2^{32.0}$ | 754 | 109 | 278.0 |
| sqrt g-v decoding | 450 | 430 | 131 | $2^{324.8}$ | $2^{474.6}$ | $2^{22.7}$ | 729 | 113 | 325.9 |
| sqrt hybrid | 460 | 390 | n/a | $2^{320.5}$ | $2^{496.6}$ | $2^{54.3}$ | 719 | 120 | 321.6 |
| Round5 (IoT): $n = 372$, $q = 2^{11}$, $\sigma \approx 4.61$, $h = 178$ |         |        |        |          |      |        |    |     |                  |
| sqrt g-v decoding (GSA) | 175 | 250 | 48 | $2^{156.7}$ | $2^{250.8}$ | $2^{4.0}$ | 506 | 80  | 157.8 |
| sqrt hybrid (GSA) | 165 | 225 | n/a | $2^{151.6}$ | $2^{234.5}$ | $2^{17.4}$ | 516 | 74  | 152.9 |
| g-v decoding | 170 | 270 | 101 | $2^{174.3}$ | $2^{237.2}$ | $2^{6.5}$ | 511 | 73  | 175.4 |
| sqrt hybrid | 180 | 240 | n/a | $2^{172.2}$ | $2^{256.4}$ | $2^{27.1}$ | 501 | 81  | 173.4 |
| HElib-1024: $n = 1024$, $q = 2^{47}$, $\sigma \approx 3.19$, $h = 64$ |         |        |        |          |      |        |    |     |                  |
| sqrt g-v decoding (GSA) | 210 | 95  | 36 | $2^{50.9}$ | $2^{50.7}$ | $2^{5.2}$ | 1599 | 9   | 62.0  |
| sqrt hybrid (GSA) | 270 | 85  | n/a | $2^{60.8}$ | $2^{63.1}$ | $2^{9.1}$ | 1539 | 9   | 61.8  |

Table 3.8: Estimates in the enumeration regime considering a “meet-in-the-middle” approach which does not consider probabilities of failure in the meet-in-the-middle phase. Such an approach considers a square-root speed-up in the guessing phase.


| Attack          | \( \tau \) | \( \beta \) | \( \eta \) | BDD Cost | \(|S|\) | Repeats | \(d\) | \#pp \(\log_2(\text{rop})\) |
|-----------------|-----------|-----------|-----------|----------|------|---------|-----|-----------------|
| **NTRU LPrime**: \(n = 761, q = 4591, \sigma = \sqrt{2/3}, h = 250\) | \(\lambda = 155\) |
| uSVP (GSA)      | 0         | 532       | 532       | n/a      | n/a  | 1       | 1352| n/a 185.1       |
| Dual (GSA)      | 45        | 586       | n/a       | n/a      | n/a  | 2\(^{148.0}\) | 1383| 14 203.1         |
| g-v decoding (GSA) | 0         | 515       | 549       | 2\(^{179.7}\) | 1   | 1       | 1351| n/a 181.0†       |
| non-mitm hybrid (GSA) | 170       | 580       | n/a       | 2\(^{218.9}\) | 2\(^{217.8}\) | 2\(^{21.4}\) | 1181| 43 220.8         |
| g-v decoding    | 0         | 530       | 562       | 2\(^{183.5}\) | 1   | 1       | 1351| n/a 185.3†       |
| non-mitm hybrid | 230       | 615       | n/a       | 2\(^{302.1}\) | 2\(^{189.6}\) | 2\(^{63.3}\) | 1121| 40 303.3         |
| **Round5**: \(n = 756, q = 2^{12}, \sigma \approx 4.61, h = 242\) | \(\lambda = 193\) |
| uSVP (GSA)      | 0         | 664       | 664       | n/a      | n/a  | 1       | 1266| n/a 223.6       |
| Dual (GSA)      | 46        | 748       | n/a       | n/a      | n/a  | 2\(^{198.6}\) | 1325| 13 251.0         |
| g-v decoding (GSA) | 0         | 645       | 679       | 2\(^{217.7}\) | 1   | 1       | 1265| n/a 218.9†       |
| non-mitm hybrid (GSA) | 225       | 705       | n/a       | 2\(^{273.7}\) | 2\(^{215.4}\) | 2\(^{39.3}\) | 1040| 49 275.2         |
| g-v decoding    | 0         | 660       | 699       | 2\(^{223.5}\) | 1   | 1       | 1265| n/a 224.1†       |
| non-mitm hybrid | 290       | 700       | n/a       | 2\(^{393.9}\) | 2\(^{214.8}\) | 2\(^{160.3}\) | 975 | 43 394.9         |
| **Round5 (IoT)**: \(n = 372, q = 2^{11}, \sigma \approx 4.61, h = 178\) | \(\lambda = 96\) |
| uSVP (GSA)      | 0         | 335       | 335       | n/a      | n/a  | 1       | 682 | n/a 126.6       |
| Dual (GSA)      | 22        | 396       | n/a       | n/a      | n/a  | 2\(^{104.2}\) | 710 | 1 145.0          |
| g-v decoding (GSA) | 0         | 315       | 349       | 2\(^{121.6}\) | 1   | 1       | 681 | n/a 122.4†       |
| non-mitm hybrid (GSA) | 85         | 375       | n/a       | 2\(^{155.2}\) | 2\(^{119.5}\) | 2\(^{18.3}\) | 596 | 38 156.9         |
| sqrt g-v decoding | 0         | 320       | 358       | 2\(^{123.9}\) | 1   | 1       | 681 | n/a 124.3†       |
| non-mitm hybrid | 95         | 380       | n/a       | 2\(^{204.5}\) | 2\(^{122.1}\) | 2\(^{65.1}\) | 586 | 35 205.6         |
| **HElib-1024**: \(n = 1024, q = 2^{47}, \sigma \approx 3.19, h = 64\) | |
| uSVP (GSA)      | 0         | 137       | 137       | n/a      | n/a  | 1       | 1939| n/a 70.3        |
| Dual (GSA)      | 80        | 115       | n/a       | n/a      | n/a  | 2\(^{19.6}\) | 1741| 7 67.1          |
| g-v decoding (GSA) | 85         | 125       | 50        | 2\(^{66.6}\) | 2\(^{30.0}\) | 2\(^{2.6}\) | 1853| 5 69.8          |
| non-mitm hybrid (GSA) | 155       | 115       | n/a       | 2\(^{66.2}\) | 2\(^{40.1}\) | 2\(^{5.6}\) | 1783| 6 69.8          |

Table 3.9: Estimates in the sieving regime, where BKZ and the BDD solver are instantiated with sieving algorithms. Estimates marked with † correspond to standard BDD decoding.
Table 3.10: Estimates in sieving regime for a “meet-in-the-middle” approach which does not consider probabilities of failure in the meet-in-the-middle phase. Such an approach considers a square-root speed-up in the guessing phase.
hybrid-decoding attack. In order to compare directly to the (full) hybrid-decoding attack, a proper analysis is required of the meet-in-the-middle probability for the g-v decoding approach, and this is left to future work.

3.5.6 Results in the Sieving Regime

Tables 3.9 and 3.10 represent results in the sieving regime, for a classical guessing strategy and a (square-root) meet-in-the-middle guessing strategy, respectively. Here, the results are less interesting since combinatorics do not result in a complexity improvement for g-v decoding and, therefore, the g-v decoding results correspond to the standard decoding attack.

Finally, we note that some of the complexities for the uSVP and decoding attacks considered in these tables outperform the dual attack. Intuitively, since the dual attack solves decision-LWE and the uSVP and decoding attacks solve search-LWE, we might expect that the dual attack should have a lower complexity. This could highlight the fact that there may be potential improvements for the dual attack which have yet-to-be discovered, or, it could outline the fact that some of the assumptions used in our analysis of the hybrid attack, and the variant of the dual attack considered in the LWE Estimator, are mis-aligned. Moreover, we note that this behaviour can also be observed in a variety of other estimates in the literature. For example, there are several examples of this behaviour in the estimates at https://estimate-all-the-lwe-ntru-schemes.github.io which are discussed in detail in Chapter 4.

3.6 Assumptions Case Study: NTRU Prime

As discussed in Section 3.3.1 there are several points during a hybrid-decoding attack-based security analysis where assumptions are required. This can make comparing two analyses of the complexity of the hybrid-decoding attack cumbersome. In order to cross-check our hybrid-decoding attack estimates, we align our code with the assumptions made in the NTRU Prime security script. That is, we consider the set of assumptions $A_0$ outlined in Table 3.11. Explicitly, assumption set $A_0$ consists of the following assumptions:

1. core-style BKZ models (“pre-quantum sieving” (i.e. $2^{0.292/3}$) and “pre-quantum enumeration”
3.6 Assumptions Case Study: NTRU Prime

<table>
<thead>
<tr>
<th>Technique</th>
<th>Assumption</th>
<th>$A_0$</th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>Our work</th>
</tr>
</thead>
<tbody>
<tr>
<td>BKZ SVP calls</td>
<td>$1$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$8d$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>$P_{babai}$</td>
<td>$\prod_{1 \leq i \leq d} \left( 1 - \frac{2}{\beta^{i+1}} \int_{\min(r,1)}^{1} (1 - t^2)^{(d-3)/2} dt \right)$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>$T_{babai}$</td>
<td>$\frac{d^e}{\pi^e r}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>BKZ output shape</td>
<td>$q$-ary GSA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BKZ Simulator</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Guessing strategy</td>
<td>MiTM (sqrt)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Classic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Target norm</td>
<td>$\sqrt{m \sigma^2 + h \frac{n-r}{n}}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Lattice scaling</td>
<td>$s \mapsto \eta s : |s| \approx |e|$</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mitm probability</td>
<td>$1$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Memory considered?</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.11: Sets of assumptions considered in this case study.

(i.e. $2^{0.18728 \beta \log(\beta) - 1.019 \beta + 16.1}$), both with “free memory”, in the language of [BCLv19],

2. the formula for the success probability of Babai’s Nearest Plane algorithm from [Wun19],
   with an associated cost of one operation,

3. the $q$-ary GSA as the output basis shape of the BKZ algorithm,

4. a meet-in-the-middle guessing phase, with associated collision probability of one,

5. the target norm of the vector recovered via the BDD algorithm is $\sqrt{m \sigma^2 + h \frac{n-r}{n}}$, 

6. free memory, and

7. no lattice scaling is considered.

We note that [BCLv19] contains estimates which consider the cost of memory. We do not compare against these estimates as we do not consider memory costs in our analysis. After
considering the assumption set $\mathcal{A}_0$, we move through assumptions until we reach those used in our work. In particular, we consider several sets of assumptions: $\mathcal{A}_i$ for $0 \leq i \leq 4$.

1. Assumption set $\mathcal{A}_1$ corresponds to $\mathcal{A}_0$ with the $q$-ary GSA swapped for the BKZ simulator, since this is a more accurate measure of the GSO basis output shape given by BKZ.

2. Assumption set $\mathcal{A}_2$ corresponds to $\mathcal{A}_1$ with the cost of Babai's Nearest Plane algorithm altered from one operation to be polynomial in the dimension of the lattice, i.e. $\frac{d^2}{2^{\frac{\beta}{2}}} \cdot 06$ operations.

3. Assumption set $\mathcal{A}_3$ corresponds to $\mathcal{A}_2$ with the core-style cost models changed to cost models which consider eight tours. As an example, this means swapping from e.g. $T_{\text{BKZ}}(\beta, d) = 2^{0.292\beta + 16.4}$ to $T_{\text{BKZ}}(\beta, d) = 8d \cdot 2^{0.292\beta + 16.4}$.

4. Assumption set $\mathcal{A}_4$ corresponds to $\mathcal{A}_3$ with the guessing strategy changed from a meet-in-the-middle approach to a classical guessing strategy, allowing us to drop the inaccurate assumption that collisions occur with probability one.

5. The only difference between assumption set $\mathcal{A}_4$ and the assumptions considered in our work is that we consider lattice scaling.

These assumption sets are summarised in Table 3.11. We now compare the outputs of the NTRU Prime script\footnote{The code used in the NTRU Prime submission can be found here: \url{https://ntruprime.cr.yp.to/estimate-20190329.sage}} our script for the hybrid-decoding attack (labelled our script (hybrid)), and our script for the $g$-$v$ decoding attack (labelled our script ($g$-$v$ decoding)) under these various assumption sets, to see how the complexities of the techniques compare in each case. Table 3.12 outlines the complexities in the enumeration regime, and Table 3.13 outlines the complexities in the sieving regime. Note that we consider the same search spaces as the NTRU LPrime script in each case, i.e. $\beta \in \{40, 80, 120, \ldots \}$ and $\tau \in \{0, 40, 80, \ldots \}$. We note the closeness of the results under assumptions $\mathcal{A}_0$ between the NTRU Prime script and our script (hybrid) given in Tables 3.12 and 3.13 and comment that this lends confidence to the estimates given by our scripts. Moreover, our code is open source and can be found at the url listed in Section 1.6.
### 3.6 Assumptions Case Study: NTRU Prime

Table 3.12: Enumeration-based estimates, where each section corresponds to a set of assumptions outlined in Table 3.11. "–" denotes a value which is not compatible with our notation (for example, our script considers a simple sqrt speed-up in the search space, the NTRU Prime script considers splitting the search space as in a meet-in-the-middle approach).

| ass alg                                | $\tau$ | $\beta$ | $\eta$ | BDD cost | $|S|$ | repeats | $d$ | #pp | $\log_2(\text{rop})$ |
|----------------------------------------|--------|---------|--------|----------|------|---------|----|-----|-----------------|
| NTRU LPrime: $n = 761, q = 4591, \sigma = \sqrt{2/3}, h = 250$ |        |         |        |          |      |         |    |     |                 |
| NTRU Prime script                      | 360    | 320     | n/a    | $2^{220.9}$ | –    | $2^{42.5}$ | 881 | –   | 222.1           |
| $A_0$ our script (hybrid)              | 360    | 320     | n/a    | $2^{221.3}$ | $2^{375.4}$ | $2^{33.6}$ | 951 | 89  | **222.9**       |
| our script (g-v decoding)              | 360    | 360     | 83     | $2^{239.0}$ | $2^{380.5}$ | $2^{18.1}$ | 951 | 91  | 240.5           |
| $A_1$ our script (hybrid)              | 400    | 360     | n/a    | $2^{257.6}$ | $2^{442.8}$ | $2^{36.2}$ | 911 | 109 | **258.7**       |
| our script (g-v decoding)              | 360    | 400     | 139    | $2^{270.1}$ | $2^{390.6}$ | $2^{14.4}$ | 951 | 95  | 271.2           |
| $A_2$ our script (hybrid)              | 400    | 360     | n/a    | $2^{274.0}$ | $2^{404.5}$ | $2^{52.2}$ | 911 | 94  | 274.5           |
| our script (g-v decoding)              | 360    | 400     | 139    | $2^{270.1}$ | $2^{390.6}$ | $2^{14.4}$ | 951 | 95  | **271.2**       |
| $A_3$ our script (hybrid)              | 400    | 360     | n/a    | $2^{276.2}$ | $2^{442.8}$ | $2^{36.2}$ | 911 | 109 | **277.9**       |
| our script (g-v decoding)              | 360    | 400     | 139    | $2^{283.6}$ | $2^{409.8}$ | $2^{8.7}$ | 951 | 103 | 284.9           |
| $A_4$ our script (hybrid)              | 320    | 400     | n/a    | $2^{386.9}$ | $2^{256.3}$ | $2^{111.8}$ | 991 | 53  | 388.1           |
| our script (g-v decoding)              | 240    | 440     | 280    | $2^{476.0}$ | $2^{141.3}$ | $2^{68.1}$ | 1071| 26  | **380.0**       |
### 3.6 Assumptions Case Study: NTRU Prime

Table 3.13: Sieving-based estimates, where each section corresponds to a set of assumptions outlined in Table 3.11. “–” denotes a value which is not compatible with our notation (for example, our script considers a simple sqrt speed-up in the search space, the NTRU Prime script considers splitting the search space as in a meet-in-the-middle approach). Estimates marked with a † correspond to a uSVP estimate with $\eta, \beta$ uncoupled.

| ass alg | $\tau$ | $\beta$ | $\eta$ | BDD cost | $|S|$ | repeats | $d$ | #$\text{pp}$ | $\log_2(\text{rop})$ |
|---------|---------|---------|--------|----------|------|---------|-----|----------|----------------|
| NTRU Prime script | 240 | 480 | n/a | $2^{156.0}$ | – | $2^{18.8}$ | 1081 | – | 159.4 |
| $A_0$ | our script (hybrid) | 240 | 480 | n/a | $2^{158.1}$ | $2^{279.7}$ | $2^{18.3}$ | 1111 | 72 | **159.4** |
| | our script (g-v decoding) | 0 | 560 | 562 | $2^{164.1}$ | 1 | 1 | 1351 | 0 | 165.0† |
| $A_1$ | our script (hybrid) | 320 | 600 | n/a | $2^{209.9}$ | $2^{348.2}$ | $2^{35.8}$ | 1031 | 85 | 211.5 |
| | our script (g-v decoding) | 0 | 560 | 593 | $2^{174.2}$ | 1 | 1 | 1351 | 0 | **173.2†** |
| $A_2$ | our script (hybrid) | 360 | 320 | n/a | $2^{221.3}$ | $2^{375.4}$ | $2^{33.6}$ | 951 | 89 | 222.9 |
| | our script (g-v decoding) | 0 | 560 | 593 | $2^{174.2}$ | 1 | 1 | 1351 | 0 | **173.2†** |
| $A_3$ | our script (hybrid) | 320 | 600 | 0 | $2^{227.2}$ | $2^{338.2}$ | $2^{39.1}$ | 1031 | 81 | 228.3 |
| | our script (g-v decoding) | 0 | 560 | 593 | $2^{176.2}$ | 1 | 1 | 1351 | 0 | **177.8†** |
| $A_4$ | our script (hybrid) | 200 | 640 | 0 | $2^{297.6}$ | $2^{180.7}$ | $2^{97.6}$ | 1151 | 40 | 298.6 |
| | our script (g-v decoding) | 0 | 560 | 593 | $2^{176.2}$ | 1 | 1 | 1351 | 0 | **177.8†** |
3.7 Conclusion

There are two main observations from Tables 3.12 and 3.13. The first observation is that we believe assumption set $A_4$ represents assumptions which are more realistic than those used in the preceding assumption sets and, as we can see in the enumeration regime (Table 3.12), the g-v decoding attack outperforms (a variant of) the hybrid attack, under these assumptions, for this parameter set. In the sieving regime, this is not the case, and here g-v decoding corresponds to the standard BDD decoding approach.

The second observation is to highlight the differences that varying assumptions can make on attacks of this type. If we consider assumption sets $A_0$ and $A_3$ in the enumeration regime (Table 3.12), we note that the difference in complexity of the g-v decoding estimates is $\approx 285 - 241 = 44$-bits, and the difference in complexity of the hybrid attack estimates is $\approx 278 - 223 = 55$-bits. This shows the importance of outlining the assumptions used in any security analysis which considers decoding attacks.

3.7 Conclusion

In this chapter, we introduced a guess-and-verify decoding technique to solve the Small-secret Learning with Errors problem. Throughout, we have compared this technique with the hybrid-decoding attack. These two attack techniques consider a myriad of trade-offs and are trick to optimise, due to the large number of degrees of freedom in the attack parameters (e.g. $(\tau, \beta, m, |S|)$). Conservative assumptions can be made (such as those discussed throughout this chapter) in order to generate an underestimate of security fairly efficiently. This is a reasonable approach, provided that the process is transparent. However, it is also important to estimate the complexity of these attacks as accurately as possible. Throughout this chapter, we have considered a variety of sets of assumptions and have shown that, under certain assumptions our g-v decoding technique outperforms a (non-mitm) variant of the hybrid-decoding attack and, under other sets of assumptions, the converse is true.
# Security Estimates for the NIST Standardisation Process for Post-quantum Cryptographic Algorithms

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Introduction and Contribution</td>
<td>107</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Related Work</td>
<td>110</td>
</tr>
<tr>
<td>4.2</td>
<td>First Round Submissions</td>
<td>110</td>
</tr>
<tr>
<td>4.3</td>
<td>Costing Lattice Reduction</td>
<td>111</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Enumeration-based Cost Models</td>
<td>111</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Sieving-based Cost Models</td>
<td>113</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Cost Models Used in the Submissions</td>
<td>114</td>
</tr>
<tr>
<td>4.4</td>
<td>Parameter Sets</td>
<td>115</td>
</tr>
<tr>
<td>4.5</td>
<td>Small Secret Variants of the uSVP and Dual Attacks</td>
<td>121</td>
</tr>
<tr>
<td>4.5.1</td>
<td>uSVP</td>
<td>121</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Dual</td>
<td>122</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Multiple Hardness Assumptions</td>
<td>125</td>
</tr>
<tr>
<td>4.5.4</td>
<td>Number of Samples</td>
<td>125</td>
</tr>
<tr>
<td>4.6</td>
<td>First Round Security Estimates</td>
<td>126</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Observation: Cost Swaps</td>
<td>128</td>
</tr>
<tr>
<td>4.7</td>
<td>Second Round Submissions</td>
<td>131</td>
</tr>
<tr>
<td>4.8</td>
<td>The Third Round</td>
<td>131</td>
</tr>
<tr>
<td>4.8.1</td>
<td>Cost Models</td>
<td>131</td>
</tr>
<tr>
<td>4.8.2</td>
<td>Parameter Sets and Estimates</td>
<td>134</td>
</tr>
<tr>
<td>4.9</td>
<td>Conclusion</td>
<td>139</td>
</tr>
</tbody>
</table>

This chapter is based on the following publication: Martin R. Albrecht, Benjamin R. Curtis, Alex Davidson, Amit Deo, Rachel Player, Eamonn Postlethwaite, Fernando Virdia and Thomas Wunderer. Estimate all the \{LWE, NTRU\} schemes! 11th International Conference on
4.1 Introduction and Contribution

In 2015, the United States’ National Institute of Standards and Technology (NIST) began a process aimed at the standardisation of post-quantum public-key encryption schemes (PKE), key encapsulation mechanisms (KEM), and digital signature algorithms (SIG). The initial call for proposals was in 2016 [Nat16]. The aim of the process is to ensure that cryptographic requirements can be met in an era where quantum computers exist. Participants were invited to submit their designs along with an associated cryptanalysis, and parameter sets aimed at meeting one or more target security categories out of the five defined by NIST. The five security categories are listed as follows:

1. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 128-bit key (e.g. AES128).

2. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 256-bit hash function (e.g. SHA256/SHA3-256).

3. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 192-bit key (e.g. AES192).
4. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 384-bit hash function (e.g. SHA384/SHA3-384).

5. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 256-bit key (e.g. AES 256).

One way to interpret these security levels is the following: each AES-\(n\) security level corresponds to \(n\)-bits of classical security and \(n/2\)-bits of quantum security (based on the complexity of key search, i.e. \(O(2^n)\) classically and \(O(2^n/2)\) using Grover’s algorithm), and each SHA-\(n\) security level corresponds to \(n/2\)-bits of classical security and \(n/3\)-bits of quantum security (based on the complexity of collision search, i.e. \(O(2^n/2)\) classically and \(O(2^n/3)\) using Grover’s algorithm), leading to the security levels in Table 4.1.

<table>
<thead>
<tr>
<th>Security Level</th>
<th>Classical Security (bits)</th>
<th>Quantum Security (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (AES-128)</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>2 (SHA-256)</td>
<td>128</td>
<td>(\approx 85)</td>
</tr>
<tr>
<td>3 (AES-192)</td>
<td>192</td>
<td>96</td>
</tr>
<tr>
<td>4 (SHA-384)</td>
<td>192</td>
<td>128</td>
</tr>
<tr>
<td>5 (AES-256)</td>
<td>256</td>
<td>128</td>
</tr>
</tbody>
</table>

Table 4.1: An example interpretation of the five NIST security levels.

Different interpretations of these five different security levels, alongside the use of vastly different assumptions considered by each submission, means that the security (and efficiency) of any two schemes can be difficult to compare fairly. In particular, the *cost model* for the BKZ algorithm, as defined in Section 2.10.1.2, varies across submissions. This means that it is possible for two *identical* parameter sets to be associated to two different security levels \(\lambda_1\) and \(\lambda_2\), of which the difference can be significant. In this chapter, we are interested in the *lattice-based* submissions to the NIST standardisation process. We extract the proposed parameter sets, and lattice reduction cost models, used in each LWE-based and NTRU-based submission. To provide a clearer view on the effect of the chosen cost model, we cross-estimate the security of each parameter set under every cost model for all first round submissions.
4.1 Introduction and Contribution

The five security categories are defined as target security levels when considering adversaries who have access to a quantum computer. Furthermore, NIST propose the assumption that such a quantum computing device would support a maximum circuit depth $\text{MAXDEPTH} \leq 2^{96}$ \cite{Nat16}. We note that not all schemes take this limitation into account, and instead opt for using an asymptotic cost model that considers the best known theoretical Grover speed-up, resulting in overestimates of the adversary’s power.

This has caused confusion within the submissions and this confusion introduces further difficulties when making a comparison of two schemes under a (quantum) cost model. Consider category 1, which states that attacks on schemes should be at least as hard as AES-128 key recovery. Some schemes interpret this by generating parameter sets with $\lambda \geq 128$ under a quantum cost model, whereas other schemes claiming the same category of hardness interpret this to mean $\lambda \geq 64$, since key recovery for AES-128 can be completed in time $O(2^{n/2})$ using Grover’s algorithm. This results in schemes listing parameter sets with vastly different claimed security being in the same category. To make this clear we list the “claimed security” levels of all schemes in our tables of estimates.

We restrict our attention to the uSVP variant of the primal lattice attack \cite{ADPS16, AGVW17} and the dual lattice attack \cite{MR09}, and we recall that both of these techniques were introduced in Chapter 2. We note that, for certain schemes where the LWE secret is small and/or sparse, we consider the small-secret variants of the uSVP attack \cite{BG14} and dual attack \cite{Alb17} which we outline in Section 4.5. We do not consider algebraic \cite{AG11, ACFP14} or combinatorial \cite{AFFP14, GJS15, KF15, GJMS17} attacks, since those algorithms are not competitive for the parameter sets considered in this work.

We do not consider the primal attack via a combination of lattice reduction and BDD enumeration often referred to as a “lattice decoding” attack \cite{Sch03, LP11} as considered in Chapter 3. However, note that the primal uSVP attack can be considered as a simplified variant of the decoding attack in the enumeration regime. For NTRU, we restrict our attention to the primal uSVP attack (possibly combined with guessing zero-entries of the short vector). We do not consider the hybrid lattice reduction and meet-in-the-middle attack \cite{How07, Wun19} or “guessing + nearest plane” after lattice reduction, as considered in Chapter 3.
4.2 First Round Submissions

4.1.1 Related Work

NIST categorised each scheme according to the family of underlying problem (lattice-based, code-based, SIDH-based, MQ-based, hash-based, other) in [Moo17]. This analysis was refined in [Fuj17]. NIST then provided a first performance comparison of all complete and proper schemes in [Nat17]. Bernstein provided a comparison of all schemes based on the sizes of their ciphertexts and keys in [Ber17].

4.2 First Round Submissions

In total, 82 submissions were made to the standardisation process and, of these submissions, 69 were deemed “complete and proper” by NIST. These 69 submissions, of which 23 were based on either the LWE or NTRU family of problems, formed the first round submissions. The 18 LWE-based submissions and five NTRU-based submissions are outlined in Table 4.2.

