Estimation of the Hardness of the Learning with Errors Problem with a Given Number of Samples

Abschätzung der Schwierigkeit des Learning with Errors Problem mit gegebener fester Anzahl von Samples

Master-Thesis von Markus Schmidt

Tag der Einreichung:

1. Gutachten: Prof. Dr. Johannes Buchmann
2. Gutachten: Nina Bindel

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Computer Science
Theoretical Computer Science -
Cryptography and Computer Algebra

Estimation of the Hardness of the Learning with Errors Problem with a Given Number of Samples
Abschätzung der Schwierigkeit des Learning with Errors Problem mit gegebener fester Anzahl von Samples

Vorgelegte Master-Thesis von Markus Schmidt

1. Gutachten: Prof. Dr. Johannes Buchmann
2. Gutachten: Nina Bindel

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 22. Februar 2016

(M. Schmidt)

Abstract

Lattice-based cryptography is a promising candidate to build cryptographic primitives that are secure even against quantum algorithms. The Learning with Errors problem is one of the most important hardness assumptions, lattice-based construction base their security on. Recently, Albrecht et al. (Journal of Mathematical Cryptology 2015) presented the Sage module "LWE-Estimator" to estimate the hardness of LWE instances, making the choice of parameters for lattice-based primitives easier and better comparable. The effectiveness of algorithms to solve LWE is often depending on the number of LWE instances, called LWE-samples, given. Therefore, the optimal number of LWE-samples is assumed to estimate the hardness. In cryptographic applications the optimal number of samples is often not given, but only a small number of samples. This leads to a more conservative choice of parameters than necessary in applications.

This work aims to improve the parameter choice with respect to described problem. The contribution presented in this work is twofold. First, we analyze the hardness of LWE instances given a fixed number of samples. For this, we describe algorithms proposed in literature to solve LWE shortly and estimate their computational cost while taking a limitation of the available number of samples into account. We consider instances of generic LWE as well as instances with small secret. Secondly, we use these results to extend the Sage module "LWE-Estimator", so that the resulting implementation can be used to estimate LWE instances with fixed numbers of samples. Furthermore, we present examples of using the implementation and show estimation results using example parameters. These indicate a significant impact on the hardness of LWE if the number of samples is strongly limited. Also, we show a comparison of the considered algorithms, focusing on the behavior when limiting the number of available samples.

Contents

List of Tables

List of Figures

1 Introduction

The Learning with Errors (LWE) problem is used in the construction of many cryptographic primitives [22, 37, 38]. It became popular due to its flexibility for instantiating cryptographic solutions and comes with the advantage that it can be instantiated such that it is provably as hard as worst-case lattice problems [38]. Moreover, LWE presumably remains hard even when quantum algorithms are considered. In general, an instance of LWE is characterized by parameters $n \in \mathbb{Z}$, $\alpha \in (0, 1)$ and $q \in \mathbb{Z}$. To solve the LWE problem, an algorithm has to be able to recover the secret vector $\mathbf{s} \in \mathbb{Z}_q^n$, given access to LWE-samples $(\mathbf{a}_i, c_i = \mathbf{a}_i \cdot \mathbf{s} + e_i \mod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ chosen according to a probability distribution characterized by α.

To estimate the hardness of concrete instances of LWE, the Sage module "LWE-Estimator" by Albrecht et al. [5, 6] can be used. In particular, this is useful for choosing and comparing parameters for lattice based primitives. Since the algorithms solving LWE often depend on the number of LWE-samples, for the "LWE-Estimator" the optimal number of samples is assumed to estimate the hardness. In contrary to this, the optimal number of samples is often not given in cryptographic applications. In such cases using the "LWE-Estimator" leads to overly conservative estimations when using the "LWE-Estimator" for cryptographic applications with limited numbers of samples. Therefore, the parameters chosen to make the system secure may be too conservative when relying on the estimations of the optimal samples case. These results can be improved by taking a possible limitation of the number of samples into account. This leads to a more precise estimation for LWE-based cryptographic systems with limited numbers of samples and hence less conservative parameter choices.