<table>
<thead>
<tr>
<th>Assumption Schemes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crystals-Dilithium</td>
</tr>
<tr>
<td>Ding Key Exchange</td>
</tr>
<tr>
<td>Frodo</td>
</tr>
<tr>
<td>KCL</td>
</tr>
<tr>
<td>LWE Variants</td>
</tr>
<tr>
<td>LOTUS</td>
</tr>
<tr>
<td>NewHope</td>
</tr>
<tr>
<td>ThreeBears</td>
</tr>
<tr>
<td>uRound2</td>
</tr>
<tr>
<td>NTRU Variants</td>
</tr>
<tr>
<td>NTRUEncrypt</td>
</tr>
<tr>
<td>pqNTRUSign</td>
</tr>
</tbody>
</table>

Table 4.2: Complete and proper lattice-based submissions to the NIST standardisation process.
4.3 Costing Lattice Reduction

There are a variety of approaches within the literature to cost the running time of the BKZ algorithm \[ \text{[CN11, APS15, ADPS16].} \] The main divergence stems from whether cost models are based on using enumeration-based algorithms as the SVP solver (the enumeration regime), or sieving-based algorithms as the SVP solver (the sieving regime). A second divergence stems from how many calls to the SVP oracle are expected to be required to recover a vector of length \( \approx \delta^d \text{Vol}(\Lambda)^{1/d}. \)

The cost of BKZ with blocksize \( \beta \) on a lattice of dimension \( d \) can be written in the form:

\[
T_{BKZ}(\beta,d) = a \cdot 2^{f(\beta,d)} = 2^{f(\beta,d)+\log(a)},
\]

where \( a \) denotes the number of calls to an SVP oracle of cost \( 2^{f(\beta,d)} \). We refer to the exponent, \( f(\beta,d)+\log(a) \), as a **BKZ cost model**. In this section, we review the cost models used as part of all first round submissions. We note that the cost models considered in this chapter do not consider the **coefficient size**, that is the bit-length of the modulus \( q \). We assume that the output costs are in ring-operations, i.e. a number of operations in \( \mathbb{Z}_q \), but we do not consider the additional costs (e.g. in binary operations) brought by varying the value of \( q \). For the schemes considered in this work, where the moduli are reasonably close in size, this is not particularly an issue. However, for homomorphic encryption schemes, where modulus sizes larger than \( 2^{1000} \) are considered, it may be important to also consider the coefficient size in the cost of lattice reduction.

### 4.3.1 Enumeration-based Cost Models

Let \( d \) be the dimension of the lattice used inside BKZ with blocksize \( \beta \). In the literature, the cost of enumeration can be found to be estimated as \( 2^{c_1 \beta \log \beta + c_2 \beta + c_3} \) [Kan83, MW15] or as \( 2^{c_1 \beta^2 + c_2 \beta + c_3} \) [FPS15, CN11], with Grover speedups considered to half the exponent. There are four enumeration-based cost models used as part of the first round submissions to the NIST standardisation process. The estimates \( 0.187 \beta \log \beta - 1.019 \beta + 16.1 \) [APS15] and \( 0.000784 \beta^2 + 0.366 \beta - 0.9 \) [HPS+15] are both based on fitting the same data from [Che13]. LOTUS [PHAM17] is the only submission not to provide a closed formula for estimating the cost of BKZ. Given their preference for enumeration, we fit their estimated cost model...
4.3 Costing Lattice Reduction

Figure 4.1: Enumeration-based cost models used as part of a first round submission to the NIST standardisation process for a lattice of dimension $d = 1024$ with $40 \leq \beta \leq 400$.

to a curve of shape $2^{c_1 \beta \log \beta + c_2 \beta + c_3}$ following [MW15]. We fit a curve to the values given by [PHAM17, (39)], the script used is available in the public repository [Lot18]. To summarise, the four enumeration-based cost models considered in the first round submissions are:

1. $0.187 \beta \log(\beta) - 1.019 \beta + 16.1$,
2. $\frac{1}{2}(0.187 \beta \log(\beta) - 1.019 \beta + 16.1)$,
3. $0.000784 \beta^2 + 0.366 \beta - 0.9 + \log(8d)$, and
4. $0.125 \beta \log(\beta) - 0.755 \beta + 2.25$.

To illustrate the differences in these models, we plot the costs for a lattice of dimension $d = 1024$ for $40 \leq \beta \leq 400$ in Figure 4.1.
4.3 Costing Lattice Reduction

4.3.2 Sieving-based Cost Models

There are ten sieving-based cost models used as part of first round submissions to the NIST standardisation process. Recall that sieving algorithms require the usage of exponential memory. However, in our work, we do not consider the memory costs associated to sieving algorithms, and simply consider the time cost. Let $d$ be the dimension of the lattice used inside BKZ with blocksize $\beta$. The $0.292\beta$ model comes from \cite{BDGL16}, and the $0.265\beta$ model accounts for speed-ups associated Grover’s algorithm, and appears in \cite{Laa15a}. There is a min-space variant $0.368\beta$ which also appears in \cite{BDGL16}, where Grover speed-ups account for the $0.2975\beta$ model, occurring in \cite{Laa15a}. The constant $16.4$ comes from experiments in \cite{Laa15b}, interpolated by \cite{APS15}. The $\log(\beta)$ term appears in submissions which assume that a sieving cost model of the form $2^{c\beta}$ requires $\beta 2^{c^3}$ CPU cycles.

With respect to the number of SVP oracle calls required by BKZ, a popular choice was to follow the “Core-SVP” model introduced in \cite{ADPS16}, that considers a single call. Alternatively, the number of calls has also been estimated to be $8d$ (for example, in \cite{Alb17}), where $d$ is the dimension of the embedding lattice. To summarise, there are ten sieving-based cost models considered in the first round submissions, all of which are the result of combinations of the above assumptions. Of the ten cost models, the five classical models are:

1. $0.292\beta$,
2. $0.292\beta + 16.4$,
3. $0.368\beta$,
4. $0.292\beta + \log(\beta)$, and
5. $0.292\beta + 16.4 + \log(8d)$.

The five quantum models are:

6. $0.265\beta$,
7. $0.265\beta + 16.4$,
8. $0.2975\beta$,  

113
4.3 Costing Lattice Reduction

9. \(0.265\beta + \log(\beta)\), and

10. \(0.265\beta + 16.4 + \log(8d)\).

Figure 4.2: Sieving-based cost models, used as part of a first round submission to the NIST standardisation process, for a lattice of dimension \(d = 1024\) with \(40 \leq \beta \leq 400\).

4.3.3 Cost Models Used in the Submissions

In this section we match the models outlined in Section 4.3 to the first round submissions. Specifically, in Table 4.3 we outline the cost models considered by each individual submission. We note that some submissions consider several models, and cost their scheme under each model (for example, LIMA [SAL+17] considers both the \(0.292\beta + 16.4\) and \(0.265\beta + 16.4\) models).

The different cost models diverge on the unit of operations they are using. In the enumeration models, the unit is “number of nodes visited during enumeration”. As discussed in Chapter 3,
it is typically assumed that processing one node costs about 100 CPU cycles \cite{CN11}. For sieving, the elementary operation is typically an operation on word-sized integers, costing about one CPU cycle (recall, however, that the cost models including a \( \log(\beta) \) term assume that a sieving cost model of the form \( 2^{c\beta} \) requires \( \beta^{2c\beta} \) CPU cycles). For quantum algorithms the unit is typically the number of Grover iterations required, and it is not clear how this translates to traditional CPU cycles. Of course, for models which suppress lower order terms, the unit of computation considered is immaterial.

4.4 Parameter Sets

In this section we outline the individual parameter sets considered inside every submission. In this work we consider Learning with Rounding-based, Ring/Module LWE-based, and NTRU-based parameter sets.

Learning with Rounding. Recall from Chapter 2 that the Learning with Rounding (LWR) problem replaces the addition of a noise term (used in LWE) with a deterministic rounding process. An instance of the LWR problem is of the form:

\[
\left( a, b := \left\lfloor \frac{p}{q} (a, s) \right\rfloor \right) \in \mathbb{Z}_n^q \times \mathbb{Z}_p,
\]

and we can interpret this as an LWE instance by multiplying the second component by \( q/p \) and assuming that:

\[
(q/p) \cdot b = (a, s) + e,
\]

where \( e \) is chosen uniformly from the set \( \{ -\frac{q^2}{p} + 1, \ldots, \frac{q^2}{p} \} \) \cite{Ngu18}. We can therefore view LWR samples as LWE samples with modulus \( q \) and error distribution uniform over the set \( \{ -\frac{q^2}{p} + 1, \ldots, \frac{q^2}{p} \} \).

Ring/Module Learning with Errors. We view Ring-LWE and Module-LWE instances as LWE instances by considering the coefficients of elements in \( \mathbb{R}_q \) as vectors in \( \mathbb{Z}_q^n \) and ignoring any algebraic structure of \( R_q \). This approach is standard when considering the complexity of algorithms solving the Ring-LWE and Module-LWE problems due to the lack of cryptanalytic techniques exploiting the algebraic structure.

NTRU. Let \( (f, g) \in \mathbb{Z}^{2n} \) be the NTRU secret as introduced in Chapter 2. We treat \( f \) as the LWE secret \( s \), and \( g \) as the LWE error \( e \) (we note that this can also be considered vice-versa).
### 4.4 Parameter Sets

<table>
<thead>
<tr>
<th>Model</th>
<th>Schemes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRYSTALS</td>
<td>[LDK+17] &lt;sup&gt;1&lt;/sup&gt; SAB+17</td>
</tr>
<tr>
<td>SABER</td>
<td>[DKRV17]</td>
</tr>
<tr>
<td>Falcon</td>
<td>[PFH+17]</td>
</tr>
<tr>
<td>ThreeBears</td>
<td>[Ham17]</td>
</tr>
<tr>
<td>HILA5</td>
<td>[Saa17]</td>
</tr>
<tr>
<td>0.292β</td>
<td>Titanium</td>
</tr>
<tr>
<td>0.265β</td>
<td>KINDI [El 17]</td>
</tr>
<tr>
<td></td>
<td>NTRU HRSS [SHRS17]</td>
</tr>
<tr>
<td></td>
<td>LAC [LLJ+17]</td>
</tr>
<tr>
<td></td>
<td>NTRUEncrypt [ZCHW17a]</td>
</tr>
<tr>
<td></td>
<td>New Hope [PAA+17]</td>
</tr>
<tr>
<td></td>
<td>pqNTRUSign [ZCHW17b]</td>
</tr>
<tr>
<td></td>
<td>Round2 [GZB+17]</td>
</tr>
<tr>
<td>0.292β + 16.4</td>
<td>LIMA [SAL+17]</td>
</tr>
<tr>
<td>0.265β + 16.4</td>
<td></td>
</tr>
<tr>
<td>0.368β</td>
<td></td>
</tr>
<tr>
<td>0.2975β</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NTRU HRSS [SHRS17]</td>
</tr>
<tr>
<td></td>
<td>Frodo [NAB+17]</td>
</tr>
<tr>
<td>0.292β + log(β)</td>
<td>KCL [ZJGS17]</td>
</tr>
<tr>
<td>0.265β + log(β)</td>
<td>Lizard [CPL+17]</td>
</tr>
<tr>
<td></td>
<td>Round2 [GZB+17]</td>
</tr>
<tr>
<td>0.292β + 16.4 + log(8d)</td>
<td>Ding Key Exchange [DTGW17]</td>
</tr>
<tr>
<td>0.265β + 16.4 + log(8d)</td>
<td>EMBLEM [SPL+17]</td>
</tr>
<tr>
<td></td>
<td>qTESLA [BAA+17]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>0.187β log β − 1.019β + 16.1</td>
<td>pqNTRUSign [ZCHW17b]</td>
</tr>
<tr>
<td></td>
<td>NTRUEncrypt [ZCHW17a]</td>
</tr>
<tr>
<td></td>
<td>NTRU HRSS [SHRS17]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NTRU HRSS [SHRS17]</td>
</tr>
<tr>
<td>0.125β log β − 0.755β + 2.25</td>
<td>LOTUS [PHAM17]</td>
</tr>
</tbody>
</table>

Table 4.3: All cost models proposed as part of a submission to the first round of the NIST standardisation procedure. The name of a model is the log (to the base 2) of its cost.
4.4 Parameter Sets

The LWE degree $n$ is exactly the degree of the NTRU polynomial $\phi$, the standard deviation of the LWE error distribution is set to $\|g\|_2/\sqrt{n}$, and the LWE modulus $q$ is exactly the NTRU modulus. We account for the presence of rotations by amplifying the success probability $p$ of guessing entries of the secret correctly to $1 - (1 - p)^k$, where $k$ is the number of rotations.

In Table 4.4 we present the parameters considered in the NTRU-based submissions. In Table 4.5 these parameters have been represented as LWE samples using the techniques discussed above. In Table 4.6 we outline the parameters considered in the LWE and LWR-based schemes.

<table>
<thead>
<tr>
<th>Name</th>
<th>$n$</th>
<th>$q$</th>
<th>$|f|$</th>
<th>$|g|$</th>
<th>NIST</th>
<th>Assumption</th>
<th>$\phi$</th>
<th>Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTRUEncrypt</td>
<td>443</td>
<td>2048</td>
<td>16.94</td>
<td>16.94</td>
<td>1</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>743</td>
<td>2048</td>
<td>22.25</td>
<td>22.25</td>
<td>1, 2, 3, 4, 5</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>107350017</td>
<td>23168.00</td>
<td>23168.00</td>
<td>4, 5</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>Falcon</td>
<td>512</td>
<td>12289</td>
<td>91.71</td>
<td>91.71</td>
<td>1</td>
<td>NTRU</td>
<td>$x^n + 1$</td>
<td>SIG</td>
</tr>
<tr>
<td></td>
<td>768</td>
<td>18433</td>
<td>112.32</td>
<td>112.32</td>
<td>2, 3</td>
<td>NTRU</td>
<td>$x^n - x^{n/2} + 1$</td>
<td>SIG</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>12289</td>
<td>91.71</td>
<td>91.71</td>
<td>4, 5</td>
<td>NTRU</td>
<td>$x^n + 1$</td>
<td>SIG</td>
</tr>
<tr>
<td>NTRU HRSS</td>
<td>700</td>
<td>8192</td>
<td>20.92</td>
<td>20.92</td>
<td>1</td>
<td>NTRU</td>
<td>$\sum_{i=0}^{n-1} x^i$</td>
<td>KEM</td>
</tr>
<tr>
<td>SNTRU Prime</td>
<td>761</td>
<td>4591</td>
<td>16.91</td>
<td>22.52</td>
<td>5</td>
<td>NTRU</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>pqNTRUSign</td>
<td>1024</td>
<td>65537</td>
<td>22.38</td>
<td>22.38</td>
<td>1, 2, 3, 4, 5</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
<td>SIG</td>
</tr>
</tbody>
</table>

Table 4.4: Parameter sets for NTRU-based schemes with secret dimension $n$, modulo $q$, small polynomials $f$ and $g$, and ring $\mathbb{Z}_q[x]/(\phi)$. The NIST column indicates the NIST security category aimed at.
### 4.4 Parameter Sets

<table>
<thead>
<tr>
<th>Name</th>
<th>n</th>
<th>q</th>
<th>$\sigma$</th>
<th>Secret dist.</th>
<th>NIST</th>
<th>Assumption</th>
<th>$\phi$</th>
<th>Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTRU Encrypt</td>
<td>443</td>
<td>2048</td>
<td>0.80</td>
<td>$((-1, 1), 287)$</td>
<td>1</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>743</td>
<td>2048</td>
<td>0.82</td>
<td>$((-1, 1), 495)$</td>
<td>1, 2, 3, 4, 5</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>1073750017</td>
<td>724.00</td>
<td>normal</td>
<td>4, 5</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>Falcon</td>
<td>512</td>
<td>12289</td>
<td>4.05</td>
<td>normal</td>
<td>1</td>
<td>NTRU</td>
<td>$x^n + 1$</td>
<td>SIG</td>
</tr>
<tr>
<td></td>
<td>768</td>
<td>18433</td>
<td>4.05</td>
<td>normal</td>
<td>2, 3</td>
<td>NTRU</td>
<td>$x^n - x^{n/2} + 1$</td>
<td>SIG</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>12289</td>
<td>2.87</td>
<td>normal</td>
<td>4, 5</td>
<td>NTRU</td>
<td>$x^n + 1$</td>
<td>SIG</td>
</tr>
<tr>
<td>NTRU HRSS</td>
<td>700</td>
<td>8192</td>
<td>0.79</td>
<td>$((-1, 1), 437)$</td>
<td>1</td>
<td>NTRU</td>
<td>$\sum_{i=0}^{n-1} x^i$</td>
<td>KEM</td>
</tr>
<tr>
<td>SNTRU Prime</td>
<td>761</td>
<td>4591</td>
<td>0.82</td>
<td>$((-1, 1), 286)$</td>
<td>5</td>
<td>NTRU</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>pqNTRUSign</td>
<td>1024</td>
<td>65537</td>
<td>0.70</td>
<td>$((-1, 1), 501)$</td>
<td>1, 2, 3, 4, 5</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
<td>SIG</td>
</tr>
</tbody>
</table>

Table 4.5: LWE parameter sets for NTRU-based schemes, with dimension $n$, modulo $q$, standard deviation of the error $\sigma$, and ring $\mathbb{Z}_q[x]/(\phi)$. The NIST column indicates the NIST security category aimed at.

<table>
<thead>
<tr>
<th>Name</th>
<th>$n$</th>
<th>$k$</th>
<th>$q$</th>
<th>$\sigma$</th>
<th>Secret dist.</th>
<th>NIST</th>
<th>Assumption</th>
<th>$\phi$</th>
<th>Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>KCL-RLWE</td>
<td>1024</td>
<td>—</td>
<td>12289</td>
<td>2.83</td>
<td>normal</td>
<td>5</td>
<td>Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>KCL-MLWE</td>
<td>768</td>
<td>3</td>
<td>7681</td>
<td>1.00</td>
<td>normal</td>
<td>4</td>
<td>Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>768</td>
<td>3</td>
<td>7681</td>
<td>2.24</td>
<td>normal</td>
<td>4</td>
<td>Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>BabyBear</td>
<td>624</td>
<td>2</td>
<td>1024</td>
<td>1.00</td>
<td>normal</td>
<td>2</td>
<td>ILWE</td>
<td>$q^{n/k} - q^{n/(2k)} - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>624</td>
<td>2</td>
<td>1024</td>
<td>0.79</td>
<td>normal</td>
<td>2</td>
<td>ILWE</td>
<td>$q^{n/k} - q^{n/(2k)} - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>MamaBear</td>
<td>936</td>
<td>3</td>
<td>1024</td>
<td>0.94</td>
<td>normal</td>
<td>5</td>
<td>ILWE</td>
<td>$q^{n/k} - q^{n/(2k)} - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>936</td>
<td>3</td>
<td>1024</td>
<td>0.71</td>
<td>normal</td>
<td>4</td>
<td>ILWE</td>
<td>$q^{n/k} - q^{n/(2k)} - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>PapaBear</td>
<td>1248</td>
<td>4</td>
<td>1024</td>
<td>0.87</td>
<td>normal</td>
<td>5</td>
<td>ILWE</td>
<td>$q^{n/k} - q^{n/(2k)} - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>1248</td>
<td>4</td>
<td>1024</td>
<td>0.61</td>
<td>normal</td>
<td>5</td>
<td>ILWE</td>
<td>$q^{n/k} - q^{n/(2k)} - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>CRYSALS-Dilithium</td>
<td>768</td>
<td>3</td>
<td>8380417</td>
<td>3.74 (6, 6)</td>
<td>1</td>
<td>Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>SIG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>4</td>
<td>8380417</td>
<td>3.16 (5, 5)</td>
<td>2</td>
<td>Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>SIG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1280</td>
<td>5</td>
<td>8380417</td>
<td>2.00 (3, 3)</td>
<td>3</td>
<td>Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>SIG</td>
<td></td>
</tr>
<tr>
<td>CRYSALS-Kyber</td>
<td>512</td>
<td>2</td>
<td>7681</td>
<td>1.58</td>
<td>normal</td>
<td>1</td>
<td>Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>768</td>
<td>3</td>
<td>7681</td>
<td>1.41</td>
<td>normal</td>
<td>3</td>
<td>Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>4</td>
<td>7681</td>
<td>1.22</td>
<td>normal</td>
<td>5</td>
<td>Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>Ding Key Exchange</td>
<td>512</td>
<td>—</td>
<td>120883</td>
<td>4.19</td>
<td>normal</td>
<td>1</td>
<td>Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>—</td>
<td>120883</td>
<td>2.60</td>
<td>normal</td>
<td>3, 5</td>
<td>Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>EMBLEM</td>
<td>770</td>
<td>—</td>
<td>1677216</td>
<td>25.00 (1, 1)</td>
<td>1</td>
<td>LWE</td>
<td>—</td>
<td>KEM, PKE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>611</td>
<td>—</td>
<td>1677216</td>
<td>25.00 (2, 2)</td>
<td>1</td>
<td>LWE</td>
<td>—</td>
<td>KEM, PKE</td>
<td></td>
</tr>
<tr>
<td>R EMBLEM</td>
<td>512</td>
<td>—</td>
<td>65536</td>
<td>25.00 (1, 1)</td>
<td>1</td>
<td>Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>—</td>
<td>16384</td>
<td>3.00 (1, 1)</td>
<td>1</td>
<td>Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
<td></td>
</tr>
</tbody>
</table>

118
### 4.4 Parameter Sets

<table>
<thead>
<tr>
<th>Name</th>
<th>$n$</th>
<th>$k$</th>
<th>$q$</th>
<th>$\sigma$</th>
<th>Secret dist.</th>
<th>NIST Assumption</th>
<th>$\phi$</th>
<th>Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frodo</td>
<td>640</td>
<td></td>
<td>32768</td>
<td>2.75</td>
<td>normal</td>
<td>1 LWE</td>
<td>$x^{n/k} + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>976</td>
<td></td>
<td>65536</td>
<td>2.30</td>
<td>normal</td>
<td>3 LWE</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>NewHope</td>
<td>512</td>
<td></td>
<td>12289</td>
<td>2.00</td>
<td>normal</td>
<td>1 Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td></td>
<td>12289</td>
<td>2.00</td>
<td>normal</td>
<td>5 Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>HILA5</td>
<td>1024</td>
<td></td>
<td>12289</td>
<td>2.83</td>
<td>normal</td>
<td>5 Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KE</td>
</tr>
<tr>
<td>KINDI</td>
<td>1024</td>
<td></td>
<td>133121</td>
<td>3.16</td>
<td>normal</td>
<td>3 Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td></td>
<td>184321</td>
<td>3.16</td>
<td>normal</td>
<td>4 Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>LAC</td>
<td>512</td>
<td></td>
<td>251</td>
<td>0.71</td>
<td>normal</td>
<td>1, 2 PLWE</td>
<td>$x^n + 1$</td>
<td>KE, KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td></td>
<td>251</td>
<td>0.50</td>
<td>normal</td>
<td>3, 4 PLWE</td>
<td>$x^n + 1$</td>
<td>KE, KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td></td>
<td>251</td>
<td>0.71</td>
<td>normal</td>
<td>5 PLWE</td>
<td>$x^n + 1$</td>
<td>KE, KEM, PKE</td>
</tr>
<tr>
<td>LIMA-2p</td>
<td>1024</td>
<td></td>
<td>133121</td>
<td>3.16</td>
<td>normal</td>
<td>3 Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td></td>
<td>184321</td>
<td>3.16</td>
<td>normal</td>
<td>4 Ring-LWE</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>LIMA-sp</td>
<td>1018</td>
<td></td>
<td>1252147</td>
<td>3.16</td>
<td>normal</td>
<td>1 Ring-LWE</td>
<td>$\sum_{i \geq 0} x^i$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1396</td>
<td></td>
<td>4818240</td>
<td>3.16</td>
<td>normal</td>
<td>2 Ring-LWE</td>
<td>$\sum_{i \geq 0} x^i$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1822</td>
<td></td>
<td>4492049</td>
<td>3.16</td>
<td>normal</td>
<td>3 Ring-LWE</td>
<td>$\sum_{i \geq 0} x^i$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>2062</td>
<td></td>
<td>1690097</td>
<td>3.16</td>
<td>normal</td>
<td>4 Ring-LWE</td>
<td>$\sum_{i \geq 0} x^i$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>Lizard</td>
<td>1024</td>
<td></td>
<td>2048</td>
<td>1.12</td>
<td>((−1,1),140)</td>
<td>1 LWE, LWR</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td></td>
<td>1024</td>
<td>1.12</td>
<td>((−1,1),128)</td>
<td>1 LWE, LWR</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td></td>
<td>2048</td>
<td>1.12</td>
<td>((−1,1),200)</td>
<td>3 LWE, LWR</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td></td>
<td>2048</td>
<td>1.12</td>
<td>((−1,1),200)</td>
<td>3 LWE, LWR</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td></td>
<td>4096</td>
<td>1.12</td>
<td>((−1,1),200)</td>
<td>5 LWE, LWR</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td></td>
<td>2048</td>
<td>1.12</td>
<td>((−1,1),200)</td>
<td>5 LWE, LWR</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>RLLizard</td>
<td>1024</td>
<td></td>
<td>1024</td>
<td>1.12</td>
<td>((−1,1),128)</td>
<td>1 Ring-LWE, Ring-LWR</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td></td>
<td>2048</td>
<td>1.12</td>
<td>((−1,1),264)</td>
<td>3 Ring-LWE, Ring-LWR</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td></td>
<td>2048</td>
<td>1.12</td>
<td>((−1,1),164)</td>
<td>3 Ring-LWE, Ring-LWR</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td></td>
<td>4096</td>
<td>1.12</td>
<td>((−1,1),256)</td>
<td>5 Ring-LWE, Ring-LWR</td>
<td>$x^n + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>LOTUS</td>
<td>576</td>
<td></td>
<td>8192</td>
<td>3.00</td>
<td>normal</td>
<td>1, 2 LWE</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>704</td>
<td></td>
<td>8192</td>
<td>3.00</td>
<td>normal</td>
<td>3, 4 LWE</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>812</td>
<td></td>
<td>8192</td>
<td>3.00</td>
<td>normal</td>
<td>5 LWE</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>uRound2 KEM</td>
<td>500</td>
<td></td>
<td>16384</td>
<td>2.29</td>
<td>((−1,1),74)</td>
<td>1 LWR</td>
<td>—</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>580</td>
<td></td>
<td>32768</td>
<td>4.61</td>
<td>((−1,1),116)</td>
<td>2 LWR</td>
<td>—</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>630</td>
<td></td>
<td>32768</td>
<td>4.61</td>
<td>((−1,1),126)</td>
<td>3 LWR</td>
<td>—</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>786</td>
<td></td>
<td>32768</td>
<td>4.61</td>
<td>((−1,1),156)</td>
<td>4 LWR</td>
<td>—</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>786</td>
<td></td>
<td>32768</td>
<td>4.61</td>
<td>((−1,1),156)</td>
<td>5 LWR</td>
<td>—</td>
<td>KEM</td>
</tr>
<tr>
<td>uRound2 KEM</td>
<td>418</td>
<td></td>
<td>4096</td>
<td>4.61</td>
<td>((−1,1),66)</td>
<td>1 Ring-LWR</td>
<td>$\sum_{i \geq 0} x^i$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>522</td>
<td></td>
<td>32768</td>
<td>36.95</td>
<td>((−1,1),78)</td>
<td>2 Ring-LWR</td>
<td>$\sum_{i \geq 0} x^i$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>540</td>
<td></td>
<td>16384</td>
<td>18.47</td>
<td>((−1,1),96)</td>
<td>3 Ring-LWR</td>
<td>$\sum_{i \geq 0} x^i$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>700</td>
<td></td>
<td>32768</td>
<td>36.95</td>
<td>((−1,1),112)</td>
<td>4 Ring-LWR</td>
<td>$\sum_{i \geq 0} x^i$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>676</td>
<td></td>
<td>32768</td>
<td>36.95</td>
<td>((−1,1),120)</td>
<td>5 Ring-LWR</td>
<td>$\sum_{i \geq 0} x^i$</td>
<td>KEM</td>
</tr>
</tbody>
</table>
4.4 Parameter Sets

<table>
<thead>
<tr>
<th>Name</th>
<th>n</th>
<th>k</th>
<th>q</th>
<th>σ</th>
<th>Secret dist.</th>
<th>NIST Assumption</th>
<th>φ</th>
<th>Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>nRound2.PKE</td>
<td>500</td>
<td>—</td>
<td>32768</td>
<td>4.61</td>
<td>((−1, 1), 74)</td>
<td>1</td>
<td>LWR</td>
<td>PKE</td>
</tr>
<tr>
<td>585</td>
<td>—</td>
<td>32768</td>
<td>4.61</td>
<td>((−1, 1), 110)</td>
<td>2</td>
<td>LWR</td>
<td>PKE</td>
<td></td>
</tr>
<tr>
<td>643</td>
<td>—</td>
<td>32768</td>
<td>4.61</td>
<td>((−1, 1), 114)</td>
<td>3</td>
<td>LWR</td>
<td>PKE</td>
<td></td>
</tr>
<tr>
<td>835</td>
<td>—</td>
<td>32768</td>
<td>2.29</td>
<td>((−1, 1), 166)</td>
<td>4</td>
<td>LWR</td>
<td>PKE</td>
<td></td>
</tr>
<tr>
<td>835</td>
<td>—</td>
<td>32768</td>
<td>2.29</td>
<td>((−1, 1), 166)</td>
<td>5</td>
<td>LWR</td>
<td>PKE</td>
<td></td>
</tr>
<tr>
<td>nRound2.PKE</td>
<td>420</td>
<td>—</td>
<td>1024</td>
<td>1.12</td>
<td>((−1, 1), 62)</td>
<td>1</td>
<td>Ring-LWR</td>
<td>PKE</td>
</tr>
<tr>
<td>540</td>
<td>—</td>
<td>8192</td>
<td>4.61</td>
<td>((−1, 1), 96)</td>
<td>2</td>
<td>Ring-LWR</td>
<td>PKE</td>
<td></td>
</tr>
<tr>
<td>586</td>
<td>—</td>
<td>8192</td>
<td>4.61</td>
<td>((−1, 1), 104)</td>
<td>3</td>
<td>Ring-LWR</td>
<td>PKE</td>
<td></td>
</tr>
<tr>
<td>708</td>
<td>—</td>
<td>32768</td>
<td>18.47</td>
<td>((−1, 1), 140)</td>
<td>4.5</td>
<td>Ring-LWR</td>
<td>PKE</td>
<td></td>
</tr>
<tr>
<td>nRound2.KEM</td>
<td>400</td>
<td>—</td>
<td>3209</td>
<td>3.61</td>
<td>((−1, 1), 72)</td>
<td>1</td>
<td>Ring-LWR</td>
<td>KEM</td>
</tr>
<tr>
<td>486</td>
<td>—</td>
<td>1949</td>
<td>2.18</td>
<td>((−1, 1), 96)</td>
<td>2</td>
<td>Ring-LWR</td>
<td>KEM</td>
<td></td>
</tr>
<tr>
<td>556</td>
<td>—</td>
<td>3343</td>
<td>3.76</td>
<td>((−1, 1), 88)</td>
<td>3</td>
<td>Ring-LWR</td>
<td>KEM</td>
<td></td>
</tr>
<tr>
<td>658</td>
<td>—</td>
<td>1319</td>
<td>1.46</td>
<td>((−1, 1), 130)</td>
<td>4.5</td>
<td>Ring-LWR</td>
<td>KEM</td>
<td></td>
</tr>
<tr>
<td>LightSaber</td>
<td>512</td>
<td>2</td>
<td>8192</td>
<td>2.29</td>
<td>normal</td>
<td>1</td>
<td>Module-LWR</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>NTRU LPrime</td>
<td>761</td>
<td>—</td>
<td>4591</td>
<td>0.82</td>
<td>((−1, 1), 250)</td>
<td>5</td>
<td>Ring-LWR</td>
<td>KEM</td>
</tr>
<tr>
<td>Saber</td>
<td>768</td>
<td>3</td>
<td>8192</td>
<td>2.29</td>
<td>normal</td>
<td>3</td>
<td>Module-LWR</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>FireSaber</td>
<td>1024</td>
<td>4</td>
<td>8192</td>
<td>2.29</td>
<td>normal</td>
<td>5</td>
<td>Module-LWR</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>qTESLA</td>
<td>1024</td>
<td>—</td>
<td>805881</td>
<td>8.49</td>
<td>normal</td>
<td>1</td>
<td>Ring-LWE</td>
<td>SIG</td>
</tr>
<tr>
<td>2048</td>
<td>—</td>
<td>12681217</td>
<td>8.49</td>
<td>normal</td>
<td>3</td>
<td>Ring-LWE</td>
<td>SIG</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>—</td>
<td>27627521</td>
<td>8.49</td>
<td>normal</td>
<td>5</td>
<td>Ring-LWE</td>
<td>SIG</td>
<td></td>
</tr>
<tr>
<td>Correction Aug.End</td>
<td>1024</td>
<td>—</td>
<td>86017</td>
<td>1.41</td>
<td>normal</td>
<td>1</td>
<td>PLWE</td>
<td>PKE</td>
</tr>
<tr>
<td>1280</td>
<td>—</td>
<td>301057</td>
<td>1.41</td>
<td>normal</td>
<td>1</td>
<td>PLWE</td>
<td>PKE</td>
<td></td>
</tr>
<tr>
<td>1536</td>
<td>—</td>
<td>737283</td>
<td>1.41</td>
<td>normal</td>
<td>3</td>
<td>PLWE</td>
<td>PKE</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>—</td>
<td>1198081</td>
<td>1.41</td>
<td>normal</td>
<td>5</td>
<td>PLWE</td>
<td>PKE</td>
<td></td>
</tr>
<tr>
<td>Correction Aug.End</td>
<td>1024</td>
<td>—</td>
<td>118273</td>
<td>1.41</td>
<td>normal</td>
<td>1</td>
<td>PLWE</td>
<td>KEM</td>
</tr>
<tr>
<td>1280</td>
<td>—</td>
<td>430081</td>
<td>1.41</td>
<td>normal</td>
<td>1</td>
<td>PLWE</td>
<td>KEM</td>
<td></td>
</tr>
<tr>
<td>1536</td>
<td>—</td>
<td>783361</td>
<td>1.41</td>
<td>normal</td>
<td>3</td>
<td>PLWE</td>
<td>KEM</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>—</td>
<td>1198081</td>
<td>1.41</td>
<td>normal</td>
<td>5</td>
<td>PLWE</td>
<td>KEM</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6: Parameter sets for LWE-based schemes with secret dimension $n$, Module-LWE rank $k$ (if any), modulo $q$, standard deviation of the error $\sigma$. If the LWE samples come from a Ring- or Module-LWE instance, the ring is $\mathbb{Z}_q[x]/(\phi)$. The NIST column indicates the NIST security category aimed at. *For Titanium no ring is explicitly chosen but the scheme relies on a family of rings where $f_i \in \{-1,0,1\}$ and $f_0 \in \{-1,1\}$. † For R EMBLEM we list the parameters from the reference implementation since a suitable $\phi$ could not be found for those proposed in [SPL+17, Table 2].
4.5 Small Secret Variants of the uSVP and Dual Attacks

In Tables 4.5 and 4.6 we see that a total of 104 parameter sets were submitted to the first round. Of these parameter sets, nine consider the NTRU assumption, 23 consider the LWE and/or LWR assumptions, 39 consider the Ring-LWE and/or Ring-LWR assumptions, 16 consider the Module-LWE and/or Module-LWR assumptions, and a further 17 consider alternative LWE-based assumptions.

4.5 Small Secret Variants of the uSVP and Dual Attacks

The uSVP and dual attacks can be optimised for variants of Small-secret LWE. In a similar manner to the decoding attack in Chapter 3, we can combine the uSVP and dual attacks with combinatorics to retrieve a complexity improvement. In this section, we introduce the small-secret variants of the uSVP and dual attacks.

4.5.1 uSVP

The uSVP attack on Small-secret LWE considers the Bai and Galbraith embedding [BG14]. As opposed to the uSVP variant outlined in Chapter 2 where the target vector is \((e, 1)\), the Bai and Galbraith embedding constructs the lattice with basis matrix \(B\), where:

\[
B = \begin{pmatrix}
I_n & 0 & 0 \\
-A & qI_m & b \\
0 & 0 & 1
\end{pmatrix}.
\]

The vector \((s, e, 1)\) is embedded in the lattice \(L(B)\), and this can be seen since:

\[
\begin{pmatrix}
I_n & 0 & 0 \\
-A & qI_m & b \\
0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
s \\
*s \\
1
\end{pmatrix} = \begin{pmatrix}
s \\
-s + q*s + b \\
1
\end{pmatrix} = \begin{pmatrix}
s \\
e \\
1
\end{pmatrix} \mod q.
\]

In the case of normal-form LWE, we can view this variant of the uSVP attack as identical to the approach considered in Chapter 2. However, we note that in the small secret variant of this attack, we can employ dimension reduction techniques [MS01, APS15, Alb17] to reduce the overall complexity. That is, we guess \(\tau\) components of the LWE secret, before solving

\[\text{These assumptions are Polynomial-LWE [SSTX09], and Integer Module-LWE [Ham17].}\]
4.5 Small Secret Variants of the uSVP and Dual Attacks

uSVP in the \((n + m - \tau)\) dimension lattice with basis:

\[
B = \begin{pmatrix}
I_{n-\tau} & 0 & 0 \\
-A(\tau) & qI_m & b \\
0 & 0 & 1
\end{pmatrix},
\]

where we recall that \(A(\tau)\) corresponds to the matrix \(A\) with the first \(\tau\) columns removed. If we correctly guess zeros then the vector \((s_{\tau+1}, s_{\tau+2}, \ldots, s_n, e, 1)\) is embedded in the lattice \(L(B)\) reduced by BKZ, and this allows for complete recovery of the LWE secret. If we incorrectly guess the zero components, then we have to restart this process with a fresh guess of \(\tau\) zero components. The running time of the dimension reduced problem, constrained by the \([ADPS16]\) success condition outlined in Chapter \(\ref{chapter:dimension_reduction}\) is:

\[
T_{uSVP} = \min_{\beta, \tau, m} \left\{ \frac{1}{p_\tau} \cdot T_{BKZ}(\beta, n + m - \tau) \right\},
\]

where \(p_\tau\) is the probability of correctly guessing \(\tau\) zero components of the LWE secret, and \(T_{BKZ}(\beta, n + m - \tau)\) is the cost of running BKZ on a dimension \((n + m - \tau)\) lattice with blocksize \(\beta\). We note that setting \(\tau = 0\) corresponds to not performing any guessing.

4.5.2 Dual

Recall the dual attack from Section \(\ref{section:dual_attack}\) which finds short vectors \(v\) in the lattice:

\[
\Lambda^* = \{ x \in \mathbb{Z}_q^m \mid xA \equiv 0 \mod q \},
\]

before computing inner products of the form:

\[
\langle v, b \rangle = \langle v, As + e \rangle = \langle vA, s \rangle + \langle v, e \rangle = \langle v, e \rangle \mod q.
\]

Inner products of this form allow us to distinguish LWE from random. For small and/or sparse secrets, Albrecht suggests several improvements to this attack \([Alb17]\), which we summarise in the following four subsections.

4.5.2.1 Combinatorics

For sparse secrets, Albrecht suggests splitting the matrix \(A = [A_0 \mid A_1]\) where \(A_0 \in \mathbb{Z}_q^{m \times (n-\tau)}\) and \(A_1 \in \mathbb{Z}_q^{m \times \tau}\). Splitting the LWE secret \(s = (s_0, s_1)\) in the same manner, we can see that:

\[
As = A_0s_0 + A_1s_1.
\]
4.5 Small Secret Variants of the uSVP and Dual Attacks

In this variant of the attack, we begin by finding short vectors in the lattice:

$$\Lambda' = \{v \in \mathbb{Z}^n \mid vA_0 \equiv 0 \mod q\},$$

which corresponds to guessing $\tau$ components of the LWE secret as zero, i.e. $(s_1 = 0)$ and hoping that $As = A_0s_0$, or equivalently that $A_1s_1 \equiv 0 \mod q$ (in a similar manner to the hybrid-decoding attack outlined in Chapter 3 and the small-secret uSVP attack outlined in Section 4.5.1). Note that setting $\tau = 0$ corresponds to not performing any guessing, and therefore reduces to the original dual attack. Suppose, then, that we proceed with the following steps:

1. we collect $\nu$ short vectors $\{v_i\}_{i=1}^\nu$ each contained within the lattice $\Lambda'$, typically by performing lattice reduction $\nu$ times, and
2. compute a collection of inner products $\{\tilde{e}_i = \langle v_i, b \rangle\}_{i=1}^\nu$.

We have:

$$\tilde{e}_i = \langle v_i, b \rangle = \langle v_i, As + e \rangle = \langle v_i, As \rangle + \langle v_i, e \rangle$$

$$= \langle v_iA_0, s_0 \rangle + \langle v_iA_1, s_1 \rangle + \langle v_i, e \rangle$$

$$= \langle v_iA_1, s_1 \rangle + \langle v_i, e \rangle,$$

since $v_iA_0 \equiv 0 \mod q$. If our guess of $\tau$ zeros (i.e. $s_1 = 0$) was correct, then these computed terms are of the form $\langle v_i, e \rangle$ since $\langle v_iA_1, s_1 \rangle = 0$. In this case, the $\tilde{e}_i$ terms follow a Discrete Gaussian distribution and we have solved Decision-LWE.

Otherwise, we know that our guess of $\tau$ zeros is incorrect, and we can therefore make additional guesses in the $\tau$-dimensional guessing space (which, for example, if $s \sim B^{-}$ would be $\{-1, 0, 1\}^\tau$). For each guess, we compare the corresponding distributions provided by the inner products against the uniform distribution modulo $q$. Specifically, we know that the additional term is of the form $\langle v_iA_1, s_1 \rangle$ and we can therefore search over possible secrets $s_1$ and attempt to remove this additional term from the inner product/s. In particular, for candidate guesses $s'$ we construct terms of the form:

$$e_i'' = e_i' - \langle v_iA_1s' \rangle.$$

For the correct candidate secret, i.e. $s' = s_1$, the terms $e_i''$ are of the form $\langle v_i, e \rangle$ as is required.
4.5 Small Secret Variants of the uSVP and Dual Attacks

4.5.2.2 Amortising the Cost of Lattice Reduction

Albrecht also considers re-randomisation techniques to amortise the cost of lattice reduction. Recall that, for a successful attack, we may require $\nu$ short vectors from the lattice $L(B)$. To retrieve these vectors, one could imagine re-randomising the input basis before running the BKZ algorithm on a new, randomised, lattice basis:

$$\tilde{B}_i \leftarrow \text{BKZ}_\beta(U_i B),$$

$\nu$ times, and using the shortest vectors in the lattice bases $\{\tilde{B}_i\}_{i=1}^\nu$. Albrecht notes that this is unnecessary and suggests performing an initial, heavy, lattice reduction step:

$$\tilde{B} \leftarrow \text{BKZ}_\beta(B),$$

and applying randomisation after this lattice reduction has taken place. That is, we can use the shortest vectors in the lattice bases:

$$B'_i \leftarrow \text{BKZ}_{\beta'}(U_i \tilde{B}),$$

for some blocksize $\beta' \ll \beta$. The idea is to take the shortest vector from each of these re-randomised lattice bases, and use the resulting set of short vectors to distinguish against the uniform distribution modulo $q$. The length of the short vectors generated via this re-randomisation process are longer than those generated using using a fresh call to BKZ-$\beta$. Albrecht suggests that these vectors have their norms increased by a factor of two compared to the initial vector retrieved directly from BKZ-$\beta$, and provides experimental evidence to back up this heuristic [Alb17]. However, these longer vectors can be generated at a significantly lower cost via this amortisation technique. The hope is that this trade-off in vector length vs computation time provides a speed-up in the overall attack cost.

4.5.2.3 Scaled Normal Form

The final technique stems from the observation that it is sufficient to find a short vector $(v, w)$ in the lattice given by:

$$\Lambda'' = \left\{(v, w) \in \mathbb{Z}^m \times \left(\frac{1}{c} \cdot \mathbb{Z}\right)^n \mid vA \equiv cw \mod q \right\}.$$  

If we find such a short vector, then we note that:

$$\langle v, b \rangle = \langle v, As + e \rangle = \langle vA, s \rangle + \langle v, e \rangle = \langle cw, s \rangle + \langle v, e \rangle,$$
and, assuming that $\mathbf{w}, \mathbf{s}$ are both sufficiently short, we can still distinguish LWE from random modulo $q$, albeit with a different advantage, since the additional error term $\langle c\mathbf{w}, \mathbf{s} \rangle$ is small. A scaling factor $c$ is computed to balance the contributions of the two terms (i.e. $\langle c\mathbf{w}, \mathbf{s} \rangle$ and $\langle \mathbf{v}, \mathbf{e} \rangle$), which is computed as:

$$c = \frac{\alpha q}{\sqrt{2\pi h}} \sqrt{m - n},$$

in [Alb17].

### 4.5.2.4 Combining Techniques

The small-secret variant of the dual attack combines these techniques (where appropriate) to leverage both the smallness as well as any sparsity of the LWE secret vector $\mathbf{s}$. We note that these techniques are implemented in the LWE Estimator for both binary and ternary secrets.

### 4.5.3 Multiple Hardness Assumptions

The Lizard (RLizard) scheme is based on two hardness assumptions, LWE (Ring-LWE) and LWR (Ring-LWR). Secret key recovery corresponds to the underlying LWE problem, and ephemeral key recovery corresponds to the underlying LWR problem. There exists parameter sets for which secret key recovery is harder than ephemeral key recovery (i.e. the underlying LWE problem is harder than the underlying LWR problem), and there also exists parameter sets for which the converse is true, i.e. ephemeral key recovery is harder than secret key recovery. To deal with this issue, in each cost model, for every attack, we consider both the cost of ephemeral key recovery and the cost of secret key recovery and always choose the lower of the two estimates.

### 4.5.4 Number of Samples

LWE as defined in Definition 2.32 provides the adversary with an arbitrary number of samples denoted by $m$. When using the LWE Estimator we can set the value $m = \infty$ and allow the adversary to have access to an infinite number of samples. In practice, however, this is not the case. In particular, in the Ring-LWE KEM setting – which is the most common for the
schemes considered in this chapter – the public key is one Ring-LWE sample:

\[(a, b) = (a, as + e),\]

for some short \(s, e\), and encapsulations consist of two Ring-LWE samples

\[va + e',\]

and:

\[vb + e'' + \tilde{m},\]

where \(\tilde{m}\) is some encoding of a random string and \(v, e', e''\) are short. Thus, depending on the target, the adversary is given either \(n\) or \(2n\) plain LWE samples.

In a typical setting, though, the adversary does not get to enjoy the full power of having two Ring-LWE samples at its disposal, because, firstly, the random string \(\tilde{m}\) increases the noise in \(vb + e'' + \tilde{m}\) and, secondly, because many schemes drop lower order bits from \(vb + e'' + \tilde{m}\) to save bandwidth. Due to the way decryption works this bit dropping can be quite aggressive, and thus the noise in the second sample can be quite large. In the case of Module-LWE, a ciphertext in transit produces a smaller number of LWE samples, but \(n\) samples can still be recovered from the public key. In this work, we consider the \(m = n\) and \(m = 2n\) scenarios for all schemes. We note that, for many schemes, \(n\) samples are sufficient to run the most efficient variant of either attack.

### 4.6 First Round Security Estimates

We provide estimates for all first round parameter sets considered as part of a submission, under all BKZ cost models considered as part of a submission, for both the uSVP and dual attacks (where appropriate). This corresponds to over 150 parameter sets, under 14 cost models, yielding more than 2000 individual security estimates. Our results can be found at [https://estimate-all-the-lwe-ntru-schemes.github.io/docs/](https://estimate-all-the-lwe-ntru-schemes.github.io/docs/) In this chapter, we present a small subset of the results. Specifically, in Table 4.7 we present the results for a single parameter set considered in each of the schemes EMBLEM, CRYSTALS-Kyber, NTRU Prime, and uRound2.KEM, under the two cost models 0.292\(\beta\) and 0.187\(\beta\) log(\(\beta\)) – 1.019\(\beta\) + 16.1. In each case we highlight the associated value of \(n\) for clarity.
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Assumption</th>
<th>Claimed Security</th>
<th>NIST Category</th>
<th>Attack</th>
<th>$0.187\beta \log(\beta) - 1.019\beta + 16.1$</th>
<th>$0.292\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMBLEM ($n = 611$)</td>
<td>LWE</td>
<td>128</td>
<td>1</td>
<td>dual</td>
<td>142</td>
<td>76</td>
</tr>
<tr>
<td>EMBLEM ($n = 611$)</td>
<td>LWE</td>
<td>128</td>
<td>1</td>
<td>primal-usvp</td>
<td>152</td>
<td>91</td>
</tr>
<tr>
<td>Kyber ($n = 512$)</td>
<td>Ring-LWE</td>
<td>128</td>
<td>1</td>
<td>dual</td>
<td>289</td>
<td>141</td>
</tr>
<tr>
<td>Kyber ($n = 512$)</td>
<td>Ring-LWE</td>
<td>128</td>
<td>1</td>
<td>primal-usvp</td>
<td>244</td>
<td>113</td>
</tr>
<tr>
<td>NTRULPrime ($n = 761$)</td>
<td>Ring-LWR</td>
<td>128</td>
<td>5</td>
<td>dual</td>
<td>365</td>
<td>179</td>
</tr>
<tr>
<td>NTRULPrime ($n = 761$)</td>
<td>Ring-LWR</td>
<td>128</td>
<td>5</td>
<td>primal-usvp</td>
<td>365</td>
<td>156</td>
</tr>
<tr>
<td>uRound2.KEM ($n = 500$)</td>
<td>LWR</td>
<td>128</td>
<td>1</td>
<td>dual</td>
<td>127</td>
<td>94</td>
</tr>
<tr>
<td>uRound2.KEM ($n = 500$)</td>
<td>LWR</td>
<td>128</td>
<td>1</td>
<td>primal-usvp</td>
<td>126</td>
<td>84</td>
</tr>
</tbody>
</table>

Table 4.7: Estimates for the first round submissions EMBLEM, NewHope, NTRU Prime, and uRound2.KEM under the $0.292\beta$ and $0.187\beta \log(\beta) - 1.019\beta + 16.1$ cost models. Estimates are taken directly from [https://estimate-all-the-lwe-ntru-schemes.github.io/docs/](https://estimate-all-the-lwe-ntru-schemes.github.io/docs/).
4.6 First Round Security Estimates

4.6.1 Observation: Cost Swaps

One of the interesting observations from our data is that cost models for lattice reduction do not necessarily preserve the ordering of the schemes under consideration. More explicitly: if scheme A is considered harder to break than scheme B under cost model 1, the same does not necessarily hold for cost model 2. That is, under cost model 2, scheme B could be considered harder to break than scheme A.

To find an example of this, we consider EMBLEM and uRound2.KEM, as highlighted in [Ber18]. As can been seen in Table 4.8, in the sieving-based cost model $0.292\beta$, the associated security estimate of EMBLEM is 76 bits and uRound2.KEM is 84 bits, making EMBLEM easier to break. However, under the enumeration-based cost model $0.187\beta\log(\beta) - 1.019\beta + 16.1$ the associated security estimate of EMBLEM is 142 bits and uRound2.KEM is 126 bits, making uRound2.KEM easier to break.

<table>
<thead>
<tr>
<th></th>
<th>EMBLEM</th>
<th>uRound2.KEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0.292\beta$</td>
<td>76</td>
<td>84</td>
</tr>
<tr>
<td>$0.187\beta\log(\beta) - 1.019\beta + 16.1$</td>
<td>142</td>
<td>126</td>
</tr>
</tbody>
</table>

Table 4.8: Security estimates for the first round variants of EMBLEM and uRound2.KEM. Best in class are highlighted in bold, and we can see this is an example of a cost swap: in the $0.292\beta$ model uRound2.KEM has a higher security estimate (84-bits vs 76-bits), whereas in the $0.187\beta\log(\beta) - 1.019\beta + 16.1$ model EMBLEM has a higher security estimate (142-bits vs 126-bits).

Similar swaps can be observed for several other pairs of schemes and cost models. In most cases the estimated securities of the two schemes are very close to each other (differing by, say, 1 or 2 bits) and thus a swap of ordering does not fundamentally alter our understanding of their relative security, as these estimates are typically derived by heuristically searching through the space of possible parameters and computing with limited precision. In some cases, though, such as the one highlighted in [Ber18], the differences in security estimates can be significant. As part of our work, we identified two cases in which this can happen: the first involves sparse secrets, and the second concerns the dual attack.
4.6 First Round Security Estimates

4.6.1.1 Sparse secrets

The first class of cases involves instances with sparse secrets. The LWE Estimator applies
guessing strategies when costing the dual attack [Alb17] and the primal attack. The basic
idea, as discussed throughout this chapter, is that many of the entries of the secret vector are
zero, and hence can be ignored. We guess \( \tau \) entries to be zero, and drop the corresponding
columns from the attack lattice. In dropping \( \tau \) columns from a \( n \)-dimensional LWE instance,
we obtain a \((n - \tau)\)-dimensional LWE instance with a more dense secret distribution, where
the density depends on the choice of \( \tau \) and the original value of \( h \). On the one hand, there is
a probability of failure when guessing which columns to drop. On the other hand there may
exist a \( \tau \) for which the \((n - \tau)\)-dimensional LWE instance is easier to solve, and in particular
requires a smaller BKZ blocksize \( \beta \).

The trade-off between running BKZ on smaller lattices and having to run it multiple times
can correspond to an overall lower expected attack cost. This probability of failure when
guessing secret entries does not depend on the cost model, but rather on the weight and
dimension of the secret, making this kind of attack more effective for very sparse secrets.

In the case of comparing an enumeration cost model versus a sieving cost model, we have
that the cost of enumeration is fitted as \( 2^{\Theta(\beta \log \beta)} \) or \( 2^{\Theta(\beta^2)} \) whereas the cost of sieving is
\( 2^{\Theta(\beta)} \). The steeper curve for enumeration means that as we increase \( \tau \), and hence decrease
\( \beta \), savings are potentially larger, justifying a larger number \( \tau \) of entries guessed. Concretely,
the computed optimal guessing dimension \( \tau \) can be much larger than in the sieving regime.
This phenomenon can also be observed when comparing two different sieving models or two
different enumeration models.

In Figure 4.3 we illustrate this for the EMBLEM and uRound2.KEM example. EMBLEM
does not have a sparse secret, while uRound2.KEM does. For EMBLEM the best guessing
dimension, giving the lowest overall cost, is \( \tau = 0 \) in both cost models. For uRound2.KEM,
we see that the optimal guessing dimension varies depending on the cost model. In the 0.292\( \beta \)
cost model, the lowest overall expected cost is achieved for \( \tau = 1 \) while in the 0.187\( \beta \log \beta - 1.019\beta + 16.1 \) model the optimal choice is \( \tau = 197 \).
4.6 First Round Security Estimates

![Graph showing estimates of cost vs. \(\tau\)]

Figure 4.3: Estimates of the cost of the primal attack when guessing \(\tau\) secret entries for the schemes EMBLEM \((n = 611)\) and uRound2.KEM \((n = 500)\).

### 4.6.1.2 The Dual attack

The second class of cases can be observed for the dual attack. Recall that the dual attack runs lattice reduction to find a small vector \(v\) in the scaled dual lattice of \(A\), and then considers \(\langle v, b \rangle\) which is short when \(A, b\) is an LWE sample. In more detail, the advantage of distinguishing \(\langle v, b \rangle\), as discussed in Chapter 2, is:

\[
\varepsilon = \exp(-\delta^2 d c_0),
\]

for some constant \(c_0\) depending on the instance and with \(d\) being the dimension of the lattice under consideration \([LP11]\). To amplify this advantage to a constant advantage, we have to repeat the experiment roughly \(1/\varepsilon^2\) times. Therefore, the overall cost of the attack is approximately:

\[
T_{\text{BKZ}}(\beta, d)/\exp(-\delta^2 d c_0)^2.
\]
In the sieving regime we have $T_{BKZ}(\beta, d) \approx 2^{c_1 \beta}$, and in the enumeration regime we have:

$$T_{BKZ}(\beta, d) \approx \beta^{c_2 \beta}$$

(from enumeration costing $2^{\Theta(\beta \log \beta)}$. For large $\beta$ we have $\delta \approx \beta^{1/2}$ \cite{Che13}, and thus we have overall log costs of roughly:

$$c_1 \beta + 2 \log(e) \beta^{d/\beta} c_0,$$

and

$$c_2 \beta \log(\beta) + 2 \log(e) \beta^{d/\beta} c_0.$$

We wish to minimise both expressions (under the constraint that $\beta \geq 2$) and the optimal trade-off depends on $c_0$, $c_1$ and $c_2$. In particular, the optimal $\beta$ in the sieving regime is not necessarily the optimal $\beta$ in the enumeration regime.

### 4.7 Second Round Submissions

In January 2019, NIST announced the second round submissions. Of the 69 submissions from the first round, 26 submissions made it through to the second round \cite{Moo19}. The 23 lattice-based submission from the first round became 12. At this point, designers were allowed to make more significant changes to their submissions including e.g. the merging of multiple submissions, design changes, and new parameter sets. The second round submissions are outlined in Table 4.9.

### 4.8 The Third Round

In July 2020, NIST announced the third round submissions. Of the 26 submissions from the second round, NIST announced seven third round finalists, as well as eight candidate algorithms. Of these eight finalists, five are lattice-based submissions, and of the seven candidate algorithms, two are lattice-based submissions \cite{Moo20}. The third round submissions are outlined in Table 4.10.