Albrecht et al. [6] give a survey of the concrete hardness of LWE based on existing algorithms solving LWE. This means, given particular values (n, α, q) for an instance of LWE, the computational cost of solving LWE using currently known algorithms is presented. These algorithms can be categorized by the strategy they employ to solve LWE into three families. One approach reduces LWE to finding a short vector in the dual lattice formed by the given samples, also known as Short Integer Solution (SIS) problem. Another strategy solves LWE by considering it as a Bounded Distance Decoding (BDD) problem, for which a lattice point in the lattice built by the samples is considered, where the error of the samples form a noise vector bounded in distance from this lattice point. The third family consists of combinatorial algorithms like the Blum-Kalai-Wasserman (BKW) algorithm. The algorithm proposed by Arora and Ge [8] is somewhat different to the others, since it solves LWE using a system of noise-free non-linear polynomials with the root being the secret of the LWE instance. Due to its high costs and consequential insignificant practical use, this algorithm is not considered throughout this work.

The following algorithms are considered to estimate the concrete hardness of LWE: exhaustive search, BKW, using lattice reduction to distinguish, decoding, standard embedding and dual embedding. Figure 1 shows the categorization by strategies used to solve LWE and by employment of lattice reduction. BKW is classified as solving via SIS strategy, since it can be seen as an oracle producing short vectors in the dual lattice constructed by the samples $\mathbf{a}_i$. The "direct" strategy implies, that the algorithms using this method solve for the secret directly.

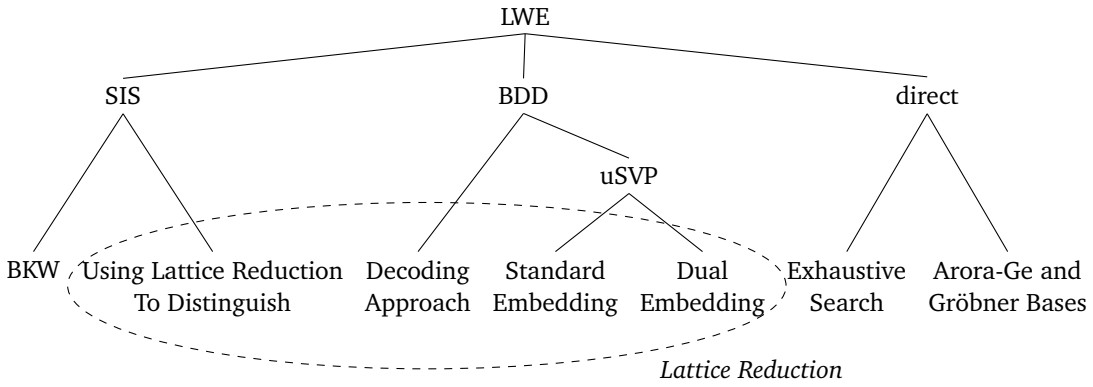


Figure 1: Overview of existing algorithms to solve LWE categorized by different strategies described in Sections 2.3 and 2.4; algorithms using lattice reduction methodes are dashed-framed; the following abbreviations are used: LWE – learning with errors problem, BDD – bounded distance decoding problem, SIS – short integer solution problem, uSVP– unique shortest vector problem and BKW– Blum-Kalai-Wassermann algorithm