### 4.8.1 Cost Models

Next, we consider which of the cost models from the first round, given in Table 4.3, are considered in third round submissions, and we present these cost models in Table 4.11.
### 4.8 The Third Round

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Schemes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crystals-Dilithium</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDK$^{+19}$</td>
<td>Finalist</td>
</tr>
<tr>
<td>Crystals-Kyber</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAB$^{+19}$</td>
<td>Finalist</td>
</tr>
<tr>
<td>Frodo</td>
<td>NAB$^{+19}$</td>
<td></td>
</tr>
<tr>
<td>LAC</td>
<td>LLJ$^{+19}$</td>
<td></td>
</tr>
<tr>
<td>LWE Variants</td>
<td>NewHope</td>
<td>PAA$^{+19}$</td>
</tr>
<tr>
<td></td>
<td>Round5</td>
<td>GZB$^{+19}$</td>
</tr>
<tr>
<td></td>
<td>Merge of HILA5 and Round2</td>
<td></td>
</tr>
<tr>
<td>Saber</td>
<td>DKRV19</td>
<td></td>
</tr>
<tr>
<td>ThreeBears</td>
<td>Ham19</td>
<td></td>
</tr>
<tr>
<td>qTesla</td>
<td>BAA$^{+19}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Falcon</td>
<td>PFH$^{+19}$</td>
</tr>
<tr>
<td>NTRU Variants</td>
<td>NTRU</td>
<td>ZCH$^{+19}$</td>
</tr>
<tr>
<td></td>
<td>Merge of NTRUEncrypt and NTRU HRSS</td>
<td></td>
</tr>
<tr>
<td>NTRU Prime</td>
<td>BCLv19</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.9: Second round lattice-based submissions to the NIST standardisation process.

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Schemes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crystals-Dilithium</td>
<td>LDK$^{+20}$</td>
<td>Finalist</td>
</tr>
<tr>
<td>LWE Variants</td>
<td>Crystals-Kyber</td>
<td>SAB$^{+20}$</td>
</tr>
<tr>
<td></td>
<td>Frodo</td>
<td>NAB$^{+20}$</td>
</tr>
<tr>
<td></td>
<td>Candidate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Saber</td>
<td>DKR$^{+20}$</td>
</tr>
<tr>
<td></td>
<td>Finalist</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Falcon</td>
<td>PFH$^{+20}$</td>
</tr>
<tr>
<td>NTRU Variants</td>
<td>NTRU</td>
<td>ZCH$^{+20}$</td>
</tr>
<tr>
<td></td>
<td>Finalist</td>
<td></td>
</tr>
<tr>
<td>NTRU Prime</td>
<td>BCLv20</td>
<td>Candidate</td>
</tr>
</tbody>
</table>

Table 4.10: Third round lattice-based submissions to the NIST standardisation process.
### 4.8 The Third Round

<table>
<thead>
<tr>
<th>Model</th>
<th>Schemes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SABER$^{DKR^+20}$</td>
<td></td>
</tr>
<tr>
<td>$0.292 \beta$</td>
<td>FALCON$^{PFH^+20}$</td>
</tr>
<tr>
<td>$0.265 \beta$</td>
<td>NTRU Prime$^{BCLv20}$</td>
</tr>
<tr>
<td>Crystals-Kyber</td>
<td>SAB$^+20$</td>
</tr>
<tr>
<td>Crystals-Dilithium</td>
<td>LDK$^+20$</td>
</tr>
<tr>
<td>$0.396 \beta$†</td>
<td>NTRU Prime$^{BCLv20}$</td>
</tr>
<tr>
<td>$0.3496 \beta$†</td>
<td>NTRU$^{ZCH^+20}$</td>
</tr>
<tr>
<td>$0.4150 \beta$†</td>
<td>NTRU$^{ZCH^+20}$</td>
</tr>
<tr>
<td>$0.292 \beta + \log(\beta)$</td>
<td></td>
</tr>
<tr>
<td>$0.265 \beta + \log(\beta)$</td>
<td>Frodo$^{NAB^+20}$</td>
</tr>
<tr>
<td>$0.2075 \beta + \log(\beta)$†</td>
<td></td>
</tr>
<tr>
<td>$0.187 \beta \log \beta - 1.019 \beta + 16.1$</td>
<td>NTRU Prime$^{BCLv20}$</td>
</tr>
<tr>
<td>$\frac{1}{2}(0.187 \beta \log \beta - 1.019 \beta + 16.1)$</td>
<td>NTRU Prime$^{BCLv20}$</td>
</tr>
</tbody>
</table>

Table 4.11: All cost models proposed as part of a submission to the third round of the NIST standardisation procedure. The name of a model is the log (to the base 2) of its cost. Cost models which were not used as part of our analysis for the first round submissions are marked with a †. The 0.396β model considers the 0.292β model mapped to the AT metric$^{Ber20}$. The 0.3496β and 0.4150β models are used in the NTRU submission under the assumption of a “local” model of computation. The 0.2075β + log(β) model was mentioned in the round one Frodo submission, but wasn’t used to produce estimates.
4.8 The Third Round

4.8.2 Parameter Sets and Estimates

Next, we consider the seven lattice-based schemes listed in Table 4.10. We present updated parameter selections in Tables 4.12, 4.13, and 4.14. Finally, we present Core-SVP estimates, both in the classical regime ($0.292\beta$) and the quantum regime ($0.265\beta$) for the Round 3 parameter sets in Tables 4.15 and 4.16.

<table>
<thead>
<tr>
<th>Name</th>
<th>$n$</th>
<th>$q$</th>
<th>$|f|$</th>
<th>$|g|$</th>
<th>NIST Assumption</th>
<th>$\phi$</th>
<th>Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTRU</td>
<td>509</td>
<td>2048</td>
<td>$\sqrt{2/3}\sqrt{509}$</td>
<td>$\sqrt{q/8 - 2}$</td>
<td>1</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
</tr>
<tr>
<td>677</td>
<td>2048</td>
<td>$\sqrt{2/3}\sqrt{677}$</td>
<td>$\sqrt{q/8 - 2}$</td>
<td>3</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>821</td>
<td>4096</td>
<td>$\sqrt{2/3}\sqrt{821}$</td>
<td>$\sqrt{q/8 - 2}$</td>
<td>5</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>701</td>
<td>8192</td>
<td>20.92</td>
<td>20.92</td>
<td>3</td>
<td>NTRU</td>
<td>$x^n - 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>Falcon</td>
<td>512</td>
<td>12289</td>
<td>$1.17\sqrt{q/2}$</td>
<td>$1.17\sqrt{q/2}$</td>
<td>1</td>
<td>NTRU</td>
<td>$x^n + 1$</td>
</tr>
<tr>
<td>1024</td>
<td>12289</td>
<td>$1.17\sqrt{q/2}$</td>
<td>$1.17\sqrt{q/2}$</td>
<td>5</td>
<td>NTRU</td>
<td>$x^n + 1$</td>
<td>SIG</td>
</tr>
<tr>
<td>SNTRU Prime</td>
<td>653</td>
<td>4621</td>
<td>$\sqrt{258}$</td>
<td>$\sqrt{653\sqrt{2/3}}$</td>
<td>1</td>
<td>NTRU</td>
<td>$x^n - x - 1$</td>
</tr>
<tr>
<td>761</td>
<td>4591</td>
<td>$\sqrt{258}$</td>
<td>$\sqrt{761\sqrt{2/3}}$</td>
<td>2</td>
<td>NTRU</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>857</td>
<td>5167</td>
<td>$\sqrt{322}$</td>
<td>$\sqrt{857\sqrt{2/3}}$</td>
<td>2</td>
<td>NTRU</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>953</td>
<td>6343</td>
<td>$\sqrt{396}$</td>
<td>$\sqrt{953\sqrt{2/3}}$</td>
<td>3</td>
<td>NTRU</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>1013</td>
<td>7177</td>
<td>$\sqrt{448}$</td>
<td>$\sqrt{1013\sqrt{2/3}}$</td>
<td>4</td>
<td>NTRU</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>1277</td>
<td>7879</td>
<td>$\sqrt{492}$</td>
<td>$\sqrt{1277\sqrt{2/3}}$</td>
<td>5</td>
<td>NTRU</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
</tbody>
</table>

Table 4.12: Parameter sets for third round NTRU-based schemes with secret dimension $n$, modulo $q$, small polynomials $f$ and $g$, and ring $\mathbb{Z}_q[x]/(\phi)$. The NIST column indicates the NIST security category aimed at. Each parameter set from SNTRU Prime has been assigned two security levels in the round 3 submission, and we always choose the lowest of the two.
### Table 4.13: LWE parameter sets for third round NTRU-based schemes, with dimension $n$, modulo $q$, standard deviation of the error $\sigma$, and ring $\mathbb{Z}_q[x] / (\phi)$. The NIST column indicates the NIST security category aimed at. Each parameter set from SNTRU Prime has been assigned two security levels in the round 3 submission, and we always choose the lowest of the two.
### 4.8 The Third Round

<table>
<thead>
<tr>
<th>Name</th>
<th>$n$</th>
<th>$k$</th>
<th>$q$</th>
<th>$\sigma$</th>
<th>Secret dist.</th>
<th>NIST Assumption</th>
<th>$\phi$</th>
<th>Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRYSTALS-Dilithium</td>
<td>1024</td>
<td>4</td>
<td>8380417</td>
<td>$\sqrt{24/12}$</td>
<td>$(-2,2)$</td>
<td>2 Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>SIG</td>
</tr>
<tr>
<td></td>
<td>1280</td>
<td>5</td>
<td>8380417</td>
<td>$\sqrt{80/12}$</td>
<td>$(-4,4)$</td>
<td>3 Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>SIG</td>
</tr>
<tr>
<td></td>
<td>1792</td>
<td>7</td>
<td>8380417</td>
<td>$\sqrt{24/12}$</td>
<td>$(-2,2)$</td>
<td>5 Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>SIG</td>
</tr>
<tr>
<td>CRYSTALS-Kyber</td>
<td>512</td>
<td>2</td>
<td>3329</td>
<td>$\sqrt{3/2}$</td>
<td>normal</td>
<td>1 Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>768</td>
<td>3</td>
<td>3329</td>
<td>1</td>
<td>normal</td>
<td>3 Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>4</td>
<td>3329</td>
<td>1</td>
<td>normal</td>
<td>5 Module-LWE</td>
<td>$x^{n/k} + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>Frodo</td>
<td>640</td>
<td>—</td>
<td>32768</td>
<td>2.80</td>
<td>normal</td>
<td>1 LWE</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>976</td>
<td>—</td>
<td>65536</td>
<td>2.30</td>
<td>normal</td>
<td>3 LWE</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td></td>
<td>1344</td>
<td>—</td>
<td>65536</td>
<td>1.40</td>
<td>normal</td>
<td>5 LWE</td>
<td>—</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>NTRU LPrime</td>
<td>653</td>
<td>—</td>
<td>4621</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1,1), 252)$</td>
<td>1 Ring-LWR</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>761</td>
<td>—</td>
<td>4591</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1,1), 250)$</td>
<td>2 Ring-LWR</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>857</td>
<td>—</td>
<td>5167</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1,1), 281)$</td>
<td>2 Ring-LWR</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>953</td>
<td>—</td>
<td>6343</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1,1), 345)$</td>
<td>3 Ring-LWR</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>1013</td>
<td>—</td>
<td>7177</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1,1), 392)$</td>
<td>4 Ring-LWR</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td></td>
<td>1277</td>
<td>—</td>
<td>7879</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1,1), 429)$</td>
<td>5 Ring-LWR</td>
<td>$x^n - x - 1$</td>
<td>KEM</td>
</tr>
<tr>
<td>LightSaber</td>
<td>512</td>
<td>2</td>
<td>8192</td>
<td>$\sqrt{63/12}$</td>
<td>$(-5,5)$</td>
<td>1 Module-LWR</td>
<td>$x^{n/k} + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>Saber</td>
<td>768</td>
<td>3</td>
<td>8192</td>
<td>$\sqrt{63/12}$</td>
<td>$(-4,4)$</td>
<td>3 Module-LWR</td>
<td>$x^{n/k} + 1$</td>
<td>KEM, PKE</td>
</tr>
<tr>
<td>FireSaber</td>
<td>1024</td>
<td>4</td>
<td>8192</td>
<td>$\sqrt{63/12}$</td>
<td>$(-3,3)$</td>
<td>5 Module-LWR</td>
<td>$x^{n/k} + 1$</td>
<td>KEM, PKE</td>
</tr>
</tbody>
</table>

Table 4.14: Parameter sets for third round LWE-based schemes with secret dimension $n$, Module-LWE rank $k$ (if any), modulo $q$, standard deviation of the error $\sigma$. If the LWE samples come from a Ring- or Module-LWE instance, the ring is $\mathbb{Z}_q[x]/(\phi)$. The NIST column indicates the NIST security category aimed at. Note that, for the SABER submission, we consider the binomial secret drawn from $B_\eta$ to be uniform over the interval $(-\eta^2, \eta^2)$. Each parameter set from NTRU LPrime has been assigned two security levels in the round 3 submission, and we always choose the lowest of the two.
### 4.8 The Third Round

<table>
<thead>
<tr>
<th>Name</th>
<th>$n$</th>
<th>$k$</th>
<th>$q$</th>
<th>$\sigma$</th>
<th>Secret dist.</th>
<th>NIST</th>
<th>Dual</th>
<th>uSVP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRYSALS-Dilithium</td>
<td>1024</td>
<td>4</td>
<td>8380417</td>
<td>$\sqrt{63/12}$</td>
<td>$(-1,1/2,252)$</td>
<td>1 152</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>1280</td>
<td>5</td>
<td>8380417</td>
<td>$\sqrt{80/12}$</td>
<td>$(-1,1/2,281)$</td>
<td>2 204</td>
<td>177</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1792</td>
<td>7</td>
<td>8380417</td>
<td>$\sqrt{24/12}$</td>
<td>$(-1,1/2,345)$</td>
<td>3 227</td>
<td>198</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRYSALS-Kyber</td>
<td>512</td>
<td>2</td>
<td>3329</td>
<td>$\sqrt{3/2}$</td>
<td>normal</td>
<td>1 148</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>768</td>
<td>3</td>
<td>3329</td>
<td>$\sqrt{80/12}$</td>
<td>$(-1/2,392)$</td>
<td>4 241</td>
<td>211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>4</td>
<td>3329</td>
<td>$\sqrt{24/12}$</td>
<td>$(-1/2,429)$</td>
<td>5 309</td>
<td>271</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frodo</td>
<td>640</td>
<td>—</td>
<td>32768</td>
<td>$\sqrt{3}$</td>
<td>normal</td>
<td>1 170</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>976</td>
<td>—</td>
<td>65536</td>
<td>$\sqrt{3}$</td>
<td>normal</td>
<td>3 241</td>
<td>207</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1344</td>
<td>—</td>
<td>65536</td>
<td>$\sqrt{3}$</td>
<td>normal</td>
<td>5 314</td>
<td>272</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTRU LPrime</td>
<td>653</td>
<td>—</td>
<td>4621</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 252)$</td>
<td>1 152</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>761</td>
<td>—</td>
<td>4591</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 250)$</td>
<td>2 179</td>
<td>156</td>
<td></td>
<td></td>
</tr>
<tr>
<td>857</td>
<td>—</td>
<td>5167</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 281)$</td>
<td>2 204</td>
<td>177</td>
<td></td>
<td></td>
</tr>
<tr>
<td>953</td>
<td>—</td>
<td>6343</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 345)$</td>
<td>3 227</td>
<td>198</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1013</td>
<td>—</td>
<td>7177</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 392)$</td>
<td>4 241</td>
<td>211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1277</td>
<td>—</td>
<td>7879</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 429)$</td>
<td>5 309</td>
<td>271</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LightSaber</td>
<td>512</td>
<td>2</td>
<td>8192</td>
<td>$\sqrt{63/12}$</td>
<td>$(-5,5)$</td>
<td>1 168</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>Saber</td>
<td>768</td>
<td>3</td>
<td>8192</td>
<td>$\sqrt{63/12}$</td>
<td>$(-4,4)$</td>
<td>3 244</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>FireSaber</td>
<td>1024</td>
<td>4</td>
<td>8192</td>
<td>$\sqrt{63/12}$</td>
<td>$(-3,3)$</td>
<td>5 321</td>
<td>278</td>
<td></td>
</tr>
<tr>
<td>NTRU</td>
<td>509</td>
<td>—</td>
<td>2048</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), q/8-2)$</td>
<td>1 —</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>677</td>
<td>—</td>
<td>2048</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), q/8-2)$</td>
<td>3 —</td>
<td>149</td>
<td></td>
<td></td>
</tr>
<tr>
<td>821</td>
<td>—</td>
<td>4096</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), q/8-2)$</td>
<td>5 —</td>
<td>180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>701</td>
<td>—</td>
<td>8192</td>
<td>0.79</td>
<td>$((-1/2,1), 437)$</td>
<td>4 —</td>
<td>135</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Falcon</td>
<td>512</td>
<td>—</td>
<td>12289</td>
<td>$1.17\sqrt{q/2n}$</td>
<td>normal</td>
<td>1 —</td>
<td>141</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>—</td>
<td>12289</td>
<td>$1.17\sqrt{q/2n}$</td>
<td>normal</td>
<td>5 —</td>
<td>285</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNTRU Prime</td>
<td>653</td>
<td>—</td>
<td>4621</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 288)$</td>
<td>1 —</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>761</td>
<td>—</td>
<td>4591</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 286)$</td>
<td>2 —</td>
<td>155</td>
<td></td>
<td></td>
</tr>
<tr>
<td>857</td>
<td>—</td>
<td>5167</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 322)$</td>
<td>2 —</td>
<td>176</td>
<td></td>
<td></td>
</tr>
<tr>
<td>953</td>
<td>—</td>
<td>6343</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 396)$</td>
<td>3 —</td>
<td>197</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1013</td>
<td>—</td>
<td>7177</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 448)$</td>
<td>4 —</td>
<td>211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1277</td>
<td>—</td>
<td>7879</td>
<td>$\sqrt{2/3}$</td>
<td>$((-1/2,1), 492)$</td>
<td>5 —</td>
<td>272</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.15: “Core-SVP” estimates for third round NTRU-based schemes, with dimension $n$, modulo $q$, standard deviation of the error $\sigma$, ring $\mathbb{Z}_q[x]/(\phi)$ and with $m = 2n$ samples. The NIST column indicates the NIST security category aimed at.
### 4.8 The Third Round

<table>
<thead>
<tr>
<th>Name</th>
<th>( n )</th>
<th>( k )</th>
<th>( q )</th>
<th>( \sigma )</th>
<th>Secret dist.</th>
<th>NIST</th>
<th>Dual uSVP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRYS$\text{TALS-Dilithium}$</td>
<td>1024</td>
<td>4</td>
<td>8380417</td>
<td>( \sqrt{24/12} )</td>
<td>(-2, 2)</td>
<td>2</td>
<td>127  113</td>
</tr>
<tr>
<td></td>
<td>1280</td>
<td>5</td>
<td>8380417</td>
<td>( \sqrt{80/12} )</td>
<td>(-4, 4)</td>
<td>3</td>
<td>186  166</td>
</tr>
<tr>
<td></td>
<td>1792</td>
<td>7</td>
<td>8380417</td>
<td>( \sqrt{24/12} )</td>
<td>(-2, 2)</td>
<td>5</td>
<td>275  229</td>
</tr>
<tr>
<td>CRYS$\text{TALS-Kyber}$</td>
<td>512</td>
<td>2</td>
<td>3329</td>
<td>( \sqrt{3/2} )</td>
<td>normal</td>
<td>1</td>
<td>137  108</td>
</tr>
<tr>
<td></td>
<td>768</td>
<td>3</td>
<td>3329</td>
<td>1</td>
<td>normal</td>
<td>3</td>
<td>202  166</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>4</td>
<td>3329</td>
<td>1</td>
<td>normal</td>
<td>5</td>
<td>279  232</td>
</tr>
<tr>
<td>Frodo</td>
<td>640</td>
<td>—</td>
<td>32768</td>
<td>2.80</td>
<td>normal</td>
<td>1</td>
<td>155  128</td>
</tr>
<tr>
<td></td>
<td>976</td>
<td>—</td>
<td>65536</td>
<td>2.30</td>
<td>normal</td>
<td>3</td>
<td>223  188</td>
</tr>
<tr>
<td></td>
<td>1344</td>
<td>—</td>
<td>65536</td>
<td>1.40</td>
<td>normal</td>
<td>5</td>
<td>285  247</td>
</tr>
<tr>
<td>NTRU LPrime</td>
<td>653</td>
<td>—</td>
<td>4621</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 252)</td>
<td>1</td>
<td>142  119</td>
</tr>
<tr>
<td></td>
<td>761</td>
<td>—</td>
<td>4591</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 250)</td>
<td>2</td>
<td>166  141</td>
</tr>
<tr>
<td></td>
<td>857</td>
<td>—</td>
<td>5167</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 281)</td>
<td>2</td>
<td>188  161</td>
</tr>
<tr>
<td></td>
<td>953</td>
<td>—</td>
<td>6343</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 345)</td>
<td>3</td>
<td>208  179</td>
</tr>
<tr>
<td></td>
<td>1013</td>
<td>—</td>
<td>7177</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 392)</td>
<td>4</td>
<td>223  191</td>
</tr>
<tr>
<td></td>
<td>1277</td>
<td>—</td>
<td>7879</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 429)</td>
<td>5</td>
<td>280  246</td>
</tr>
<tr>
<td>LightSaber</td>
<td>512</td>
<td>2</td>
<td>8192</td>
<td>( \sqrt{63/12} )</td>
<td>((-5, 5)</td>
<td>1</td>
<td>156  113</td>
</tr>
<tr>
<td>Saber</td>
<td>768</td>
<td>3</td>
<td>8192</td>
<td>( \sqrt{63/12} )</td>
<td>((-4, 4)</td>
<td>3</td>
<td>222  184</td>
</tr>
<tr>
<td>FireSaber</td>
<td>1024</td>
<td>4</td>
<td>8192</td>
<td>( \sqrt{63/12} )</td>
<td>((-3, 3)</td>
<td>5</td>
<td>300  253</td>
</tr>
<tr>
<td>NTRU</td>
<td>509</td>
<td>—</td>
<td>2048</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), q/8 - 2)</td>
<td>1</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>677</td>
<td>—</td>
<td>2048</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), q/8 - 2)</td>
<td>3</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>821</td>
<td>—</td>
<td>4096</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), q/8 - 2)</td>
<td>5</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>701</td>
<td>—</td>
<td>8192</td>
<td>0.79</td>
<td>((-1, 1), 437)</td>
<td>4</td>
<td>—</td>
</tr>
<tr>
<td>Falcon</td>
<td>512</td>
<td>—</td>
<td>12289</td>
<td>1.17 ( \sqrt{q/2n} )</td>
<td>normal</td>
<td>1</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>—</td>
<td>12289</td>
<td>1.17 ( \sqrt{q/2n} )</td>
<td>normal</td>
<td>5</td>
<td>—</td>
</tr>
<tr>
<td>SNTRU Prime</td>
<td>653</td>
<td>—</td>
<td>4621</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 288)</td>
<td>1</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>761</td>
<td>—</td>
<td>4591</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 286)</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>857</td>
<td>—</td>
<td>5167</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 322)</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>953</td>
<td>—</td>
<td>6343</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 396)</td>
<td>3</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>1013</td>
<td>—</td>
<td>7177</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 448)</td>
<td>4</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>1277</td>
<td>—</td>
<td>7879</td>
<td>( \sqrt{2/3} )</td>
<td>((-1, 1), 492)</td>
<td>5</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 4.16: Quantum “Core-SVP” estimates for third round NTRU-based schemes, with dimension \( n \), modulo \( q \), standard deviation of the error \( \sigma \), ring \( \mathbb{Z}_q[x]/(\phi) \), and with \( m = 2n \) samples. The NIST column indicates the NIST security category aimed ats.
4.9 Conclusion

In Tables 4.15 and 4.16 we see the 30 parameter sets analysed using the $0.292\beta$ and $0.265\beta$ BKZ cost models, under the uSVP attack and the dual attack (where appropriate).

4.9 Conclusion

In this chapter, we have considered the first, second, and third round submissions to the NIST standardisation process. We extracted the parameter sets from all first round submissions, and estimated the security of the uSVP and dual attacks (where appropriate) for each scheme, under every cost model considered as part of a submission. This allows for the security of any two schemes to be compared in a more easy manner. Moreover, we have also shown that cost models for the BKZ algorithm are not order preserving, and have provided a summary of the schemes which progressed into the second and third rounds. In the case of the third round schemes, we have provided an updated analysis on the security of these schemes under the Core-SVP model, i.e. $2^{0.292\beta}$ in the classic case, and $2^{0.265\beta}$ in the quantum case.
Homomorphic Encryption Standardisation

Contents

5.1 Introduction and Contribution ............................................. 141
  5.1.1 Bootstrapping Complexities for CKKS, BFV, and BGV ............ 142
  5.1.2 Structure and Contributions .......................................... 148
5.2 Comments on Small and Sparse-secret LWE .......................... 149
  5.2.1 Keyspace Size .......................................................... 149
  5.2.2 Secret Density ........................................................ 150
5.3 Algorithms for solving Small-secret LWE ............................. 151
  5.3.1 Hybrid-decoding Attack Assumptions ............................ 152
  5.3.2 The Hybrid-dual Attack .............................................. 153
5.4 Currently Recommended Parameters ................................. 156
5.5 Investigating Sparse-secrets ............................................. 157
  5.5.1 Using Sparse Secrets with Existing Recommended Parameter Sets 158
  5.5.2 Sparsity vs. Performance Trade-off .............................. 160
  5.5.3 Sparsity as a Proportion of Target Security: Exploration of Choices for $\zeta$ ......................................................... 160
  5.5.4 Standardising Larger Dimensions $n$ ............................. 161
5.6 Conclusion ................................................................. 166

This chapter is based on the following publication: Benjamin R. Curtis and Rachel Player. On the Feasibility and Impact of Standardising Sparse-secret LWE Parameter Sets for Homomorphic Encryption. In Proceedings of the 7th ACM Workshop on Encrypted Computing and Applied Homomorphic Cryptography (pp. 1-10). Association for Computing Machinery, 2019. Additional details have been added in this thesis.

In this chapter we investigate the security of homomorphic encryption-style LWE parameter sets against hybrid attacks. We consider the effect of secret sparsity on both the performance and security of these schemes.
The author of this thesis contributed towards (a) the writing of the paper, (b) the writing of the code used for experiments, as well as (c) running the experiments and presenting the experimental data.

5.1 Introduction and Contribution

The homomorphicencryption.org consortium have begun an effort to standardise both an API [BDH+17] and advice on secure parameter selection for LWE-based homomorphic encryption schemes. The Homomorphic Encryption Security Standard (HE Standard) [ACC+18] recommends parameter sets for usage in homomorphic encryption schemes achieving target security levels \( \lambda \in \{128, 192, 256\} \).

Recall from Chapter 2 that LWE instances can always be transformed into Normal form, where the secret follows the error distribution [ACPS09]. For error distributions \( D_\sigma \) which satisfy \( \sigma = O(\sqrt{n}) \), reductions exist from worst-case hard lattice problems to LWE [Reg05]. In this setting, hardness results for a binary secret \( s \in \{0, 1\}^n \) can be obtained at the cost of increasing the LWE dimension [BLP+13]. We note that there are currently no known hardness results for ternary secret LWE, or sparse-secret LWE.

Implementations of LWE-based homomorphic encryption libraries typically choose an error distribution which is much narrower than those considered in security reductions. Indeed, an early example uses \( \sigma = 3.19 \) [GHS12], which remains a popular choice today. We note that for \( \sigma = 3.19 \), currently known security reductions do not apply\(^1\). However, \( \sigma = 3.19 \) is used for all of the the currently recommended parameter sets in the HE Standard [ACC+18].

The HE Standard specifies parameters \((n, q, \sigma)\) achieving a security level \( \lambda \in \{128, 192, 256\} \) according to the LWE Estimator [APS15], described in Chapter 2. The parameters considered are power-of-two ring dimensions \( n \in \{1024, 2048, ..., 32768\} \), and a fixed Discrete Gaussian error distribution \( D_\sigma \) with standard deviation \( \sigma = 3.19 \). For each ring dimension \( n \), a bit-length \( \log q \) is standardised. For a given modulus \( q \), the constraint on the error distribution can be equivalently expressed as fixing the parameter \( \alpha = \frac{8}{q} \), where \( \alpha \) is defined such that \( \sigma = \frac{\alpha q}{\sqrt{2\pi}} \), as outlined in Definition 2.30. Typical secret distributions \( \chi_s \) used in homomorphic encryption schemes are not covered by these reductions either.