1.1 Related Work

The paper presented by Albrecht et al. [6] gives a survey of estimations of the hardness of concrete LWE instances and is a key paper for this work. These estimations are based on state-of-the-art algorithms both in standard and small secret variants and assume that the optimal number of samples is available. Additionally, the authors describe their Sage module called "LWE-Estimator", which calculates hardness estimations for concrete instances of LWE, and provide examples of usages. A general description of lattice-based cryptography, its relevant problems and main cryptographic primitives is given by Micciancio and Regev in [35]. As part of this, also lattice reduction is broached, while a more detailed discussion is shown by Gama and Nguyen in [20]. They assess the practical hardness of the main lattice problems based on experiments using the NTL library. Linder and Peikert [31] introduce a new scheme being a generalization of prior LWE-based cryptographic systems, which achieves smaller keys and ciphertexts. Along with that, they present a new decoding approach for attacking LWE based on Babai's Nearest Plain attack introduced in [9]. Furthermore, Lindner and Peikert introduce a new model of estimating BKZ runtimes based on extrapolating from experiments using small parameters. Blum, Kalai and Wasserman [12] present the BKW algorithm with estimations of the hardness being discussed in detail by Albrecht et al. in [2]. This is further improved by Duc et al. [19] using Fourier transform. Also, they apply the same technique to the Learning with Rounding (LWR) problem, which is not in the scope of this work. Another improvement of BKW, called Coded-BKW, is introduced by Guo, Johansson and Stankovski in [24], where the authors utilize a lattice code to map subvectors to codewords. This leads to increased noise, but produces better overall results. Additionally, they analyze the complexity of Coded-BKW in the small secret case. One of the the small secret variants of BKW employs lazy modulus switching as described by Albrecht et al. [3]. Albrecht, Fitzpatrick and Göpfert [4] analyze the concrete hardness of instances of LWE under an attack known as Kannan's attack or standard embedding, which reduces the Bounded Distance Decoding (BDD) problem to the unique-Shortest-Vector-Problem (uSVP) and then solves this via embedding. To solve small secret instances in this manner, Bai and Galbraith [11] present their embedding approach for these instances, which is somewhat similar to standard embedding, but utilizes a dual lattice. Another embedding approach is the dual-embedding attack, which is introduced by Bai and Galbraith [10]. Dagdelen et al. [17] give an improvement of the estimations regarding this attack.

1.2 Contribution

Estimations of the "LWE-Estimator" by Albrecht et al. [6] on the hardness of LWE assume that the optimal number of samples is accessible. Often, this property is not given in cryptographic applications, allowing for less restricting parameter sets than proposed by the "LWE-Estimator". We aim to solve this problem. As first part, we analyze the hardness of LWE instances while taking a fixed number of samples into account. Our analysis is based on several currently known algorithms. Except for Arora and Ge's algorithm, these are the algorithms mentioned in Figure 1. We describe each of them shortly and then analyze them regarding their computational costs when solving LWE while taking a fixed number of samples into account. Additionally, we analyze the small secret variants of these algorithms, where the components of the secret vector are chosen from a pre-defined set of small numbers.

As discussed by Albrecht et al. [6], there are mostly no sufficiently precise closed formulas for calculating the runtimes only depending on parameters of a given LWE instance (n, α, q), since there is no function of δ_0 to calculate the runtime of lattice reduction. To compensate this, they introduce the "LWE-Estimator" as a Sage implementation, which provides estimations of the computational costs given concrete instances of LWE. In the second part of this work, we provide an implementation of the results of the analyses of the first part based on the "LWE-Estimator". We always use the existing estimations, assuming the optimal number of samples is available, if the given, fixed number of samples exceeds the optimal number. If not enough samples are given, we calculate the computational costs using the estimations presented in this work. The implementation will be publicly available at `https://www.cdc.informatik.tu-darmstadt.de/cdc/personen/nina-bindel/` and is expected to be integrated into the existing LWE-Estimator at `https://bitbucket.org/malb/lwe-estimator` in due time.

Also, we present examples of the usage and the output of the implementation, which give an exemplary evaluation of the results. These show, that the hardness of most of the considered algorithms are influenced significantly by the limitation of the available number of samples. Furthermore, we describe the structure of the code and highlight some implementation details. At the end, we show a comparison of the behavior of the hardness of the considered algorithms with focus on the limitation of samples.

1.3 Structure

In Section 2 we introduce notations along with some definitions of important hardness assumptions like the LWE problem and structures required to understand the subsequent sections. We describe lattice reduction and its runtime estimations in Section 3. The Section 4 shows our analyses of the considered algorithms. For each of them, we present a description of the attack itself at first and then analyze the standard instances of LWE with parameters (n, α, q) and the corresponding small secret variants while taking a limitation of the numbers of samples into account. In Section 4.7 we show the small

secret variant by Bai and Galbraith using an embedding approach in the same manner. In Section 5 we give an example of using the implementation together with an explanation of the possible outputs, a description of the structure of the code and a comparison of the hardness of the considered algorithms in terms of behavior when limiting the number of available samples. Additionally, we demonstrate the results of the presented estimates using example parameters and finish with a summary in Section 6.