\(^1\)Further, we note that this is not just the case for homomorphic encryption schemes, and that some of the parameters typically considered in e.g. lattice-based KEMs (see Chapter 4) are not covered by these reductions either.
5.1 Introduction and Contribution

encryption schemes are:

1. uniform ternary, i.e. $s$ is chosen uniformly at random from the set $\{-1, 0, 1\}^n$,
2. uniform, i.e. $s$ is chosen uniformly at random from the set $\mathbb{Z}_q^n$,
3. error, i.e. each coefficient of $s$ is sampled from the error distribution $D_\sigma$, or
4. uniform binary, i.e. $s$ is chosen uniformly at random from the set $\{0, 1\}^n$.

and we note that the HE Standard supports the first three secret distributions. All major implementations of homomorphic encryption use a uniform binary or a ternary secret distribution. Moreover, many implementations use a sparse secret, for which all but a certain Hamming weight $h$ of the coefficients are zero. As an example, HEAAN \cite{HEA20}, uses by default a sparse ternary secret of Hamming weight $h = 64$. An important issue motivating the use of sparse secrets is the complexity of bootstrapping.

It is worth noting that, in this chapter, we are not concerned with the depth of computation, or any potential issues with correctness. Our goal is, for power-of-two ring dimensions $n$, to provide a maximal permissible modulus $\log q$ which allows a user to attain a desired level of security $\lambda$, as in the HE Standard \cite{ACC+18}. Determining the optimal parameters for a specific computation of multiplicative depth $L$ is beyond the scope of this work.

5.1.1 Bootstrapping Complexities for CKKS, BFV, and BGV

We provide a brief overview of the complexity of the bootstrapping procedures for the CKKS \cite{CKKS17}, BGV \cite{BGV12}, and BFV \cite{FV12} schemes. We have deliberately chosen to omit details on encoding and decoding as these details are not required. Recall that $R_q = \mathbb{Z}_q[X]/(X^n + 1)$. Throughout this section we let $\chi$ be some distribution which samples small polynomials from the ring $R_q$. We also set $R_3 = \mathbb{Z}_3[X]/(X^n + 1)$ and we recall here that $\mathbb{Z}_3 = \{-1, 0, 1\}$ and, therefore, the co-efficient vector $g$ of non-zero polynomials $g \leftarrow R_3$ satisfies $\|g\|_\infty = 1$. 

142
5.1 Introduction and Contribution

5.1.1.1 CKKS

We briefly outline the CKKS homomorphic encryption scheme as presented in [CKKS17]. Consider values $p, q_0 > 0, q_\ell = p^\ell q_0$ for $0 < \ell \leq L$ for some $L \in \mathbb{N}$, and set $P \approx q_L$.

- **KGen($1^n$)**. The key generation algorithm takes as input the security parameter and generates three keys: a secret key $sk$, a public encryption key $pk$, and a public evaluation key $ek$.
  
  - To generate the secret key, we sample a small polynomial $s \leftarrow R_3$ and set:
    
    $$sk = (1, s) \in R_3^2.$$
  
  - To generate the public key, we sample $a \leftarrow R_{q_\ell}, e \leftarrow \chi$ and set:
    
    $$pk = ((as + e) \mod q_L, a) \in R_{q_\ell}^2.$$
  
  - To generate the evaluation key we sample $a' \leftarrow R_{pq_\ell}, e' \leftarrow \chi$ and set:
    
    $$ek = (a's + e + Ps^2 \mod Pq_L, a') \in R_{pq_\ell}^2.$$

- **Enc($pk, m$)**. The encryption algorithm takes as input a public key $pk = (pk_1, pk_2) \in R_{q_\ell}^2$ and a message $m \in R$, samples $v \leftarrow R_3$ and $e_1, e_2 \leftarrow \chi$, and outputs:
  
  $$c = vpk + (m + e_1, e_2) \in R_{q_\ell}^2.$$

- **Dec($sk, c$)**. The decryption algorithm takes as input a secret key $sk = s \in R_3$ and a ciphertext $c = (c_1, c_2) \in R_{q_\ell}^2$, and outputs:
  
  $$m' \leftarrow (c_1 + c_2s) \mod q_L.$$

- **Add($c, c'$)**. The addition algorithm takes as input two ciphertexts $c, c' \in R_{q_\ell}^2$ and outputs:
  
  $$c_+ = (c_1 + c'_1, c_2 + c'_2) \mod q_\ell.$$

- **Mult($c, c'$)**. The multiplication algorithm takes as input two ciphertexts $c = (c_1, c_2), c' = (c'_1, c'_2) \in R_{q_\ell}^2$ and computes the values $d_1 = c_1c'_1, d_2 = c_1c'_2 + c_2c'_1,$ and $d_3 = c_2c'_2$. The algorithm outputs:
  
  $$c_\times = (d_1, d_2) + [P^{-1} \cdot d_3 \cdot ek] \mod q_\ell.$$
5.1 Introduction and Contribution

The decryption function in the CKKS scheme is a modular reduction, i.e. evaluating:

\[ m' \leftarrow \langle ct, sk \rangle \mod q_L. \]

In order to bootstrap we need to homomorphically evaluate the decryption function, we note that this decryption function can be represented as a scaled sine function \[ \text{CHK}^{+18} \]:

\[ \langle ct, sk \rangle \mod q_L = \frac{q}{2\pi} \sin \left( \frac{2\pi}{q_L} \cdot \langle ct, sk \rangle \right) + O(\epsilon^3 \cdot q_L), \]

when \( \| \langle ct, sk \rangle \|_{q_L} \leq \epsilon \cdot q_L \). Bootstrapping can therefore be implemented by evaluating a Chebyshev interpolant (polynomial) in degree \( d = \mathcal{O}(K + \log q) \), where \( q \) is the ciphertext modulus, and \( K \) is a constant depending on the secret distribution. This evaluation requires \( \mathcal{O}(\sqrt{d}) \) ciphertext multiplications \[ \text{CCS19a} \]. For sparse secrets with Hamming weight \( h \), the heuristic argument of \[ \text{CHK}^{+18} \] shows that we have \( K = \mathcal{O}(\sqrt{h}) \), while for a uniform ternary secret we have \( K = \mathcal{O}(\sqrt{n}) \). We note that a passive attack against CKKS, as well as potential countermeasures, has been outlined in \[ \text{LM20} \].

5.1.1.2 BGV

We briefly outline the BGV homomorphic encryption scheme as presented in \[ \text{CLP19} \]. Here, the ciphertext space is \( R_q \) and the plaintext space is \( R_t \). We note that \( w \) is a base, and \( \ell \) denotes the number of terms in the decomposition of an integer modulo \( q \) into base \( w \), i.e. \( w = \lfloor \log_w(q) \rfloor \).

- **KGen\((1^n)\).** The key generation algorithm takes as input the security parameter and generates three keys: a secret key \( sk \), a public encryption key \( pk \), and a public evaluation key \( ek \).

  - To generate the secret key, we sample a small polynomial \( s \leftarrow R_3 \) and set:
    \[ sk = s. \]

  - To generate the public key, we sample \( a \leftarrow R_q, e \leftarrow \chi \) and set:
    \[ pk = (-as + te) \mod q, a). \]

  - To generate the evaluation key, for \( 1 \leq i \leq \ell \) we sample \( a_i \leftarrow R_q \) and \( e_i \leftarrow \chi \), and set:
    \[ ek = \left\{ (-a_i s + te_i) + w^i s^2 \mod q, a_i) \right\}_{i=1}^\ell. \]
5.1 Introduction and Contribution

- **Enc(pk, m).** The encryption algorithm takes as input a public key $pk = (pk_1, pk_2) \in \mathbb{R}_q^2$ and a message $m \in \mathbb{R}_t$, samples $v \leftarrow \mathbb{R}_3$ and $e_1, e_2 \leftarrow \chi$, and outputs a ciphertext of the form:

  $$c = (pk_1 v + te_1 + m, pk_2 v + te_2) \in \mathbb{R}_q^2.$$

- **Dec(sk, c).** The decryption algorithm takes as input a secret key $sk = s \in \mathbb{R}_q$ and a ciphertext $c = (c_1, c_2) \in \mathbb{R}_q^2$, and outputs:

  $$m = (c_1 + c_2 s \mod q) \mod t.$$

- **Add(c, c').** The addition algorithm takes as input two ciphertexts $c, c' \in \mathbb{R}_q^2$ and outputs:

  $$c_+ = (c_1 + c_1' \mod q, c_2 + c_2' \mod q).$$

- **Mult(c, c').** The multiplication algorithm takes as input two ciphertexts $c, c' \in \mathbb{R}_q^2$ and computes the values $d_1 = c_1 c_1' \mod q$, $d_2 = c_1 c_2' + c_2 c_1' \mod q$, and $d_3 = c_2 c_2' \mod q$.

  From there, we output $c_\times = (d_1, d_2, d_3)$, that is:

  $$c_\times = (c_1 c_1' \mod q, c_1 c_2' + c_2 c_1' \mod q, c_2 c_2' \mod q).$$

- **Relinearize(ek, c).** The relinearisation algorithm takes as input an evaluation key $ek = \{ (ek_{i,1}, ek_{i,2}) \}_{i=1}^\ell$ and a ciphertext $c = (c_1, c_2, c_3)$, expresses $c_3$ in base $w$, i.e. computes $c_3 = \sum_{i=1}^\ell c_3^{(i)} w^i$. Computes $d_1 = c_1 + \sum_{i=1}^\ell ek_{i,1} c_3^{(i)}$ and $d_2 = c_2 + \sum_{i=1}^\ell ek_{i,2} c_3^{(i)}$, and outputs $c_{\text{relin}} = (d_1, d_2)$, that is:

  $$c_{\text{relin}} = (c_1 + \sum_{i=1}^\ell ek_{i,1} c_3^{(i)}, c_2 + \sum_{i=0}^\ell ek_{i,2} c_3^{(i)}).$$

The decryption function in the BGV scheme is:

$$m' \leftarrow (c_1 + c_2 s \mod q) \mod t,$$

and bootstrapping therefore requires the evaluation of a circuit of depth $\log(\|s\|_1) + \log t$ \cite{CH18}, where $t$ is the plaintext modulus. This evaluation requires $O(\log^{3/2} \|s\|_1 \log t + \log^2 t)$ ciphertext multiplications \cite{CH18 Table 2].
5.1 Introduction and Contribution

5.1.1.3 BFV

We briefly outline the BFV homomorphic encryption scheme as presented in [CLP19]. Here, the ciphertext space is $R_q$ and the plaintext space is $R_t$. We also define the quantity $\Delta = \lfloor q/t \rfloor$.

We note that $w$ is a base, and $\ell$ denotes the number of terms in the decomposition of an integer modulo $q$ into base $w$, i.e. $w = \lfloor \log_w(q) \rfloor$.

- $\text{KGen}(1^n)$. The key generation algorithm takes as input the security parameter and generates three keys: a secret key $sk$, a public encryption key $pk$, and a public evaluation key $ek$.
  - To generate the secret key, we sample a small polynomial $s \leftarrow R_3$ and set:
    $$sk = s.$$
  - To generate the public key, we sample $a \leftarrow R_q$, $e \leftarrow \chi$ and set:
    $$pk = (- (as + e) \mod q, a).$$
  - To generate the evaluation key, for $1 \leq i \leq \ell$ we sample $a_i \leftarrow R_q$ and $e_i \leftarrow \chi$, and set:
    $$ek = \{( -(a_is + e_i) + \ell s^2 \mod q, a_i) \}_{i=1}^\ell.$$

- $\text{Enc}(pk,m)$. The encryption algorithm takes as input a public key $pk = (pk_1, pk_2) \in R_q^2$ and a message $m \in R_t$, samples $v \leftarrow R_3$ and $e_1, e_2 \leftarrow \chi$, and outputs a ciphertext of the form: and outputs a ciphertext of the form:
    $$c = (pk_1v + e_1 + \Delta m, pk_2v + e_2) \in R_q^2.$$

- $\text{Dec}(sk,c)$. The decryption algorithm takes as input a secret key $sk = s \in R_3$ and a ciphertext $c = (c_1, c_2) \in R_q^2$, and outputs:
    $$m' = \left\lfloor \frac{t}{q} (c_1 + c_2s) \mod q \right\rfloor \mod t.$$

- $\text{Add}(c, c')$. The addition algorithm takes as input two ciphertexts $c, c' \in R_q^2$ and outputs:
    $$c_+ = (c_1 + c'_1 \mod q, c_2 + c'_2 \mod q).$$
5.1 Introduction and Contribution

- **Mult**($c, c'$). The multiplication algorithm takes as input two ciphertexts $c, c' \in R_q^2$ and computes the values $d_1 = \left\lfloor \frac{t}{q} c_1 c'_1 \right\rfloor \mod q$, $d_2 = \left\lfloor \frac{t}{q} (c_1 c'_2 + c_2 c'_1) \right\rfloor \mod q$, and $d_3 = \left\lfloor \frac{t}{q} c_2 c'_2 \right\rfloor \mod q$. From there, we output $c_x = (d_1, d_2, d_3)$, that is:

$$c_x = \left( \left\lfloor \frac{t}{q} c_1 c'_1 \right\rfloor \mod q, \left\lfloor \frac{t}{q} (c_1 c'_2 + c_2 c'_1) \right\rfloor \mod q, \left\lfloor \frac{t}{q} c_2 c'_2 \right\rfloor \mod q \right).$$

- **Relinearize**($ek, c$). The relinearisation algorithm takes as input an evaluation key $ek = \{(ek_{i,1}, ek_{i,2})\}_{i=1}^\ell$ and a ciphertext $c = (c_1, c_2, c_3)$, expresses $c_3$ in base $w$, i.e. computes $c_3 = \sum_{i=1}^\ell c_3^{(i)} w^i$. Computes $d_1 = c_1 + \sum_{i=0}^\ell ek_{i,1} c_3^{(i)} \mod q$ and $d_2 = c_2 + \sum_{i=1}^\ell ek_{i,2} c_3^{(i)} \mod q$, and outputs $c_{\text{relin}} = (d_1, d_2)$, that is:

$$c_{\text{relin}} = (c_1 + \sum_{i=1}^\ell ek_{i,1} c_3^{(i)} \mod q, c_2 + \sum_{i=1}^\ell ek_{i,2} c_3^{(i)} \mod q).$$

The decryption function in the BFV scheme can be written as:

$$m' \leftarrow \left\lfloor \frac{(c_1 + c_2 \cdot s \mod q)}{\Delta} \right\rfloor,$$

which requires the evaluation of a circuit of depth $\log (\|s\|_1) + \log \log t$. This, in turn, requires $O((\log \|s\|_1 + \log t)^{1/2} \log \|s\|_1)$ ciphertext multiplications [CH18, Table 2].

5.1.1.4 Summary

For sparse ternary secret with Hamming weight $h$, we have $\|s\|_1 = h$, whereas for a uniform ternary secret we expect $\|s\|_1 = O(n)$. Current implementations for bootstrapping in CKKS, BGV or BFV use sparse secrets for efficiency reasons [CH18, CHK+18]. We also note that there are works aimed towards improving bootstrapping techniques for non-sparse keys [BMTPH20].

We summarise this discussion in Table 5.1.

5.1.1.5 Sparse Secrets

Sparse secret distributions are not currently supported in the HE Standard. This is likely due to the loss of security as compared to a uniform ternary secret for a fixed set of LWE parameters $(n, q, \sigma)$, as well as uncertainty over the attack landscape for these distributions.
5.1 Introduction and Contribution

<table>
<thead>
<tr>
<th>Scheme</th>
<th>CKKS</th>
<th>BGV</th>
<th>BFV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decryption circuit depth</td>
<td>$O(K + \log q)$</td>
<td>$\log(| s |_1) + \log t$</td>
<td>$\log(| s |_1) + \log \log t$</td>
</tr>
<tr>
<td>Ciphertext multiplications</td>
<td>$O(\sqrt{d})$</td>
<td>$O(\log^{3/2} | s |_1 + \log^{1/2} | s |_1 \cdot \log t + \log^2 t)$</td>
<td>$O((\log | s |_1 + \log t)^{1/2} \log | s |_1)$</td>
</tr>
<tr>
<td>Parameter sizes when $s \leftarrow B^-$</td>
<td>$K = O(\sqrt{\ell})$</td>
<td>$| s |_1 = O(\alpha)$</td>
<td>$| s |_1 = O(\alpha)$</td>
</tr>
<tr>
<td>Parameter sizes when $s \leftarrow B^-_h$</td>
<td>$K = O(\sqrt{\ell})$</td>
<td>$| s |_1 = h$</td>
<td>$| s |_1 = h$</td>
</tr>
</tbody>
</table>

Table 5.1: A summary of the bootstrapping complexities for CKKS, BGV and BFV.

The loss of security is intuitive, as this corresponds to shrinking the size of the keyspace. Moreover, several attacks are known which can exploit the sparsity of an LWE secret [How07, Alb17, HHC19]. As discussed in Chapters 3 and 4, these attack techniques trade-off the probability of guessing $\tau$ secret components against the lower cost of solving a lattice problem in dimension $(d - \tau)$.

Another way in which the recommended parameter sets in the HE Standard do not always reflect implementation choices is in the maximal supported dimension $n = 2^{15}$. For example, many implementations of bootstrapping, such as [CHK+18, CCS19, HHC19], choose ring dimension $n = 2^{16}$. In addition, advanced applications of homomorphic encryption, such as logistic regression training [KSK+18, KSW+18], have been reported using dimension $n = 2^{16}$ or $n = 2^{17}$. Such large dimensions are used to support the choice of $q$, which is chosen to be large enough to allow for evaluation of a circuit of a specific depth.

5.1.2 Structure and Contributions

The discussion from Section 5.1.1.5 motivates the widening of the recommended parameter sets to include sparse secrets, or parameter sets for larger dimension $n > 2^{15}$. In this chapter we consider such possible extensions.

An outline of the structure and contributions of this chapter is as follows. In Section 5.2 we make some comments on small and sparse secret LWE. In Section 5.3 we discuss algorithms used to solve small and sparse LWE, including introducing the hybrid dual attack [CHHS19]. In Section 5.4 we outline some of the parameter sets recommended in the current variant of the HE Standard. In Section 5.5 we assess the impact on security and performance of using a sparse secret of Hamming weight $h$ instead of a uniform ternary secret, for various choices of $h$. In Section 5.5.4 we show how the methodology of the Standard could be used
5.2 Comments on Small and Sparse-secret LWE

to select parameters with larger power-of-two dimension \( n \geq 2^{16} \). In Section 5.6 we discuss open problems and future work.

5.2 Comments on Small and Sparse-secret LWE

Definition 2.34 outlines some small-secret distributions of interest. Recall that the uniform ternary distribution is denoted by \( B^{-} \), and the uniform binary distribution is denoted by \( B^{+} \).

5.2.1 Keyspace Size

When considering combinatorial attacks against LWE, the size of the keyspace is an important quantity, since this determines the maximal size of the guessing set in combinatorial attacks. To illustrate this, we consider the following examples:

1. when the secret is drawn from \( \mathbb{Z}_{q}^{n} \), the size of the keyspace is \( \|S_{\mathbb{Z}_{q}^{n}}\| = q^{n} \),
2. when the secret is drawn from \( B^{-} \), the size of the keyspace is \( \|S_{B^{-}}\| = 3^{n} \),
3. when the secret is drawn from \( B^{-}_{h} \), the size of the keyspace is \( \|S_{B^{-}_{h}}\| = \binom{n}{h} \cdot 2^{h} \),
4. when the secret is drawn from \( B^{+} \), the size of the keyspace is \( \|S_{B^{+}}\| = 2^{n} \), and
5. when the secret is drawn from \( B^{+}_{h} \), the size of the keyspace is \( \|S_{B^{+}_{h}}\| = \binom{n}{h} \).

In Figure 5.1 we highlight the size of the keyspace \( B^{-}_{h} \) and \( B^{+}_{h} \), for each potential value of \( h \) when the associated ring dimension is \( n = 1024 \). If \( h = 64 \) then the size of the keyspace is \( \approx 2^{405} \), whereas if the secret is is drawn from \( B^{-} \) then the size of the keyspace is \( \approx 2^{1623} \). We note that the LWE Estimator, as well as our scripts for hybrid attacks, assume that uniformly random ternary secrets have fixed Hamming weight \( h = \lfloor \frac{2n}{3} \rfloor \), and that uniformly random binary secrets have fixed Hamming weight \( h = \lfloor \frac{n}{2} \rfloor \). For the examples considered in Figure 5.1 we have that \( h = \lfloor \frac{2 \times 1024}{3} \rfloor = 683 \) and \( \|S_{B^{-}_{683}}\| = \binom{1024}{683} \cdot 2^{683} \approx 2^{1618} \), and \( h = \lfloor \frac{1024}{2} \rfloor = 512 \) and \( \|S_{B^{+}_{512}}\| = \binom{1024}{512} \approx 2^{1018} \).
5.2 Comments on Small and Sparse-secret LWE

5.2.2 Secret Density

Keeping a fixed Hamming weight (e.g. $h = 64$) for a variety of ring dimensions means that the density $\kappa = \frac{h}{n}$ of the secret decreases as $n$ grows. One approach to scaling sparse secrets is to fix the density parameter:

$$\kappa = \frac{h}{n}.$$ 

For example, we could consider $\kappa = \frac{1}{16}$ such that:

$$(n, h) \in \{(1024, 64), (2048, 128), (4096, 256), \ldots \}.$$ 

This follows the approach of several submissions to the ongoing NIST post-quantum standardisation effort: for example, Lizard [CKLS18] uses $h = \frac{3}{8}$, i.e. $\kappa = \frac{1}{8}$. For larger ring dimensions used in homomorphic encryption libraries this approach can lead to a large Hamming weight $h$, 

![Figure 5.1: Example LWE (secret) keyspace sizes with $n = 1024$ for binary, ternary, fixed-weight binary, and fixed-weight ternary secrets.](image-url)
leading to a more expensive bootstrapping operation. For example, for a ring dimension \( n = 32768 \), choosing \( \kappa = \frac{1}{16} \) would require \( h = 2048 \).

Another approach is to fix the ratio between the Hamming weight \( h \) of the secret and the security parameter \( \lambda \), i.e. fix the value \( \zeta \), where:

\[
\zeta = \frac{h}{\lambda}.
\]

This approach would mean that, for each target security level \( \lambda_{\text{target}} \), the value of the Hamming weight \( h \) for every ring dimension \( n = 2^k \) is fixed. For example, if \( \zeta = 1 \), then for a fixed security level \( \lambda \) we consider secrets of Hamming weight \( h = \lambda \). Such an approach means that the (theoretical) complexity of bootstrapping would remain the same for each dimension \( n \) with an associated security level \( \lambda \). In this work we consider the second approach, i.e. fixing the value of \( \zeta = \frac{h}{\lambda} \).

### 5.3 Algorithms for solving Small-secret LWE

As discussed in Chapter 2, the concrete security of LWE-based parameter sets is typically determined by considering the best known attacks. That is, given an LWE parameter set \((n, \alpha, q)\) and a corresponding secret distribution, we set \( \lambda \) to be the logarithm of the running time of the fastest attack. The current version of the HE Standard [ACC+18] uses the LWE Estimator to determine parameters, based on the running time of three attacks: uSVP, decoding and dual. Hybrid attacks [How07, CHHS19] are typically among the most competitive in the case of sparse secrets, although they are not currently supported by the LWE Estimator, and therefore have not been considered in the latest version of the HE Standard.

Following the HE Standard, in this chapter we only consider sieving-based cost models for BKZ. Specifically, we view BKZ as a black box which runs in (pre-quantum) time:

\[
T_{\text{BKZ}}(\beta, d) = 2^{0.292\beta + 16.4 + \log(8d)},
\]

and, if instantiated with quantum algorithms to solve SVP, runs in time:

\[
T_{\text{BKZ}}(\beta, d) = 2^{0.265\beta + 16.4 + \log(8d)}.
\]

Estimates for the primal decoding attack [LP11, LN13] reported by the LWE Estimator do not assume state-of-the-art techniques, hence may be inaccurate and are often not competitive.
5.3 Algorithms for solving Small-secret LWE

More precisely, the Estimator currently assumes the decoding attack is implemented with the Nearest Planes algorithm [LP11] as opposed to the more efficient pruned enumeration [LN13]. Moreover, the Estimator does not consider combinatorial techniques for the decoding attack as considered in [ACW20] and Chapter 3. As an example, for the example parameter set $n = 653, q = 4621, \sigma \approx \sqrt{2/3}, \chi_s = B_{100}$ considered in Section 3.4, the decoding estimate provided by the LWE Estimator estimates the complexity as $2^{229.9}$, whereas the batch-BDD techniques outlined in Chapter 3 report a complexity of $2^{186.1}$. For this reason, we do not report dec estimates in this chapter.

We consider the uSVP, hybrid-decoding, dual and hybrid-dual attacks on LWE. For uSVP and dual, we consider the small-secret variant of these attacks as described in Chapter 4. For the hybrid-decoding attack, as discussed in Chapter 3, we outline the assumptions considered in this Chapter in Section 5.3.1. We describe the hybrid-dual attack in Section 5.3.2.

5.3.1 Hybrid-decoding Attack Assumptions

As discussed in Chapter 3, an analysis of the hybrid-decoding attack requires the usage of several assumptions. In Chapter 3 we considered multiple sets of assumptions. For clarity, we outline the assumptions considered in this chapter:

- The output GSO basis shape of lattice reduction is given by the Geometric Series Assumption [Sch03].
- The (heuristic) success probability of Babai’s Nearest Plane algorithm follows the analysis in [Wun18] and is given by:
  $$ p \approx \prod_{1 \leq i \leq d} \left(1 - \frac{2}{B\left(\frac{d-1}{2}, \frac{1}{2}\right)} \cdot \int_{\min(r_i, 1)}^{1} \left(1 - t^2\right)^{(d-3)/2} dt\right), $$
  where $d$ is the dimension of the lattice under consideration, and $r_i = \|b_i^*\|/\|v\|$ where $\|v\|$ is the (expected) norm of the target vector, and $B(\cdot, \cdot)$ denotes the Beta function [OLBC10].
- The cost of running Babai’s Nearest Plane algorithm in a lattice of dimension $d$ is given by $T_{babai} = \frac{d^2}{2^{d-1}}$.
- The meet-in-the-middle search phase provides a square-root speed-up as compared to an exhaustive search.
5.3 Algorithms for solving Small-secret LWE

- The associated meet-in-the-middle probability is set to be $p_{\text{mitm}} = 1$, thus providing an explicit underestimate of security.
- The meet-in-the-middle search phase has access to unlimited memory.

Under these assumptions, the cost of the hybrid-decoding attack is given by:

$$
\min_{\beta, \tau, d, t} \frac{T_{\text{BKZ}}(\beta, d)}{p} + \frac{d^2}{2^{10n}} \cdot \sqrt{\sum_{i=0}^{t} \|S_i\|},
$$

where $\beta$ is the BKZ blocksize, $d$ is the dimension in which lattice reduction is performed, $\tau$ is the guessing dimension (i.e. the number of guessed components of the secret), $t$ is the maximal Hamming weight considered in the search space, and $\sum_{i=0}^{t} \|S_i\|$ is the size of the search space, i.e. the number of points on which we decode. Here $p$ is the probability of success of Babai’s Nearest Plane algorithm, and $p_i$ is the probability that the guessed part of the secret has Hamming weight $i$. Estimates for the cost of the hybrid-decoding attack are generated using custom code\[2\].

5.3.2 The Hybrid-dual Attack

Albrecht’s variant of the dual attack, as described in Chapter \[4\] was recently adapted by Cheon et al. \[CHHS19\] to include a meet-in-the-middle step in the combinatorial search phase of the attack, giving rise to a hybrid-dual attack. It is shown in \[CHHS19\] that when fixing a maximal memory of $2^{80}$, the hybrid dual attack outperforms the dual attack for certain homomorphic encryption-style parameter sets with a sparse ternary secret. Cheon et al. \[CHHS19\] provide a script\[3\] that can be used to estimate the security of a given parameter set against the hybrid dual attack. Recall from Chapter \[4\] that the small-secret variant of the dual attack proceeds by performing lattice reduction on the lattice $\Lambda''$, where:

$$
\Lambda'' = \left\{ (v, w) \in \mathbb{Z}^m \times \left( \frac{1}{c} \cdot \mathbb{Z} \right)^n \left| vA \equiv cw \pmod{q} \right. \right\}.
$$

For an LWE instance $(A, b = As + e)$, with sparse secret of Hamming weight $h$, the crux of the meet-in-the-middle process is based on considering the noisy relationship:

$$
As_1 \approx b - As_2
$$

\[2\] This code can be found at [https://github.com/bencrts/hybrid_attacks/hybrid_decoding.py](https://github.com/bencrts/hybrid_attacks/hybrid_decoding.py)

\[3\] The script is available at [https://github.com/swanhong/HybridLWEAttack](https://github.com/swanhong/HybridLWEAttack)

153
for a pair \( s_1, s_2 \) satisfying \( s_1 + s_2 = s \). The attack begins by considering the list which constructs \( A v_1 \) for candidate values \( v_1 \) of \( s_1 \):

\[
\mathcal{T} = \{ A v_1 \mid v_1 \in \{-1, 0, 1\}^n : HW(v_1) \leq h' \},
\]

which is constructed for some \( h' \leq h \). Note that \( h' \) is an attack parameter and can be optimised over. A hash table \( \mathcal{H} \), initialised as \( 2^m \) empty linked lists with indexes in \( \{0, 1\}^m \), is generated from this list by appending an entry \( t \in \mathcal{T} \) into the position indexed with \( sgn(t) \) where:

\[
sgn(t)_i = \begin{cases} 
1 & \text{if } t_i \in [0, q/2) \\
0 & \text{else} 
\end{cases}.
\]

After this process is completed for all \( t \in \mathcal{T} \), we begin a search over \( \mathcal{H} \). The search proceeds by checking whether \( b - A v_2 \) is close to the list \( \mathcal{T} \) for candidate choices \( v_2 \), where the closeness depends on the size of the error vector \( e \). That is, we are interested in the norm of:

\[
\| A s_1 - (b - A s_2) \|_{\infty} = \| e \|_{\infty},
\]

and the meet-in-the-middle process is successful if \( \| e \|_{\infty} < B \), for some \( B \). This is referred to as a \( B \)-noisy collision.

**Definition 5.1 (B-noisy collision [CHHS19])** For a vector \( a \in \mathbb{Z}_q^m \), a vector \( t \in \mathbb{Z}_q^m \) is referred to as a \( B \)-noisy collision of \( a \) if \( \| a - t \|_{\infty} \leq B \) for some \( B < q/2 \).

In the analysis of [CHHS19], the parameter \( B \) is chosen as:

\[
B = \left( 2 + \frac{1}{\sqrt{2\pi}} \sqrt{\frac{m}{m+n} \cdot \alpha q \| y \|} \right),
\]

where \( y \) is the short vector retrieved via lattice reduction on the lattice \( \Lambda'' \). If we can find a \( B \)-noisy collision of \( A s_1 \), for a candidate \( s_1 \), then we can find \( b - A s_2 \) and therefore solve decisional LWE.

The search phase takes as input the constructed hash table \( \mathcal{H} \), a query point \( q \) (which corresponds to \( b - A v_2 \) for some candidate \( v_2 \) of \( s_2 \)) as well as a distance bound \( B \). Defining:

\[
sgn'(t)_i = \begin{cases} 
sgn(t_i) & \text{if } t_i \in V_B \\
0 & \text{else} 
\end{cases},
\]

where \( V_B = [-q/2 + B, -B] \cup [B, q/2 - B] \), we then consider:

\[
sgn'(q),
\]
where $q$ is a candidate for $s_2$. $\text{sgn}'(q)$ represents a list of binary strings generated by replacing each $\times$ by 0 or 1. Then, for each element $x$ in this list, we check if $H$ contains a set indexed by $x$. If this is the case, then for every $t$ within this list we check if:

$$\|q - t\|_\infty < B,$$

and, if this is the case, we return this vector.

To summarise, the hybrid-dual attack consists of three phases:

1. a lattice reduction phase, where we perform lattice reduction on the lattice $\Lambda''$,
2. a hash-table construction phase, where we generate the hash table $H$, and
3. a searching phase, where we search for $B$-noisy collisions using the hash table $H$.

The total cost of phases 2. and 3. is the sum of:

(a) constructing the list $T$,
(b) generating the hash table $H$, and
(c) the cost of the search algorithm applied to the candidate secrets.

These costs are computed as $N_T(n^2 + m)$ for steps (a) and (b) and $O(N_q 2^{4mB/q})$ for step (c) [CHHS19].

5.3.2.1 Hybrid-dual Assumptions

In this thesis, we do not use the script provided by [CHHS19] 4. Instead, we conservatively assume that the meet-in-the-middle process admits a square-root speed-up with no probability loss – thus providing an explicit underestimate of security, for the purposes of generating conservative parameters. This matches our approach with the hybrid-decoding attack (i.e. $P_{mitm} = 1$). To do this, we consider Albrecht’s attack script [Est20] and, in the search phase, balance the cost of searching with the square of the lattice reduction cost (as opposed to simply balancing the cost of searching with the cost of lattice reduction). The changes to the source code can be found in Figure 5.2.

4This is because the script was not designed to be used for non-sparse (i.e. uniform-ternary) secrets.
5.4 Currently Recommended Parameters

```python
if postprocess:
    repeat = current['repeat']
    dim = current['d']

    for i in range(1, k):
        cost_post_i = (2 * repeat * dim * k) + (repeat * binomial(k, i) * (b-a)**i * i)
        probability_i = success_probability_drop(n, h, k, i, rotations=rotations)
        if cost_post + cost_post_i >= cost_lat**2:
            postprocess = i
            break

    cost_post += cost_post_i
    probability += probability_i

    current['rop'] = cost_lat + sqrt(cost_post)
```

Figure 5.2: Changes applied to the dual attack source code inside the LWE Estimator to produce our hybrid-dual estimates. This replaces lines 1944-1959 in commit 428d6ea of the LWE Estimator.

5.4 Currently Recommended Parameters

In Tables 5.2 we reproduce a subset of the parameter sets \((n, \log q, \alpha)\) recommended in the current version of the HE Standard [ACC+18] to achieve target security level \(\lambda\) for \(\lambda \in \{128, 192, 256\}\), for power-of-two ring dimensions between 1024 and 32768 and for a secret having coefficients chosen uniformly in \([-1, 0, 1]\). Table 5.2 reports the estimated cost of running the uSVP, decoding, and dual attacks on these parameter sets under the sieving cost model:

\[
T(\beta, d) = 2^{0.292\beta+16.4+\log(8d)}
\]
5.5 Investigating Sparse-secrets

<table>
<thead>
<tr>
<th>n</th>
<th>log q</th>
<th>α</th>
<th>usvp</th>
<th>decoding</th>
<th>dual</th>
<th>λ_{target}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>27</td>
<td>8/q</td>
<td>131.6</td>
<td>160.2</td>
<td>138.7</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>54</td>
<td>8/q</td>
<td>129.7</td>
<td>144.4</td>
<td>134.2</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>109</td>
<td>8/q</td>
<td>128.1</td>
<td>134.9</td>
<td>129.9</td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td>218</td>
<td>8/q</td>
<td>128.5</td>
<td>131.5</td>
<td>129.2</td>
<td>128</td>
</tr>
<tr>
<td>16384</td>
<td>438</td>
<td>8/q</td>
<td>128.1</td>
<td>129.9</td>
<td>129.0</td>
<td></td>
</tr>
<tr>
<td>32768</td>
<td>881</td>
<td>8/q</td>
<td>128.5</td>
<td>129.1</td>
<td>128.5</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>19</td>
<td>8/q</td>
<td>193.0</td>
<td>259.5</td>
<td>207.7</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>37</td>
<td>8/q</td>
<td>197.5</td>
<td>233.0</td>
<td>207.8</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>75</td>
<td>8/q</td>
<td>194.7</td>
<td>212.2</td>
<td>198.5</td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td>152</td>
<td>8/q</td>
<td>192.2</td>
<td>200.4</td>
<td>194.6</td>
<td>192</td>
</tr>
<tr>
<td>16384</td>
<td>305</td>
<td>8/q</td>
<td>192.1</td>
<td>196.2</td>
<td>193.2</td>
<td></td>
</tr>
<tr>
<td>32768</td>
<td>611</td>
<td>8/q</td>
<td>192.7</td>
<td>194.2</td>
<td>193.7</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>14</td>
<td>8/q</td>
<td>265.6</td>
<td>406.4</td>
<td>293.8</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>29</td>
<td>8/q</td>
<td>259.1</td>
<td>321.7</td>
<td>273.5</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>58</td>
<td>8/q</td>
<td>260.4</td>
<td>292.6</td>
<td>270.1</td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td>118</td>
<td>8/q</td>
<td>256.7</td>
<td>270.4</td>
<td>260.6</td>
<td>256</td>
</tr>
<tr>
<td>16384</td>
<td>237</td>
<td>8/q</td>
<td>256.9</td>
<td>264.2</td>
<td>259.8</td>
<td></td>
</tr>
<tr>
<td>32768</td>
<td>476</td>
<td>8/q</td>
<td>256.4</td>
<td>260.2</td>
<td>258.2</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: Currently standardised LWE parameters at the 128-, 192- and 256-bit security level for uniform ternary secret specified in [ACC+18, Table 1] and estimates of their security against usvp, decoding, and dual attacks under the BKZ cost model $T(\beta,d) = q^{0.292\beta+16.4+\log(8d)}$, where $\beta$ is the blocksize and $d$ is the dimension. The best performing attack for each parameter set is highlighted in bold.

5.5 Investigating Sparse-secrets

It may be desirable to extend the HE Standard to include parameter sets with sparse secret distributions, due to the complexity of bootstrapping outlined in Section 5.1.1. In this section, we consider the feasibility and impact of including sparse ternary secret distributions in the HE Standard. To begin, in Figure 5.3 we compare the four attack techniques discussed in
this chapter for the parameter set $n = 1024, q = 2^{40}$ and $\sigma \approx 3.2$ with a sparse ternary secret of Hamming weight $h \in \{64, 128, 256, 512\}$, to see which attacks perform best as the density of the secret changes.

Figure 5.3: A comparison of the usvp, dual, hybrid—dual and hybrid—decoding attacks under the BKZ cost model $T(\beta, d) = 2^{0.292\beta^3 + 16.4 + \log(8d)}$, for the parameter set $n = 1024, q = 2^{40}$ and $\sigma \approx 3.2$ with a sparse ternary secret with a variety of Hamming weights $h \in \{64, 128, 256, 512\}$.

5.5.1 Using Sparse Secrets with Existing Recommended Parameter Sets

Next, we consider the impact of using a sparse ternary secret of Hamming weight $h = 128$ for the sets of parameters $(n, \log q, \alpha)$ as recommended in Table 5.2 for uniform ternary secret. That is, we take the parameter sets currently recommended in the HE standard for uniform-ternary secrets, and swap the secret distribution to be sparse ternary of Hamming weight $h = 128$, in order to determine the effect on the security. In Table 5.3 we report the output of the LWE Estimator giving an estimate of the concrete security of the parameter sets $(n, \log q, \alpha, h)$ with a sparse ternary secret of Hamming weight $h = 128$. For consistency,
5.5 Investigating Sparse-secrets

we have used the same sieving cost model $T(\beta, d) = 2^{0.292\beta+16.4+\log(8d)}$ which is used to generate [ACC+18] Table 1.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\log q$</th>
<th>$\alpha$</th>
<th>$h$</th>
<th>usvp</th>
<th>dual</th>
<th>$\lambda_{\text{target}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>27</td>
<td>8/q</td>
<td>128</td>
<td><strong>124.9</strong></td>
<td>127.8</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>54</td>
<td>8/q</td>
<td>128</td>
<td><strong>122.0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>109</td>
<td>8/q</td>
<td>128</td>
<td>124.9</td>
<td><strong>117.9</strong></td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td>218</td>
<td>8/q</td>
<td>128</td>
<td>126.4</td>
<td><strong>117.2</strong></td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>438</td>
<td>8/q</td>
<td>128</td>
<td>127.0</td>
<td><strong>117.1</strong></td>
<td></td>
</tr>
<tr>
<td>32768</td>
<td>881</td>
<td>8/q</td>
<td>128</td>
<td>127.5</td>
<td><strong>116.5</strong></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>19</td>
<td>8/q</td>
<td>128</td>
<td><strong>178.2</strong></td>
<td>179.2</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>37</td>
<td>8/q</td>
<td>128</td>
<td><strong>173.7</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>75</td>
<td>8/q</td>
<td>128</td>
<td>186.5</td>
<td><strong>165.2</strong></td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td>152</td>
<td>8/q</td>
<td>128</td>
<td>186.4</td>
<td><strong>167.5</strong></td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>305</td>
<td>8/q</td>
<td>128</td>
<td>187.7</td>
<td><strong>159.1</strong></td>
<td></td>
</tr>
<tr>
<td>32768</td>
<td>881</td>
<td>8/q</td>
<td>128</td>
<td>188.8</td>
<td><strong>161.9</strong></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>14</td>
<td>8/q</td>
<td>128</td>
<td><strong>235.4</strong></td>
<td>238.5</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>29</td>
<td>8/q</td>
<td>128</td>
<td>231.9</td>
<td><strong>217.3</strong></td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>58</td>
<td>8/q</td>
<td>128</td>
<td>234.3</td>
<td><strong>210.2</strong></td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td>118</td>
<td>8/q</td>
<td>128</td>
<td>232.8</td>
<td><strong>207.9</strong></td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>237</td>
<td>8/q</td>
<td>128</td>
<td>233.8</td>
<td><strong>195.8</strong></td>
<td></td>
</tr>
<tr>
<td>32768</td>
<td>476</td>
<td>8/q</td>
<td>128</td>
<td>234.6</td>
<td><strong>214.7</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3: Impact of using a sparse ternary secret of Hamming weight $h = 128$, using the currently standardised LWE parameter sets at the target 128-, 192- and 256-bit security level for uniform ternary secret specified in Table 5.2. Estimates of the security of each parameter set against usvp and dual attacks under the BKZ cost model $T(\beta, d) = 2^{0.292\beta+16.4+\log(8d)}$ are presented, where $\beta$ is the blocksize and $d$ is the dimension. The best performing attack for each parameter set is highlighted in bold.

It can be seen from Table 5.3 that introducing a sparse secret of Hamming weight $h = 128$, there is a noticeable security loss, by up to 11 bits at the target 128-bit security level, by up to 32 bits at the target 192-bit security level, and by up to 60 bits at the target 256-bit security level. Table 5.3 does not take into account hybrid attacks, which are likely to be
5.5 Investigating Sparse-secrets

competitive, and hence the security loss may be even greater.

5.5.2 Sparsity vs. Performance Trade-off

In Table 5.4 we illustrate the effect of using a sparse ternary secret with various Hamming weights \( h \) on the bit size \( \log q \) with \( n = 1024 \) and \( \sigma = 3.19 \) fixed. For comparison, we also note the bit-length \( \log Q \) which is currently recommended to achieve target security level \( \lambda \) for the same \( n, \sigma \) with a uniform ternary secret in the HE Standard. A smaller modulus \( q \) may impact on practical performance of the schemes. For example when we need to ensure that \( q \) is large enough to support the full computation, to ensure correct decryption. The lower \( q \) required by introducing a sparse secret may necessitate moving to a higher dimension \( n \) to support the computation, which in turn will be slower.

5.5.3 Sparsity as a Proportion of Target Security: Exploration of Choices for \( \zeta \)

In Table 5.5 we present an exploration of possible choices for the value \( \zeta = \frac{h}{\lambda} \), illustrating the reduction in bit-length \( \log q \) required to retain the desired level of security when using a sparse ternary secret compared to a uniform ternary secret. Table 5.5 uses the cost model \( T(\beta, d) = 2^{0.292\beta+16.4+\log(8d)} \), i.e. BKZ.sieve available in the LWE Estimator, and considers the following attacks: uSVP, dual, hybrid-decoding, and hybrid-dual. We provide as a point of comparison \( \log Q \), the bit-length of the modulus \( Q \) currently recommended in \[ ACC+18 \] for the given parameters \( (n, \sigma = 3.19) \) with uniform ternary secret at target security level \( \lambda \).

Table 5.5 indicates that a choice such as \( \zeta = 1 \) gives a reasonable trade-off between performance and security. In this case, we can retain secure parameters with at most a 24% drop in the bit-length of the modulus \( \log q \) compared to that recommended at target security level \( \lambda \) for the same values of \( n \) and \( \sigma = 3.19 \) and uniform ternary secret. This choice corresponds to a sparse ternary secret with Hamming weight \( h \in \{128, 192, 256\} \), depending on the desired security level, which allows for more efficient bootstrapping when compared to uniform-ternary secrets.
5.5 Investigating Sparse-secrets

<table>
<thead>
<tr>
<th>$h$</th>
<th>$n$</th>
<th>$\lambda$</th>
<th>log $q$</th>
<th>usvp</th>
<th>dual</th>
<th>log $Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1024</td>
<td>128</td>
<td>24</td>
<td>131.0</td>
<td>133.8</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192</td>
<td>16</td>
<td>193.8</td>
<td>198.3</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td>11</td>
<td>265.2</td>
<td>279.1</td>
<td>14</td>
</tr>
<tr>
<td>128</td>
<td>1024</td>
<td>128</td>
<td>26</td>
<td>130.4</td>
<td>131.4</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192</td>
<td>17</td>
<td>198.6</td>
<td>199.8</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td>12</td>
<td>269.6</td>
<td>279.7</td>
<td>14</td>
</tr>
<tr>
<td>256</td>
<td>1024</td>
<td>128</td>
<td>27</td>
<td>128.1</td>
<td>134.1</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192</td>
<td>18</td>
<td>194.9</td>
<td>208.6</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td>13</td>
<td>267.0</td>
<td>293.2</td>
<td>14</td>
</tr>
<tr>
<td>512</td>
<td>1024</td>
<td>128</td>
<td>27</td>
<td>132.0</td>
<td>137.6</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192</td>
<td>18</td>
<td>202.9</td>
<td>219.0</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td>14</td>
<td>261.3</td>
<td>292.6</td>
<td>14</td>
</tr>
<tr>
<td>$\lceil \frac{2n}{3} \rceil$</td>
<td>1024</td>
<td>128</td>
<td>27</td>
<td>133.3</td>
<td>138.7</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192</td>
<td>19</td>
<td>194.7</td>
<td>207.7</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td>14</td>
<td>265.6</td>
<td>293.8</td>
<td>14</td>
</tr>
</tbody>
</table>

Table 5.4: Bit-length log $q$ of moduli required to provide target security level $\lambda$, for $\lambda \in \{128, 192, 256\}$, for various secret densities. We note that the LWE Estimator treats uniform ternary secrets as ternary fixed-weight secrets with Hamming weight $h = \lceil \frac{2n}{3} \rceil$. The best performing attack for each parameter set is highlighted in bold.

5.5.4 Standardising Larger Dimensions $n$

With current progress in applied homomorphic encryption it is becoming necessary to work in dimensions larger than $n = 2^{15}$, the largest dimension currently supported in the HE Standard. Several recent papers [CHK+18, CCS19a, HHC19, KSK+18] have reported implementations in dimension $n = 2^{16}$, and an implementation in dimension $n = 2^{17}$ was reported in [KSW+18]. A natural extension of the current standard would therefore be to standardise parameter sets for dimension $n = 2^k$ for some $k \geq 16$, since power-of-two $n$ remain the most widely used in practice. Moreover, power-of-two $n$ enables convenient coefficient-wise error sampling and
5.5 Investigating Sparse-secrets

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\lambda$</th>
<th>$\log q^{(\zeta=1/2)}$</th>
<th>$\log q^{(\zeta=3/4)}$</th>
<th>$\log q^{(\zeta=1)}$</th>
<th>$\log q^{(\zeta=3/2)}$</th>
<th>$\log Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>128</td>
<td>14</td>
<td>19</td>
<td>21</td>
<td>23</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>10</td>
<td>13</td>
<td>14</td>
<td>16</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>2048</td>
<td>128</td>
<td>27</td>
<td>37</td>
<td>41</td>
<td>46</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>19</td>
<td>26</td>
<td>29</td>
<td>32</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>15</td>
<td>19</td>
<td>22</td>
<td>24</td>
<td>29</td>
</tr>
<tr>
<td>4096</td>
<td>128</td>
<td>55</td>
<td>74</td>
<td>83</td>
<td>92</td>
<td>109</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>37</td>
<td>52</td>
<td>57</td>
<td>64</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>30</td>
<td>39</td>
<td>44</td>
<td>49</td>
<td>58</td>
</tr>
<tr>
<td>8192</td>
<td>128</td>
<td>111</td>
<td>148</td>
<td>171</td>
<td>186</td>
<td>218</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>84</td>
<td>100</td>
<td>114</td>
<td>130</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>60</td>
<td>79</td>
<td>89</td>
<td>98</td>
<td>118</td>
</tr>
<tr>
<td>16384</td>
<td>128</td>
<td>223</td>
<td>300</td>
<td>342</td>
<td>377</td>
<td>438</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>157</td>
<td>201</td>
<td>232</td>
<td>265</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>115</td>
<td>161</td>
<td>176</td>
<td>202</td>
<td>237</td>
</tr>
<tr>
<td>32768</td>
<td>128</td>
<td>496</td>
<td>619</td>
<td>699</td>
<td>767</td>
<td>881</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>350</td>
<td>411</td>
<td>479</td>
<td>523</td>
<td>611</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>263</td>
<td>313</td>
<td>361</td>
<td>408</td>
<td>476</td>
</tr>
</tbody>
</table>

Table 5.5: The reduction in bit-length $\log q$ of the modulus $q$ required to retain the desired level of security against the dual, usvp, hybrid-dual and hybrid-decoding attacks, under our assumptions, when using a sparse ternary secret parameterised by $\zeta = \frac{h}{\lambda}$ compared to a uniform ternary secret. The lattice reduction cost model is $T(\beta, d) = 2^{0.292\beta+16.4+\log(8d)}$, and a (conservative) estimate for both the hybrid-dual and hybrid-decoding attacks are obtained by considering a square-root speed-up in the search space, and ignoring any meet-in-the-middle probabilities.

...would require no change to the currently standardised error distribution.

For $n = 2^{16}$, it is straightforward to apply the methodology used in the current HE Standard [ACC+18].
5.5 Investigating Sparse-secrets

i.e. to use the LWE Estimator to find an appropriate log \( q \) to meet security requirements for fixed \( \sigma = 3.19 \) and a currently standardised secret distribution. We present the results of such an analysis for a uniform ternary secret distribution in Table 5.6, which gives an estimate of the security of the proposed parameter sets \((n = 2^{16}, \sigma = 3.19, \log q)\) against the usvp and dual attacks under a sieving lattice reduction cost model.

<table>
<thead>
<tr>
<th>( \lambda )</th>
<th>( n )</th>
<th>\log ( q )</th>
<th>usvp</th>
<th>dual</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>65536</td>
<td>1782</td>
<td>128.3</td>
<td>128.4</td>
</tr>
<tr>
<td>192</td>
<td>65536</td>
<td>1242</td>
<td>192.5</td>
<td>192.0</td>
</tr>
<tr>
<td>256</td>
<td>65536</td>
<td>963</td>
<td>256.7</td>
<td>257.7</td>
</tr>
</tbody>
</table>

Table 5.6: Required bit-length \( \log q \) of moduli required to attain target security level \( \lambda \) under the usvp and dual attacks, with \( \lambda \in \{128, 192, 256\} \), for dimension \( n = 65536 \), under the sieving-based cost model \( T(\beta, d) = 2^{\beta\cdot2^{16}+16.4+\log(8d)} \).

For \( n \geq 2^{17} \) the same methodology works in theory, although it can become cumbersome to run hybrid attack estimates many times for such very large parameter sets. To find suitable moduli \( q \) achieving target security \( \lambda \) for higher values of \( n \), we can extrapolate using the data we already have using the apparent linear relationship between \( n \) and \( \log q \). That is, for a fixed \( \sigma \), \( n \log q \) is essentially constant for a fixed target security level. This means we can easily extrapolate entries for larger values of \( n \), without having to explicitly run new experiments. We illustrate in Figure 5.4 this for a sparse ternary secret of fixed Hamming weight \( h = \lambda \) (i.e. \( \zeta = 1 \)). When considering pre-quantum estimates, we can represent \( \log(q) \) as a linear function of \( n \) by extrapolation from the data in Table 5.5:

\[
\log_{\text{sieve}}(\lambda, \zeta=(128,1))(n) = 0.021370n - 3.601989
\]
\[
\log_{\text{sieve}}(\lambda, \zeta=(192,1))(n) = 0.014630n - 3.139303, \text{ and}
\]
\[
\log_{\text{sieve}}(\lambda, \zeta=(256,1))(n) = 0.011007n - 1.184080.
\]

These linear models were found using the \texttt{find_fit} function in SageMath [S+20]. For readability, we round the coefficients to six decimal places in each case.
5.5 Investigating Sparse-secrets

Figure 5.4: Extrapolation to $n = 65536$ and $n = 131072$ using the data from Table 5.5 for the value $\zeta = 1$. Here, we consider the lattice reduction cost model $T_{\text{BKZ}}(\beta, d) = 2^{0.292\beta + 16.4 + \log(8d)}$ and extrapolate using the SageMath function `find_fit`. Note that the solid lines represent values covered by data points, and the dashed lines represent extrapolation.

We can use these extrapolations to generate values for $\log(q)$. For $n = 65536$, we see that:

\[
\lfloor \log_{\text{sieve}}^{(\lambda, \zeta)=(128, 1)}(65536) \rfloor = 1396, \\
\lfloor \log_{\text{sieve}}^{(\lambda, \zeta)=(192, 1)}(65536) \rfloor = 955, \\
\lfloor \log_{\text{sieve}}^{(\lambda, \zeta)=(256, 1)}(65536) \rfloor = 720,
\]

that is, for $\lambda = 128$ we have $\log(q) = 1396$, for $\lambda = 192$ we have $\log(q) = 955$, and for $\lambda = 256$ we have $\log(q) = 720$. For $n = 131072$, we see that:

\[
\lfloor \log_{\text{sieve}}^{(\lambda, \zeta)=(128, 1)}(131072) \rfloor = 2797, \\
\lfloor \log_{\text{sieve}}^{(\lambda, \zeta)=(192, 1)}(131072) \rfloor = 1914, \\
\lfloor \log_{\text{sieve}}^{(\lambda, \zeta)=(256, 1)}(131072) \rfloor = 1441,
\]
5.5 Investigating Sparse-secrets

that is, for \( \lambda = 128 \) we have \( \log(q) = 2797 \), for \( \lambda = 192 \) we have \( \log(q) = 1914 \), and for \( \lambda = 256 \) we have \( \log(q) = 1441 \).

5.5.4.1 Standardising the Standard Deviation \( \sigma \)

The choice of standard deviation \( \sigma = 3.19 \) is somewhat arbitrary, and we could consider including wider standard deviations. In Table 5.7 we consider the impact of various standard deviations \( \sigma \in \{0.80, 2.90, 3.20, 32, 320, 3200\} \) on the bit-length \( \log q \) of the required modulus to achieve a target level of security \( \lambda \) for fixed \( \lambda = 128 \), and fixed weight ternary secret with Hamming weight \( h = 128 \). As an illustrative example, we choose the case of \( n = 8192 \). We consider security against the usvp and dual attacks under the sieving-based cost model \( T(\beta, d) = 2^{0.292\beta + 16.4 + \log(8d)} \) for lattice reduction. However, we note that, as \( \sigma \) decreases, it may be beneficial to guess components of the error vector, and the LWE Estimator does not consider such techniques.

<table>
<thead>
<tr>
<th>( \sigma )</th>
<th>( n )</th>
<th>( \log q )</th>
<th>usvp</th>
<th>dual</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.80</td>
<td>8192</td>
<td>194</td>
<td>141.8</td>
<td>128.5</td>
</tr>
<tr>
<td>2.90</td>
<td>8192</td>
<td>196</td>
<td>141.7</td>
<td>128.5</td>
</tr>
<tr>
<td>3.20</td>
<td>8192</td>
<td>196</td>
<td>141.8</td>
<td>128.6</td>
</tr>
<tr>
<td>32</td>
<td>8192</td>
<td>204</td>
<td>138.2</td>
<td>128.4</td>
</tr>
<tr>
<td>320</td>
<td>8192</td>
<td>203</td>
<td>141.5</td>
<td>128.1</td>
</tr>
<tr>
<td>3200</td>
<td>8192</td>
<td>211</td>
<td>137.9</td>
<td>128.1</td>
</tr>
</tbody>
</table>

Table 5.7: Maximal moduli \( q \) required to attain target security level \( \lambda = 128 \) for various values of \( \sigma \). In each case, we consider a ring dimension of \( n = 8192 \) and a fixed weight ternary secret with Hamming weight \( h = 128 \).

In general, we can see that as the value of \( \sigma \) increases, we attain a larger permissible value of \( \log(q) \) for the fixed value of \( n = 8192 \).
5.6 Conclusion

In this chapter, we have considered the current state of the Homomorphic Encryption Security Standard. We have considered the impact and feasibility of considering sparse secret parameter sets in future variants of the standard. We note that the current version of the HE Standard does not consider hybrid attacks, and urge that any parameter sets are analysed against hybrid attacks before standardisation.

We have shown that, when sparse secrets are considered, the LWE modulus $q$ can require a significant reduction in size in order to attain the target security level. This can be problematic in homomorphic encryption schemes, where we are looking for a large modulus $q$ in order to be able to carry out computations correctly. The attack landscape for sparse-secret LWE is fast moving, with new contributions $[\text{CHHS19}, \text{EJK20}]$ emerging in recent years.
Chapter 6

Privacy-preserving Kriging Interpolation

Contents

6.1 Introduction and Contribution ........................................... 168
6.2 Kriging Interpolation ..................................................... 171
   6.2.1 Overview of the Kriging Procedure ............................... 172
   6.2.2 The Variogram .................................................... 172
   6.2.3 The Normal Equations ........................................... 174
   6.2.4 Toy Example ..................................................... 175
6.3 Private Outsourced Kriging Interpolation ............................. 180
6.4 Our Techniques .......................................................... 184
   6.4.1 The Canonical Normal Equations ................................. 184
6.5 Our Construction ........................................................ 187
6.6 Discussion ............................................................... 189
   6.6.1 Implementation .................................................... 190
6.7 Conclusion .................................................................. 192

This chapter is based on the following publication:


In this chapter we present a solution which enables private outsourced Kriging interpolation. We outline techniques that can be used to “factor out” sensitive parameters from the system of equations which need to be solved by the server. Using these technique, alongside additively homomorphic encryption, we can securely outsource Kriging interpolation.
6.1 Introduction and Contribution

The author of this thesis contributed towards (a) the design of the private outsourced kriging interpolation scheme, and (b) the writing of the paper. No contribution was made towards the implementation, and therefore minimal details are provided in this chapter.

6.1 Introduction and Contribution

Kriging is a spatial interpolation algorithm which provides the best linear unbiased prediction (BLUP) of an observed phenomenon, by taking a weighted average of samples within a specified neighbourhood. It is widely used in areas such as geo-statistics where, as an example, it may be used to predict the quality of mineral deposits at an unobserved location based on previous measurements. There are many variants of Kriging, but we focus on the widely used Ordinary Kriging variant, and refer to it simply as Kriging throughout this chapter. Kriging can be outsourced to a cloud service provider, although measurements and predictions can be highly sensitive and must therefore be protected.

In this chapter, we present a method for the private outsourcing of Kriging interpolation. We use a modified variant of the Kriging algorithm in combination with homomorphic encryption (as described in Chapter 2). Our solution allows crucial information relating to measurement values to be hidden from the cloud service provider. Crucially, we only require additively homomorphic encryption which allows for an efficient solution to be built.

Kriging is a form of linear interpolation that predicts the value $z_0^*$ of some phenomenon at an unobserved location $q_0 = (x_0, y_0)$ in a two-dimensional region. The quality of Kriging prediction relies on the parameters considered, as well as the variogram model which describes the spatial continuity of the data. These parameters, and the variogram model, are chosen prior to interpolation. The Kriging prediction is then formed as a weighted sum of prior measurements, where measurements closer to the query point $q_0$ are given a greater weight in the sum than those which are further away. This reflects the assumption that measurements taken at nearby locations are more likely to be ‘similar’ than measurements taken further apart.

Kriging was designed with geo-statistical applications in mind (e.g. to predict the best location to mine based on the mineral deposits found at previous measurements within a region), but has also found applications in a variety of settings including remote sensing [RDB91], real-
6.1 Introduction and Contribution

estate appraisal [KH14], and computer simulations [Kle09]. Kriging has been identified as a
good candidate process to be outsourced, based on the practical and legislative requirements
of industrial users [Cla16, Ing16].

Many users may need access to a Kriging prediction service (indeed, legal frameworks may
require such data to be shared amongst relevant authorities [Eur07]). A secured storage server
may be preferable to distributing copies of the entire dataset to each authorised user, especially
when datasets are large and/or user devices are constrained. Furthermore, Kriging might
need to be performed over data owned by multiple organizations, with an independent cloud
service provider performing processing duties on behalf of all concerned parties.\footnote{In this case, it may be preferable to use Multi-key Homomorphic Encryption [LTV12, PS16, CCS19b, CDKS19]. We do not consider this setting in our work, and assume that there is a single data owner $C$.} Centralised
outsourcing also makes sense when remote sensors take frequent measurements and push the
results to a central database.

Consider a client $C$ that owns a Kriging dataset (a set of measurements taken at various
locations) which it wishes to outsource to an honest-but-curious cloud service provider $S$. In
the honest-but-curious model, the cloud service provider $S$ follows the protocol as described,
but also attempts to learn as much as possible from the resulting transcript of messages.
Our results do not translate to the malicious setting, since we have no way of verifying the
computations made by the sever in our solution. A maliciously-secure protocol for the private
outsourcing of Kriging interpolation is left for future work. The client $C$ would like to make
use of both the storage and computational power of the server $S$ to make a Kriging prediction
service on its dataset available to multiple users. Further, other data generating nodes may
be authorised by $C$ to add/remove data (measurements) to/from the outsourced dataset.

A trivial solution consists of encrypting all data using a symmetric encryption scheme and
using the server only for storage-as-a-service. To compute a Kriging prediction, all relevant
data is retrieved, decrypted and computed on locally. Unfortunately, this solution may not be
efficient, particularly if client devices have limited computational power or storage capacity.
Moreover, this solution requires high bandwidth during queries. This may be an issue if, for
example, a surveyor in the field requires an on-line Kriging prediction service. In this case,
mobile data services may be expensive, intermittently available, or slow.

An alternative is to compute the entire Kriging process on encrypted data by encrypting
all data using FHE. Unfortunately, Kriging involves several computations that are currently

169
6.1 Introduction and Contribution

challenging when using FHE, including computing square roots and natural exponentiations. It is certainly possible to outsource the Kriging process and protect all information using FHE. However, this results in prohibitively high encryption and decryption costs as well as a large amount of interactivity and local computation, which may diminish the benefits of the client/server setting. Preliminary experiments using the SEAL library [Mic20] (without optimization of code or parameter choices) did not yield promising results when computing a Kriging prediction using a dataset of only three measurements.

Our proposed solution uses additively homomorphic encryption to outsource Kriging interpolation efficiently. We make a trade-off by protecting only the most sensitive parameters. That is, we protect the measurement values in the dataset, the generated Kriging predictions and (a subset of) the variogram parameters chosen by the client. We do not hide locations (of prior measurements or queries), noting that prior measurement locations may well be externally observable (e.g. if measurements come from previous mining operations). We discuss issues surrounding data leakage in more detail in Section 6.6.

Our main contribution is to show that the Kriging process can be adapted such that the sensitive variogram parameters may be ‘factored out’ from the online computation provided by the server, whilst the remainder of the Kriging computation may be performed on encrypted measurement values using an additively homomorphic encryption scheme. We thus gain a practical, efficient, and secure solution to privately outsource Kriging. An outline of our protocol is as follows:

1. C uploads an encrypted dataset, consisting of n measurements, to S.
2. S prepares the Kriging dataset for future queries. This process includes plaintext operations that are also necessary in an unprotected outsourced Kriging scheme.
3. C makes a query to S requesting a Kriging prediction at a location \((x_0, y_0)\), which is done in plaintext with virtually no cost.
4. S computes the interpolation on encrypted measurements.
5. C decrypts the result.

Cryptographically-secured Kriging has been studied previously in a different setting, where a server owns a dataset and clients may query the dataset at a previously unsampled location [TP13]:

170
the queried location and resulting prediction should be private from the server, whilst the
dataset held by the server should be private from the client. Two solutions are proposed
in [TP13] which, unlike our solution, support only one variogram model and require high
communication complexity, interactivity and local computation. The first is based on creating
random ‘dummy’ queries to hide the queried location, and using an oblivious transfer protocol
to hide predictions for all but the legitimate query location. The second solution uses the
Paillier encryption scheme [Pai99] in an interactive protocol requiring multiple round-trips
between client and server. In [TP14] collaborative private Kriging was investigated, where
users combine their datasets to gain more accurate Kriging predictions.

The remainder of this chapter is structured as follows. In Section 6.2 we describe the Kriging
interpolation process. In Section 6.3 we define our system model and analyse the required
security properties of each piece of data in our setting. In Section 6.4 we introduce the idea
of a canonical variogram, which we use in our construction to allow the server to compute a
Kriging prediction without relying on the sensitive parameters. Our construction is given in
Section 6.5 and we discuss its performance in Section 6.6. Finally, in Section 6.7 we conclude
and outline some potential directions for future work.

6.2 Kriging Interpolation

This section outlines the background theory of Kriging interpolation. For more detail, see
[CD99, Cre92, Kri51, Wac13]. To highlight which objects are to be kept secret from the server,
we use brackets, e.g. [x] represents that we want to keep the value x secret (via encryption
or other techniques). The Kriging process starts with a set of measurements taken at some
locations in a spatial region, and produces predicted measurements at unsampled locations.

We denote this spatial region by $R \subset \mathbb{R}^2$, and the locations of prior measurements by
$P = (r_1, r_2, \ldots, r_n)$, where each $r_i = (x_i, y_i) \in R$. The Euclidean distance\footnote{We
note that some Kriging datasets consider the Great Circle distance, i.e. the distance between
two points along the surface of a sphere, as the distance metric; we do not consider such cases in this chapter.} between two locations
$r_i, r_j \in R$ is:

$$d(r_i, r_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$ 

We refer to the set of taken measurements by $S = ([z_1], [z_2], \ldots, [z_n])$, where $[z_i]$ corresponds
to a measurement taken at the corresponding location $r_i \in P$. We define the Kriging dataset
to be the tuple $(P, S)$, which consists of all measurement values and their corresponding
locations. That is:
\[(P, S) = ((r_1, r_2, \ldots, r_n), ([z_1], [z_2], \ldots, [z_n])).\]

The Kriging process allows a client to query an arbitrary location \(r_0 \in R\), in order to receive a prediction \([z_0^*]\) of the true value \([z_0]\) that would be measured at \(r_0\).

### 6.2.1 Overview of the Kriging Procedure

Informally, Kriging consists of three phases:

1. **Computing the experimental variogram**: one of the underlying assumptions of the Kriging process is that two measurements of a phenomenon will be similar when measured at nearby locations. Using the sampled dataset, one can plot the experimental variogram to show the dependence between measurements sampled at locations at certain distances \(h\).

2. **Fitting a variogram model**: the experimental variogram is not usually sufficient to use in the Kriging prediction directly, since there may not be sampled data at every required distance. Therefore, one chooses a variogram model and empirically selects model parameters to fit a curve to the points of the experimental variogram.

3. **Computing the prediction**: using the variogram, we can determine the appropriate weights for each measurement (based on the distance between each measurement and the queried location). The Kriging prediction is then computed as a weighted sum of the measured samples.

### 6.2.2 The Variogram

Let \(N(h)\) be the set of all pairs of measurements taken approximately distance \(h\) apart, that is:
\[N(h) = \{([z_i], [z_j]) \mid d(r_i, r_j) \in (h - \Delta, h + \Delta)\}\[3\]}

\[\Delta\] The approximation tolerance \(\Delta\) can be increased when the Kriging dataset does not include enough sample points at a close enough distance.
The experimental variogram $\gamma^*$ plots, for every distance $h$ such that $N(h) \neq \emptyset$:

$$\gamma^*(h) = \frac{1}{2N(h)} \sum_{(z_i, z_j) \in N(h)} ([z_i] - [z_j])^2. \tag{6.1}$$

A suitable variogram function $\gamma : \mathbb{R}^\geq 0 \to \mathbb{R}$ must satisfy a set of conditions [CD99, Cre92]. The most commonly used models require:

- $\gamma(0) = 0$,
- $\gamma(h)$ is positive and bounded, and
- the existence of the limits $\lim_{h \to 0^+} \gamma(h)$ and $\lim_{h \to \infty} \gamma(h)$.

These models are parametrized by the following four variables:

1. the **nugget effect** $[\eta]$: the limit of $\gamma(h)$ as $h \to 0^+$,
2. the **sill** $[\nu]$: the limit of $\gamma(h)$ as $h \to \infty$,
3. the **range** $\rho$: which controls how fast $\gamma(h)$ approaches $[\nu]$ as $h$ increases, and
4. the **partial sill** $[\mu]$: the difference between the sill and the nugget, i.e. $[\mu] = [\nu] - [\eta]$.

Typically, a variogram model is chosen from a set of standard parametric variogram models. This model is then fitted to the experimental variogram by empirically adjusting the nugget effect, sill, and range parameters. The most common choices of bounded variogram models are, for $h > 0$:

1. the **bounded linear model**: $\gamma(h) = [\nu] - ([\nu] - [\eta]) \left(1 - \frac{h}{\rho}\right) \cdot 1_{(0,\rho)}(h)$,
2. the **exponential variogram model**: $\gamma(h) = [\nu] - ([\nu] - [\eta]) e^{-h/\rho}$,
3. the **spherical variogram model**: $\gamma(h) = [\nu] - ([\nu] - [\eta]) \left(1 - \frac{3h}{2\rho} + \frac{h^3}{4\rho^3}\right) \cdot 1_{(0,\rho)}(h)$, and
4. the **Gaussian variogram model**: $\gamma(h) = [\nu] - ([\nu] - [\eta]) e^{-h^2/\rho^2}$,

where $1_I(x) = 1$ if $x \in I$, and $1_I(x) = 0$ otherwise.
6.2 Kriging Interpolation

6.2.3 The Normal Equations

Let $\gamma$ be one of the variogram models discussed in Section 6.2.2 with parameters chosen to fit the experimental variogram. To construct the best linear unbiased predictor of the phenomenon at a queried location $r_0 = (x_0, y_0) \in \mathbb{R}$, we first form the Kriging matrix $K \in \mathbb{R}^{(n+1) \times (n+1)}$ with elements:

- $K_{i,j} = \gamma(d(r_i, r_j))$ for $1 \leq i, j \leq n$,
- $K_{n+1,i} = K_{i,n+1} = 1$ for $i \neq n + 1$, and
- $K_{n+1,n+1} = 0$.

That is,

$$K = \begin{pmatrix}
\gamma(d(r_1, r_1)) & \gamma(d(r_1, r_2)) & \ldots & \gamma(d(r_1, r_n)) & 1 \\
\gamma(d(r_2, r_1)) & \gamma(d(r_2, r_2)) & \ldots & \gamma(d(r_2, r_n)) & 1 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\gamma(d(r_n, r_1)) & \gamma(d(r_n, r_2)) & \ldots & \gamma(d(r_n, r_n)) & 1 \\
1 & 1 & \ldots & 1 & 0
\end{pmatrix} \in \mathbb{R}^{(n+1) \times (n+1)}.$$

Define $v \in \mathbb{R}^{n+1}$ with $v_i = \gamma(d(r_0, r_i))$ for $1 \leq i \leq n$, and $v_{n+1} = 1$. Finally, let $w \in \mathbb{R}^{(n+1)}$ satisfy $Kw = v$:

$$K \begin{pmatrix} w_1 \\
w_2 \\
\vdots \\
w_n \\
w_{n+1} \end{pmatrix} = \begin{pmatrix} \gamma(d(r_0, r_0)) \\
\gamma(d(r_0, r_1)) \\
\vdots \\
\gamma(d(r_0, r_n)) \\
1 \end{pmatrix}.$$

The (Ordinary) Kriging prediction $[z^*_0]$ of the measured phenomenon at the location $r_0$ is computed as the weighted sum of the sampled measurements, with the weights defined by $w$. Specifically, we have:

$$[z^*_0] = \sum_{i=1}^{n} w_i [z_i].$$
6.2 Kriging Interpolation

The set of linear equations defined by $K$, $w$, and $v$ are known as the Normal Equations. They are derived by ensuring that the induced linear predictor is unbiased (this is done by forcing the first $n$ weights to sum to one, i.e. $\sum_{i=1}^{n} w_i = 1$) while minimizing the variance of the induced linear predictor \cite{Wac13}. The resulting minimized variance $[\sigma^2_0]$ is called the (Ordinary) Kriging variance, and it is described by the following expression:

$$[\sigma^2_0] = w_{n+1} + \sum_{i=1}^{n} w_i \gamma(d(r_0, r_i)).$$

The Kriging variance allows for the construction of confidence intervals for each prediction and this describes the error associated with the prediction. We define a variogram function to be non-degenerate if $[\eta] \neq [\nu]$, and we restrict our attention to non-degenerate variograms in this chapter. Using the degenerate variogram (also called the nugget effect variogram \cite{Wac13}) results in the average Kriging predictor $[z^*_0] = \sum_{i=1}^{n} [z_i]/n$ at all unsampled locations $r_0 \notin P$, with Kriging variance $[\sigma^2_0] = n + 1$.

6.2.4 Toy Example

Throughout this chapter, we make use of a small example dataset for illustrative purposes. To do this, we use the PyKrige library \cite{PYK20} following in a similar fashion to the example code provided with the library. Specifically, we expand their Ordinary Kriging example code and utilise their example dataset. This example code considers the toy dataset, consisting of five points, outlined in Table 6.1.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$[z]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>1.2</td>
<td>[0.47]</td>
</tr>
<tr>
<td>1.9</td>
<td>0.6</td>
<td>[0.56]</td>
</tr>
<tr>
<td>1.1</td>
<td>3.2</td>
<td>[0.74]</td>
</tr>
<tr>
<td>3.3</td>
<td>4.4</td>
<td>[1.47]</td>
</tr>
<tr>
<td>4.7</td>
<td>3.8</td>
<td>[1.74]</td>
</tr>
</tbody>
</table>

Table 6.1: The toy dataset used throughout this section, as in the ordinary kriging example used in the PyKrige library \cite{PYK20}. The $x_i$, $y_i$ values are coordinates of the measurement $[z_i]$.

In our example, we choose to set the value of $\Delta$ to be 0. Therefore, since the size of our
6.2 Kriging Interpolation

def compute_experimental_variogram(data):
    vals = []
    Nh = 1
    variogram_vals = []
    h = []
    for i in range(len(data)):
        for j in range(i+1,len(data)):
            xij = data[i][0] - data[j][0]
            yij = data[i][1] - data[j][1]
            zij = data[i][2] - data[j][2]
            hij = sqrt(xij**2 + yij**2)
            h.append(hij)
            vals.append(zij**2)
    variogram_vals = [1/(2*Nh) * z for z in vals]
    return (h, variogram_vals)

Figure 6.1: Code used to plot the experimental variogram of a dataset, such as the example dataset presented in Table 6.1.

dataset is five points, we note that there are at most \( \binom{5}{2} = 10 \) distinct sets of the form:

\[
N(h_k) = \{([z_i], [z_j]) \mid d(r_i, r_j) = h_k\},
\]

for \( 1 \leq k \leq 10 \). For our dataset, it turns out that each value \( h_k \) is distinct, and we therefore have exactly ten sets \( \{N(h_k)\}_{k=1}^{10} \). We plot \( h \) against \( \gamma^*(h) \) in Figure 6.2.

We can use the custom code outlined in Figure 6.1 to determine a good variogram model. We note that PyKrige allows the usage of binning when computing a variogram, which corresponds to a selection of \( \Delta \neq 0 \) and considering Euclidean distances in the range \((h - \Delta, h + \Delta)\). As noted above, we consider \( \Delta = 0 \) in our example. We begin by plotting the experimental variogram according to Equation 6.1 in Figure 6.1.

We can also use Pykrige to plot the experimental variogram, which we do using the example code provided with the PyKriige library, and fit a linear variogram to the data (since Figure 6.2 shows that this is a good model for fitting). Recall that the linear model is of the form:

\[
\gamma(h) = [\eta] - ([\eta] - [\nu])(1 - \frac{h}{\rho}),
\]

176
and PyKrige fits a model of the form:

$$\gamma(h) = 5.2517 \cdot 10^{-11} + 0.1168h,$$

which corresponds to choices of $\rho = 1$, $[\eta] \approx [0.1168]$, $[\nu] = [5.2517 \cdot 10^{-11}]$. Note that PyKrige does not consider a range in their linear models, so we have omitted the Indicator function used in Section 6.2.2. For this variogram we perform Kriging interpolation on a $10 \times 10$ grid between the values $x, y \in [0, 5]$, and the results are displayed in Figure 6.4. We can, of course, change the granularity of this prediction and predict over an $(r \times s)$ grid for any $r, s \in \mathbb{Z}$. 

Figure 6.2: The experimental variogram for our example dataset, presented in Table 6.1, with $\Delta = 0$
6.2 Kriging Interpolation

Figure 6.3: A linear model fitted to the experimental variogram using the Pykrige library, for our example dataset presented in Table 6.2.
6.2 Kriging Interpolation

Figure 6.4: Example interpolation grid using the linear model fitted to the experimental variogram presented in Figure 6.3.
6.3 Private Outsourced Kriging Interpolation

We now introduce the setting that we consider in this chapter. We have a client $C$ that owns a Kriging dataset $(P, S)$, as defined in Section 6.2, along with a variogram $\gamma$ computed as in Section 6.2.4. We also have an honest-but-curious server $S$ that is willing to perform outsourced Kriging on behalf of the client $C$. Finally, we have additional users $U$ that are authorised by $C$ to make Kriging queries to $S$, and we may also consider additional data generating nodes (e.g. several remote sensors placed in locations of interest) that may update the outsourced dataset by producing additional measurement data or removing previous (e.g. outdated) measurements. The requirements of each party are as follows:

- The data owner must choose the variogram model to be used, as well as the associated parameters. They upload a Kriging dataset to the server, and should be able to update data and request Kriging predictions.

- The data users may request Kriging predictions and update data. Here, update data refers to the addition of a new measurement $z_i$ to the set of measurements $S$, and the addition of the corresponding location $r_i$ to the set of locations $P$. After an update has taken place, all future Kriging predictions use the updated dataset.

- The data generating nodes should only be able to update data.

- The honest-but-curious server should only be able to perform Kriging predictions, and should do so without learning the data used in the computation.

We now informally describe the protocol, which proceeds as follows:

1. The data owner $C$ chooses the variogram model to be used and runs the Outsource algorithm to generate the (protected) dataset to be sent to the server, as well as ‘keys’ that are issued to authorise a party to update the outsourced dataset, or to perform Kriging queries.

2. Upon receipt of the protected data, the server may run the Setup algorithm to process the data and perform any necessary pre-computation. After this step has been completed, the system is ready to accept queries and perform the associated predictions.
3. The data owner, or an authorised data user (in possession of the query key), may request a Kriging prediction at a specified location by running the Query algorithm to generate a query token $Q$.

4. This query token $Q$ is sent to the server, who runs the Interpolate algorithm using the processed database to generate an encrypted prediction and an encoding of the Kriging variance (the estimation of the error in the prediction).

5. A party authorised to perform queries may learn the prediction and variance by running the Decrypt algorithm.

6. To dynamically update the outsourced dataset, an authorised party (in possession of the update key) may run the AddRequest algorithm on a specified location $r'$ and measurement $[z']$, or the DeleteRequest algorithm on a specified location $r$. These algorithms produce an addition token $\alpha_{r',[z']}$ or deletion token $\delta_{r}$, respectively, that is sent to the server.

7. Upon receipt of an addition or deletion token, the server runs the Add or Delete algorithm, respectively, to update the database accordingly.

We note that, throughout execution of all of the algorithms discussed above, the server cannot access the underlying predictions due to the encryption provided by the homomorphic encryption scheme. In this chapter, we assume that any user authorised to generate a Kriging query is also permitted to update the dataset. If this is not the case, then the construction can be modified to include a digital signature computed on any addition or deletion token, where the signing key is contained in the update key (and not the query key). The server should be trusted to reject any tokens that do not have a valid signature (which is the case since we are working in the honest-but-curious model). In this case, only users in possession of the private signature key would be able to update the dataset. We now formalise this discussion.

**Definition 6.1 (Private Outsourced Kriging Interpolation Scheme)** A private outsourced Kriging interpolation scheme is made up of the following algorithms:

- $(U,K,Q,C) \leftarrow \text{Outsource}(1^\lambda,P,S,\gamma)$: a probabilistic algorithm run by $C$ which takes as input a security parameter $\lambda$, the Kriging dataset which is made up of measurement locations $P$ and measurement values $S$, and the chosen variogram $\gamma$. It produces:
6.3 Private Outsourced Kriging Interpolation

- an update key $UK$ that may be used to update the outsourced dataset, which is given to parties authorised by $C$ to update the dataset,
- a query key $QK$ which may be used to form Kriging queries, which is kept by $C$, and
- an outsourceable dataset $C$ that may be transmitted to the server.

- $DB \leftarrow \text{Setup}(C)$: a deterministic algorithm run by $S$ which takes as input the outsourceable dataset $C$. This algorithm enables $S$ to perform any necessary processing that will enable it to compute Kriging predictions, and produces a processed outsourced dataset $DB$.

- $Q \leftarrow \mathcal{Q}(QK, r_0)$: a probabilistic algorithm run by $C$ or a data user in $U$ which takes as input a query key $QK$ and a query location $r_0 = (x_0, y_0) \in R$ for which a Kriging prediction should be computed. The output is a query token $Q$ which is sent to $S$.

- $(Z_0, \sigma_0^2) \leftarrow \text{Interpolate}(Q, DB)$: a deterministic algorithm run by $S$ that, given a query token $Q$ and the database $DB$, returns an encrypted Kriging interpolation $Z_0$ and the partially computed Kriging variance $\sigma_0^2$.

- $(z_0^*, \sigma_0^2) \leftarrow \text{Decrypt}(QK, Z_0, \sigma_0^2)$: a deterministic algorithm run by $C$ or a user in $U$ that takes the query key $QK$, the Kriging results $Z_0$ and $\sigma_0^2$ from the server and outputs the Kriging prediction $[z_0^*]$ and the Kriging variance $[\sigma_0^2]$ at the queried location.

- $\alpha_{r', [z']}$ $\leftarrow \text{AddRequest}(UK, r', [z'])$: a deterministic algorithm run by $C$, a data user in $U$ or a data generating node, which takes the update key $UK$, a location $r'$, a measurement value $[z']$, and outputs an addition token $\alpha_{r', [z']}$. 

- $DB' \leftarrow \text{Add}(DB, \alpha_{r', [z']})$: a deterministic algorithm run by $S$ which takes the current outsourced database $DB$ and an addition token $\alpha_{r', [z']}$, and outputs an updated database $DB'$ representing the Kriging dataset $(P \cup \{r'\}, S \cup \{[z']\})$.

- $\delta_r \leftarrow \text{DeleteRequest}(UK, r)$: a deterministic algorithm run by $C$, a data user in $U$ or a data generating node which takes as input a location $r \in P$ and the update key $UK$ and outputs a deletion token $\delta_r$.

- $DB' \leftarrow \text{Delete}(DB, \delta_r)$: a deterministic algorithm by the server which takes as input the current database $DB$, as well as a deletion token $\delta_r$, and outputs an updated database $DB'$ representing the Kriging dataset $(P \setminus \{r\}, S \setminus \{[z_r]\})$, where $[z_r] \in S$ is the measurement corresponding to location $r \in P$ in $DB$. 

182
6.3 Private Outsourced Kriging Interpolation

Next, we analyse the security requirements of each component within a Kriging system.

1. The measurement values \( z_i \in S \) are highly sensitive and must be protected at all times.

2. We consider the coordinates \( r_i \in P \) of previous measurements to not be sensitive. This is reasonable since in some applications these locations may be externally observable, such as the case where they are the locations of previous mining activity.

3. The queried location \( r_0 \) at which a new prediction should be computed may reveal areas of particular interest to the user. The sensitivity of this relies on the setting and individual user requirements. However, in practice, Kriging queries are often made at every location within a region to produce a heat map of a phenomenon (as considered in our toy example), which may limit the sensitivity of individual query locations. Further, the basic assumption of Kriging is that the quality of predictions degrades with distance. Therefore, the best Kriging results will be obtained when the queried location is broadly within the region of prior (observed) measurements.

4. The computed prediction \( z_0^* \) is highly sensitive as it may form the basis of future decisions and may also be business-critical, and therefore must be protected.

5. The choice of variogram model (without the variogram parameters) may reveal something about the overall trend of the spatial dependencies of the measurements. We assume that this is not particularly sensitive information, and we leave a detailed leakage analysis to future work.

6. The range parameter \( \rho \) of the variogram is a constant scaling of the region \( R \) denoting the inter-measurement distance \( h \) at which the spatial dependency becomes negligible. For distances \( h > \rho \), the variogram approaches the variance of the measurements \( [Wac13] \), which is represented by the sill \( [\nu] \).

7. The nugget effect \( [\eta] \) reveals the spatial dependency at very small distances.

We assume that the range is not sensitive (as it merely scales the region \( R \)), but that information revealed by the nugget and sill may be sensitive. Even in applications where this direct information on the variance and spatial dependency of measurements is deemed non-sensitive, it may be the case that the variogram parameters are commercially sensitive. These parameters must be chosen empirically to best fit the experimental data, a process which may be time-consuming. The quality of predictions depends on how well the variogram matches the experimental variogram. We summarise our security requirements in Table 6.2.
### 6.4 Our Techniques

In this section we introduce the main concepts used in our construction: the canonical variogram. We show how to factor out the variogram parameters in the Normal equations which allows us to remove these parameters from the outsourced dataset, and use them to recover the final prediction on the client side. The crux of our solution for the private outsourcing of Kriging interpolation is to observe how the Kriging prediction varies according to the variogram nugget effect \[ \eta \], the sill \[ \nu \], and range \[ \rho \] in the non-degenerate case. We define a canonical variogram for each variogram model by arbitrarily fixing the parameters \[ \eta = \rho = 1 \] and \[ \nu = 0 \], although our results clearly translate to other choices.

Since the Kriging process is inherently linear, we show how to ‘factor out’ the sensitive parameters \[ \eta \] and \[ \nu \] from the variogram to leave just the canonical variogram. Using this result, in combination with an additively homomorphic scheme, an untrusted server can compute a related Kriging prediction, and variance, without any knowledge of \[ \eta \], \[ \nu \] and the actual measurements. The variogram parameters can then be used by the client locally to compute the final prediction.

#### 6.4.1 The Canonical Normal Equations

**Definition 6.2 (Canonical Variogram)** Let \( \gamma(h) \) be a non-degenerate variogram function with nugget effect \[ \eta \], sill \[ \nu \] and range \[ \rho \]. We define its associated canonical variogram as the function \( \tilde{\gamma} : \mathbb{R}^{\geq 0} \to \mathbb{R} \) satisfying \( \tilde{\gamma}(0) = 0 \) and:

\[
\tilde{\gamma}(h) = -\frac{1}{\nu - \eta} \gamma(\rho h) + \frac{\nu}{\nu - \eta} \quad \text{for } h > 0.
\]

(6.2)

Note that for any non-degenerate variogram function coming from the parametric variogram models defined in Section 6.2.2 the canonical variogram depends only on the model and not...
6.4 Our Techniques

on any of the parameters. Given a Kriging dataset \((P, S)\) of \(n\) measurements, a query position \(r_0 \notin P\) and a variogram function \(\gamma\) with nugget effect \([\eta]\), sill \([\nu]\) and range \(\rho\), let \(Kw = v\) be the corresponding Normal equations as defined in Section 6.2. Our main result is that it is sufficient to consider a canonical version of the Normal equations that depends only on the chosen variogram model, as well as \(P\) and the range parameter \(\rho\) of \(\gamma\).

**Definition 6.3 (Canonical Normal Equations)** We define the canonical Normal equations as the linear system obtained from the Normal equations \(Kw = v\) by replacing:

1. every \(r_i \in P\) by \(r_i/\rho\), i.e. \((x_i, y_i) \mapsto (\frac{x_i}{\rho}, \frac{y_i}{\rho})\),
2. the query position \(r_0\) by \(r_0/\rho\), similarly to above, and
3. the variogram \(\gamma(h)\) by the canonical variogram \(\tilde{\gamma}(h)\).

Note that when the positions are scaled we have
\[
d(r_i/\rho, r_j/\rho) = \frac{1}{\rho}d(r_i, r_j).
\]

We denote the canonical Normal equations as \(\tilde{K}\tilde{w} = \tilde{v}\). The matrix \(\tilde{K}\) is given by:
\[
\left(\begin{array}{cccc}
\tilde{\gamma}(d(r_1/\rho, r_2/\rho)) & \tilde{\gamma}(d(r_1/\rho, r_2/\rho)) & \ldots & \tilde{\gamma}(d(r_1/\rho, r_n/\rho)) \\
\tilde{\gamma}(d(r_2/\rho, r_1/\rho)) & \tilde{\gamma}(d(r_2/\rho, r_2/\rho)) & \ldots & \tilde{\gamma}(d(r_2/\rho, r_n/\rho)) \\
\vdots & \vdots & \ddots & \vdots \\
\tilde{\gamma}(d(r_n/\rho, r_1/\rho)) & \tilde{\gamma}(d(r_n/\rho, r_2/\rho)) & \ldots & \tilde{\gamma}(d(r_n/\rho, r_n/\rho)) \\
1 & 1 & \ldots & 1
\end{array}\right) \in \mathbb{R}^{(n+1) \times (n+1)},
\]
and the canonical Normal equations are given by the system of equations:
\[
\left(\begin{array}{c}
w_1 \\
w_2 \\
\vdots \\
w_n \\
w_{n+1}
\end{array}\right) = \left(\begin{array}{c}
\tilde{\gamma}(d(r_0/\rho, r_0/\rho)) \\
\tilde{\gamma}(d(r_0/\rho, r_1/\rho)) \\
\vdots \\
\tilde{\gamma}(d(r_0/\rho, r_n/\rho)) \\
1
\end{array}\right).
\]

Note that, since the canonical variogram is parameterless, the canonical Normal equations involve only the variogram model and the locations in \(P\) scaled by \(1/\rho\). This observation
allows us to take advantage of the linearity of the Kriging predictor in order to protect the measurements and interpolation value, whilst hiding the sill and nugget parameters $[\nu]$ and $[\eta]$ from the server by storing them locally. We now discuss the solution to the canonical Normal equations in Proposition 6.1.

**Proposition 6.1** Let $K, K' \in \mathbb{R}^{(n+1) \times (n+1)}$ be real matrices, and let $v, v' \in \mathbb{R}^{n+1}$ be real vectors such that:

1. there exists $a, b \in \mathbb{R}$ such that $K'_{i,j} = aK_{i,j} + b$ and $v'_{i} = av_{i} + b$ for all $1 \leq i, j \leq n$,
2. $K_{i,n+1} = K_{n+1,i} = K'_{i,n+1} = K'_{n+1,i} = v_{n+1} = v'_{n+1} = 1$ for all $1 \leq i \leq n$,
3. $K_{n+1,n+1} = K'_{n+1,n+1} = 0$.

Then, if $w \in \mathbb{R}^{n+1}$ satisfies $Kw = v$, the vector $w' \in \mathbb{R}^{n+1}$ defined by:

$$w'_i = w_i \text{ for all } 1 \leq i \leq n, \text{ and }$$

$$w'_{n+1} = aw_{n+1},$$

satisfies $K'w' = v'$.

**Proof.** Note that:

$$(K'w')_{i} = av_{i} + b \sum_{i=1}^{n} w_{i},$$

for $1 \leq i \leq n$, and we have $(K'w')_{n+1} = 1$. Since $\sum_{i=1}^{n} w_{i} = 1$ (by the last equation of the system $Kw = v$), the result follows. □

This result extends an observation in [Cre92], which states that summing a constant to the variogram does not alter the solutions of the Normal equations, and that such a transformation of the variogram may sometimes be necessary in order to obtain a numerically stable Kriging prediction. We apply this proposition to the Normal equations with $a = -1/([\nu] - [\eta])$ and $b = [\nu]/([\nu] - [\eta])$, and consider the canonical Normal equations. By the definitions of the Kriging prediction and the Kriging variance in Section 6.2, we obtain Proposition 6.2.

**Proposition 6.2** Let $[z^*_0]$ and $[\tilde{z}^*_0]$ be the Kriging predictions computed from the Normal and the canonical Normal equations respectively. Denote by $[\sigma^2_0]$ and $\tilde{\sigma}^2_0$ the Kriging variance...
associated with each of the predictors. Then:
\[ [\tilde{z}_0^*] = [z_0^*] \quad \text{and} \quad \tilde{\sigma}_0^{*2} = 1 + \frac{[\nu]}{[\nu] - [\eta]} [\sigma_0^{*2}] + \frac{[\nu]}{[\nu] - [\eta]} \]

In the case that the variogram is non-degenerate, the Kriging prediction is independent of the sill \([\nu]\) and nugget \([\eta]\) parameters of the variogram, whilst the range parameter \([\rho]\) scales positions. When we apply a linear transformation to the variogram, the Kriging variance of the obtained Kriging predictor changes according to the same transformation.

### 6.5 Our Construction

We now outline the operation of each of the algorithms in our scheme. Denote by \(H = (\text{KGen}, \text{Enc}, \text{Dec}, \text{Eval})\) an IND-CPA-secure additive homomorphic encryption scheme. Then:

- \((C, UK, QK) \overset{\text{Outsource}}{\leftarrow} (1^\lambda, P, S, \gamma)\): If \(\gamma\) is a degenerate variogram function, halt and return \(\bot\); in this case, our protocol fails. Otherwise, generate a keypair for the homomorphic encryption scheme:
  \[(pk, sk) \leftarrow \text{KGen}(1^\lambda).\]

Recall that: \(P \subseteq \mathbb{R}^2\) is the ordered set of locations \((r_i)_{i=1}^n\), \(S \subseteq \mathbb{R}\) is the ordered set of measurements \(([z_i])_{i=1}^n\), and that the variogram \(\gamma\) is made up of three parameters: the nugget \([\eta]\), the sill \([\nu]\) and the range \(\rho\). Let \(\tilde{\gamma}\) be the canonical variogram associated with \(\gamma\), as defined in Section 6.4. Define the update key \(UK\) and the query key \(QK\) as:
\[ UK = (pk, \rho) \quad \text{and} \quad QK = (sk, [\eta], [\nu], \rho).\]

To account for the factor of \(\rho\) in the input to \(\gamma\) in Equation 6.2, compute the scaled locations:
\[ \tilde{P} = ((x_i/\rho, y_i/\rho))_{i=1}^n.\]

Finally, encrypt each measurement in \(S\) and define the ordered set:
\[ Z = (\text{Enc}(pk, [z_i]))_{i=1}^n.\]

Output \(C = (\tilde{P}, Z, \tilde{\gamma})\), along with \(UK\) and \(QK\).

\(^4\)However, if \(\gamma\) is degenerate, the variogram is constant (the so-called ‘nugget effect model’) and models a purely random variable with no spatial correlation. In this case the prediction is \([z_0^*] = \sum z_i/n\) for \(r_0 \not\in P\) and the variance is \([\sigma_0^{*2}] = n + 1\).
6.5 Our Construction

- DB ← Setup(C): Instantiate the matrix \( \tilde{K} \) from the canonical Normal equations using the positions \( r'_i \in \tilde{P}, 1 \leq i \leq n \), and the canonical variogram \( \tilde{\gamma} \). Set
  1. \( \tilde{K}_{i,j} = \tilde{\gamma}(d(r'_i, r'_j)) \) for \( 1 \leq i, j \leq n \),
  2. \( \tilde{K}_{n+1,i} = \tilde{K}_{i,n+1} = 1 \) for \( i \neq n + 1 \), and
  3. \( \tilde{K}_{n+1,n+1} = 0 \).

Return DB = (\( \tilde{K}, C \)).

- \( Q \leftarrow \text{Query}(r_0, QK) \): Let \( r_0 = (x_0, y_0) \) and, recalling that \( \rho \) is contained within \( QK \), return \( Q = (x_0/\rho, y_0/\rho) \).

- \( (\tilde{Z}_0, \tilde{\sigma}_0^*2) \leftarrow \text{Interpolate}(Q, DB) \): Recall that \( C = (\tilde{P}, Z, \tilde{\gamma}) \). If \( Q \in \tilde{P} \), then the exact measurement is contained in the outsourced dataset and no prediction is required. Let \( j \) be the index such that \( Q = r_j \), and return \((Z_j, \perp)\), where \( \perp \) is a distinguished symbol denoting that the prediction is exact.

Otherwise, compute the vector \( v \) from the canonical Normal equations using the locations \( r'_i \in \tilde{P} \), the query position \( Q \) and the canonical variogram \( \tilde{\gamma} \):
  1. \( v_i = \tilde{\gamma}(d(Q, r'_i)) \) for \( 1 \leq i \leq n \), and
  2. \( v_{n+1} = 1 \).

Compute the solution \( \tilde{w} \) to the canonical Normal equation \( \tilde{K}\tilde{w} = \tilde{v} \) in plaintext. This step essentially computes the Kriging coefficients \( w \) using the canonical variogram and the scaled locations without requiring the parameters of the variogram. Then, using the homomorphic property of the encryption, compute the encrypted prediction \( \tilde{Z}_0 \):

\[
\tilde{Z}_0 = \sum_{i=1}^{n} \tilde{w}_i \cdot Z_i \quad \text{and} \quad \tilde{\sigma}_0^*2 = \tilde{w}_{n+1} + \sum_{i=1}^{n} \tilde{w}_i \cdot \tilde{\gamma}(Q, r'_i).
\]

Return the encrypted prediction \( \tilde{Z}_0 \), and the partially computed Kriging variance (error estimation) \( \tilde{\sigma}_0^*2 \).

- \( ([z_0^*], [\sigma_0^*2]) \leftarrow \text{Decrypt}(\tilde{Z}_0, \tilde{\sigma}_0^*2, QK) \): First decrypt the Kriging prediction:
  \[
  [\tilde{z}_0^*] = \text{Dec}(sk, \tilde{Z}_0),
  \]

where \( sk \) is contained within \( QK \). Then, if \( \tilde{\sigma}_0^*2 = \perp \), set \( [\sigma_0^*2] = 0 \). Else, compute the Kriging variance:

\[
[\sigma_0^*2] = [\nu] - ([\nu] - [\eta])\tilde{\sigma}_0^*2.
\]

188
This final step essentially adds back in the parameters of the variogram, which were removed for outsourcing, using the result from Proposition 6.2.

- $\alpha_{\hat{r}',[z']}$ $\leftarrow$ AddRequest($r',[z'],UK$): Let $r_a = \frac{r'}{\rho}$ and compute the ciphertext:

$$Z_a = \text{Enc}(pk,[z'])$$

where $\rho$ and $pk$ are contained within $UK$. Output the addition token:

$$\alpha_{\hat{r}',[z']} = (r_a, Z_a)$$

- $DB' \leftarrow \text{Add}(DB, \alpha_{\hat{r}',[z']})$: Recall that $\alpha_{\hat{r}',[z']} = (r_a, Z_a)$. Compute the updated dataset:

  if $r_a \in \tilde{P}$ then let $j$ be the index such that $r_j = r_a$ and modify $Z_j \in Z$ to be $Z_a$. Otherwise, set $C' = (\tilde{P} \cup \{r_a\}, Z \cup \{Z_a\}, \tilde{\gamma})$. Return the output of $\text{Setup}(C')$.

- $\delta_r \leftarrow \text{DeleteRequest}(r,UK)$: Return $\delta_r = r/\rho$.

- $DB' \leftarrow \text{Delete}(DB, \delta_r)$: If $\delta_r \notin \tilde{P}$, return $DB$ as there is nothing to remove. Otherwise, let $j$ be the index such that $r = r_j$ in $\tilde{P}$. Compute the updated dataset $C' = (\tilde{P} \setminus \{r_j\}, Z \setminus \{Z_j\}, \tilde{\gamma})$ and return the output of $\text{Setup}(C')$.

### 6.6 Discussion

The correctness of the scheme is immediate from Proposition 6.2 as well as the correctness of the homomorphic encryption scheme $\mathcal{H}$. These homomorphic properties enable addition and scalar multiplication of ciphertexts, whilst ensuring that the results decrypt appropriately. Proposition 6.2 shows that the Kriging prediction, as well as the Kriging variance, can be computed by applying a linear transformation to the result computed using the canonical (parameterless) variogram. Correctness of the updates is apparent because the addition and deletion tokens format the data in the same way as the original dataset. Since we are using the honest-but-curious model, the server will modify the dataset correctly. The remainder of the update algorithms then simulate a new setup procedure running $\text{Setup}$ on a new Kriging dataset from $\text{Outsource}$.

In terms of security, measurement values are always in encrypted form whilst outsourced. Leakage is bounded by the variogram model as well as both the queried and observed locations (scaled by the inverse of range parameter $\rho$). Therefore, assuming no collusion...
between the server and users, the data is confidential from the server. The homomorphic encryption scheme enables the computation to be performed on the measurements whilst they are encrypted, and therefore at no point during the computation is the data revealed to the server.

It is also clear that neither the variogram parameters \( \eta \) and \( \nu \), nor any values computed from them, are ever revealed to the server. The final parameter of the variogram, the range \( \rho \), is never explicitly given to the server. However, the server does learn the coordinates (scaled by \( \rho \)) of the measurements. Hence, the range could be revealed if the server has existing knowledge of the measurement locations. Of the three variogram parameters, we believe that the range is the least sensitive: it reveals how quickly the variogram approaches the sill (i.e. the distance at which the spatial correlation between measurements becomes negligible) but does not reveal anything relating to the measurement values themselves.

Whilst the queried location is revealed in the plain to the server, we note that the mechanism of Tugrul and Polat [TP13] may easily be used to gain a weak form of secrecy: during the Query algorithm, the party carrying out the query may choose \((q - 1)\) additional locations from the region, and scale each by \( \rho \). The query token then is made up of \( q \) scaled locations, randomly permuted. The server must perform \textit{Interpolate} for each location, and the client discards all results except the one that it is interested in. Unlike [TP13], we do not require an oblivious transfer protocol, since the querier is authorised to learn as many queries on the dataset as it wishes. However, as in [TP13], the server may guess the location of interest with probability \( 1/q \) (but cannot learn the prediction at this location). Data generating nodes cannot learn Kriging predictions as they do not have the decryption key and \( H \) is assumed to be IND-CPA secure.

### 6.6.1 Implementation

We have used the PHE library [PHE16] to implement our construction, and consider the Paillier encryption scheme [Pai99]. We note that the plaintext space of the Paillier encryption scheme is \( Z_n \) for some \( n \). In order to encode the values \([z_i] \in \mathbb{R}\) into this plaintext space, we can use the \texttt{phe.encoding} function in PHE. This function takes as input a number \( x \in \mathbb{R} \), multiplies by a large constant, usually \( 10^k \) for some value of \( k \), and rounds the result to recover \([z10^k] \in \mathbb{Z}\). The implementation is intended as a proof of concept to evaluate the efficiency of the proposed solution. All code is executed locally on an Amazon EC2 instance.
with a 2.5GHz Intel Xeon processor and 1GB memory running Ubuntu 14.04.4. All timings are averaged over 30 iterations, each on a new randomly generated dataset.

![Average Timings](image_url)

Figure 6.5: Graphs showing the timing costs of each algorithm.

Figures 6.5 and 6.6 give some basic timing results for the implementation our construction. Figure 6.5 shows the algorithm costs. Note that the cost of the `Outsource` algorithm dominates all others, and this is due to the cost of $n$ encryptions.

Therefore, for clarity, we also present Figure 6.6 which shows the same results with the exclusion of the `Outsource` algorithm. It can be seen that, with the exception of the (high) one-time cost of `Outsource` (which may be amortised over many queries), the remaining client-side processes are very efficient. The server must perform quadratic work to perform `Setup`, but this will be required only during initial setup and when the outsourced dataset is updated. The online workload of the client is very low, whilst the server’s online workload is linear in the size of the dataset. Further, we note that the server’s online workload greater than the client’s workload, which makes outsourcing worthwhile.
We believe that these experiments demonstrate the performance and scalability of our solution. We have considered the well-known Meuse dataset [BMML15], which contains 155 measurements, and we believe this to be reasonable compared to what is used in practice.

### 6.7 Conclusion

The Kriging interpolation technique describes the best unbiased linear prediction of an observed phenomenon in a geographical region, based on a set of measurements, and it is used in a wide range of applications. In this chapter, we have presented a construction that allows for Kriging interpolation to be securely outsourced to a cloud service provider, such that the measurement values and sensitive variogram parameters are withheld from the server. This solution allows the Kriging interpolation technique to be performed in a privacy-preserving manner, under the assumption that the cloud service provider acts in an honest-but-curious
manner. We have implemented this solution in the PHE library, making use of the Paillier encryption scheme, and have provided some timing results for our implementation.
Chapter 7

Conclusion and Future Work

Contents

7.1 Conclusion ................................................. 194
  7.1.1 Security ............................................. 194
  7.1.2 Standardisation ................................. 195
  7.1.3 Applications ...................................... 196
7.2 Future Work ............................................. 196
  7.2.1 Security ............................................. 196
  7.2.2 Standardisation ................................. 197
  7.2.3 Applications ...................................... 197

7.1 Conclusion

In this thesis we have considered a variety of topics within lattice-based cryptography and homomorphic encryption. Our work has focused around three areas: security, standardisation, and applications. We have studied the security of lattice-based cryptosystems in Chapters 3, 4, and 5, standardisation efforts in Chapters 4 and 5, and applications in Chapter 6.

7.1.1 Security

Understanding the security of the Learning with Errors problem is of central importance to the future of lattice-based cryptography. In this thesis, we have considered variants of the uSVP, dual, decoding, hybrid-dual, and hybrid-decoding attacks which can be used to determine the security of a given set of LWE parameters.
7.1 Conclusion

Specifically, in Chapter 3 we introduced a *guess-and-verify decoding* attack (g-v decoding) on small-secret LWE. This approach follows the approach of the hybrid-decoding attack, but considers a more expensive, higher probability, BDD solver. This alters the landscape of trade-offs and we show that, when the BKZ simulator is assumed, as well as an enumeration cost model for BKZ, our g-v decoding technique outperforms a (non-mitm) variant of the hybrid-decoding attack.

In Chapters 4 and 5 we have considered the security of a variety of different LWE-based parameter sets currently involved in standardisation processes. As part of this work, we have also contributed to the LWE Estimator, to ensure that current state-of-the-art attacks are switched on for binary secrets, and have also released custom code\(^1\).

7.1.2 Standardisation

Standardisation efforts for public-key encryption schemes, digital signature algorithms, and homomorphic encryption schemes based on the Learning with Errors problem are well underway.

The NIST standardisation process, discussed in Chapter 4, aims to standardise a suite of public-key encryption and digital signature algorithms designed for use in an era where quantum computers exist. In this thesis, we have analysed the security of all of the first round submissions against the uSVP and dual attacks (where appropriate). This resulted in a large set of security estimates, which allows for any two schemes to be compared in a fair manner. Moreover, our work highlighted that cost models for BKZ are not *order preserving*. We also provided an update regarding the current state of the NIST standardisation process, including some security estimates for the schemes in the third round.

The [homomorphicencryption.org] standardisation effort, discussed in Chapter 5, aims to standardise LWE-based parameter sets to be used in homomorphic encryption schemes\(^2\). In our work we have considered potential extensions to the latest variant of the Homomorphic Encryption Security Standard. Specifically, we considered the feasibility and impact of standardising LWE-based parameter sets with a *sparse secret distribution*. We present a variety of parameter sets which balance security requirements with the cost of the expensive

---

\(^1\)Available at [https://github.com/bencrts/hybrid_attacks](https://github.com/bencrts/hybrid_attacks) and [https://github.com/estimate-all-the-lwe-ntru-schemes/estimate-all-the-lwe-ntru-schemes.github.io](https://github.com/estimate-all-the-lwe-ntru-schemes/estimate-all-the-lwe-ntru-schemes.github.io)

\(^2\)The consortium is also in the process of standardising an API for use in homomorphic encryption schemes.
7.2 Future Work

Applications of homomorphic encryption schemes are plentiful and ever-increasing. In Chapter 6, we have considered the outsourcing of an interpolation algorithm called Kriging to an honest-but-curious cloud server. In our work, we showed how to “factor out” sensitive parameters in order to allow the system of equations used to determine the weights to be solved by the server in a secure manner. When used in combination with an additively homomorphic encryption scheme, this allows for the weights to be applied to the encrypted measurement values, and for the sum to be computed securely. Finally, the sensitive parameters are re-applied on the client-side, to allow for the interpolation value to be determined.

7.2 Future Work

In this section we conclude this thesis with some comments on potential future work.

7.2.1 Security

Cryptanalysis of lattice-based cryptography is a fast-moving field. Further work is required to accurately estimate the cost of the hybrid-decoding attack, and this includes verification of the various heuristics considered in the hybrid-decoding attack. Future areas to explore also includes considering the impact of quantum enumeration algorithms on the concrete running time of the hybrid-decoding attack, and studying sieving-based BDD solvers in more detail. Moreover, an accurate analysis of the meet-in-the-middle probability for the g-v decoding approach would allow for a direct comparison with the hybrid-decoding attack. Further work is also needed to fully understand the effects of using a sparse secret on the concrete security of LWE.
7.2 Future Work

7.2.2 Standardisation

As the NIST standardisation process continues into the final rounds, it is important for the literature to attempt to converge onto BKZ cost models for both the enumeration and sieving cases. Once this has been achieved, estimating attack complexities becomes a more simple process. Moreover, it is also important for the memory cost of cryptanalytic attacks to be considered, as the majority of cryptanalytic arguments in the submissions to the NIST standardisation process focus solely on the time cost of attacks. As an example, we have seen that hybrid-decoding attacks, as outlined in Chapter 3, can outperform the dual and uSVP attacks. However, these attacks require exponential memory.

In terms of homomorphic encryption standardisation, the methodology in the current variant of the HE Standard relies on the LWE Estimator. However we note that the LWE Estimator has a number of limitations, including outdated decoding estimates and lack of support for hybrid attacks. Moreover, the attack landscape for sparse-secret LWE is fast moving, with several new contributions emerging in recent years. Further work is needed to ensure that any parameter sets standardised for homomorphic encryption schemes are analysed under hybrid attacks.

7.2.3 Applications

Our proposed construction for the private outsourcing of Kriging interpolation may be extended in several ways. For example, we could consider extending our protocol to protect the locations of the query points. This can be achieved by increasing interactivity, communication complexity, and client computation in the query process. However, this approach requires the server to compute square roots and natural exponentials over encrypted data (in an efficient manner) which remains an open problem.
Bibliography


BIBLIOGRAPHY


[BAA+19] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Kramer, Patrick Longa,
BIBLIOGRAPHY


[Ber18] Daniel J. Bernstein, 2018. Comment on PQC forum in response to an earlier version of this work. Available at https://groups.google.com/a/list.nist.gov/d/msg/pqc-forum/h4_LCVNejCI/FyV5hgnqBAAJ.


BIBLIOGRAPHY


BIBLIOGRAPHY


Microsoft Research. SEAL. [https://github.com/Microsoft/SEAL](https://github.com/Microsoft/SEAL) 2020. Github repository, commit c9af10d.


Dustin Moody. NIST announcement of 2nd round candidates. Available at [https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/bBxcfFFUsxE](https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/bBxcfFFUsxE) 2019.

Dustin Moody. NIST announcement of 3rd round candidates. Available at [https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/0ieuPB-bSeg](https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/0ieuPB-bSeg) 2020.

BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


[PFH+19] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim
Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William
Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute

[PFH+20] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim
Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William
Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute
projects/post-quantum-cryptography/round-3-submissions.

[PHAM17] Le Trieu Phong, Takuya Hayashi, Yoshinori Aono, and Shiho Moriai.
LOTUS. Technical report, National Institute of Standards and Technology,
2017. available at https://csrc.nist.gov/projects/post-quantum-
cryptography/round-1-submissions.

[PHE16] python-paillier: a library for partially homomorphic encryption in

Royal Holloway University of London, 2019.

[PS16] Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin
Hirt and Adam D. Smith, editors, TCC 2016-B: 14th Theory of Cryptography
Conference, Part II, volume 9986 of Lecture Notes in Computer Science,
pages 217–238, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg,
Germany.


homomorphisms. In Foundations on Secure Computation, Academia Press,