2 Notations and Definitions

The notations in this work are inspired by Albrecht et al. [6]. That is, logarithms are base 2 except indicated otherwise. Column vectors are denoted as lowercase bold letters and matrices by uppercase bold letters. Let $\mathbf{a}$ be a vector, then $\mathbf{a}_{(i)}$ denotes the i-th component of $\mathbf{a}$ and $\mathbf{a}_i$ denotes the i-th vector of a list of vectors. Furthermore, let $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n$ and $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{R}^n$ be two vectors. Then $\mathbf{a} || \mathbf{b} = (a_1, \ldots, a_n, b_1, \ldots, b_n) \in \mathbb{R}^{2n}$ is the standard concatenation of two vectors and $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$ is the usual dot product. We use $\approx$ to indicate that something is sufficiently close to equal and therefore treated as equal for our estimates. We use the standard definition of the Landau notation as shown in Table 1.

Notation	Definition
$f \in o(g)$	f asymptotically negligible compared to g
$f \in \mathcal{O}(g)$	g asymptotic upper bound for f
$f \in \Omega(g)$	g asymptotic lower bound for f
$f \in \omega(g)$	f asymptotically dominant compared to g $(g \in o(f))$

Table 1: Definition of the Landau notation as used in this work

2.1 Lattice

A lattice L is defined as a discrete additive subgroup of $\mathbb{R}^m$ containing all integer linear combinations of n linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$, which together form a non-unique basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ of L. Consider $\mathbf{B}$ as a matrix, where the columns are formed by the basis vectors. Then the lattice can be expressed as $L(\mathbf{B}) = \{\mathbf{Bu} \mid \mathbf{u} \in \mathbb{Z}_q^n\}$. The rank of L is the rank of the basis matrix $\mathbf{B}$ and L is called full rank, if $\text{rank}(L) = m$. The determinant of the lattice L is defined as the absolute value of the determinant of the basis matrix $\det(L(\mathbf{B})) = |\det(\mathbf{B})|$. The determinant is independent of the actual choice of the basis. Using this, the volume $\text{vol}(L)$ of a full-rank lattice L can be defined as the absolute value of the determinant. A q-ary lattice is a lattice L, that satisfies $q\mathbb{Z}^m \subseteq L \subseteq \mathbb{Z}^m$. In the following, we consider only full-rank, q-ary lattices.

The scaled (by q) dual lattice $L^\perp$ of the lattice L, generated by $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, contains all vectors orthogonal to the columns of $\mathbf{A}$. In the following we define lattices.

Definition 1 (Lattice)**.** Let $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$ be n linearly independent vectors forming a basis and let $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ be a matrix containing the basis vectors as columns. Then, the q-ary lattice $L(\mathbf{A})$ and the corresponding dual lattice $L^\perp(\mathbf{A})$ are defined by

$$L(\mathbf{A}) = \{\mathbf{y} \in \mathbb{Z}^m \mid \exists \mathbf{s} \in \mathbb{Z}^n : \mathbf{y} = \mathbf{As} \mod q\} \ , \tag{1}$$

$$L^\perp(\mathbf{A}) = \{\mathbf{y} \in \mathbb{Z}^m \mid \mathbf{y}^T\mathbf{A} = \mathbf{0} \mod q\} \ . \tag{2}$$

In this work, the distance between a lattice L and a vector $\mathbf{v} \in \mathbb{R}^m$ is defined as the distance between $\mathbf{v}$ and the closest lattice point $\mathbf{x} \in L$ to $\mathbf{v}$. Written formally, it is $\text{dist}(\mathbf{v}, L) = \min\{\|\mathbf{v} - \mathbf{x}\| \mid \mathbf{x} \in L\}$.

Some definitions about lattices will be needed later and are given in the following. The i-th successive minimum $\lambda_i(L)$ of the lattice L is defined as the smallest radius r, such that there are i linearly independent vectors of norm r in the lattice. Usually, the norm is the euclidean norm and therefore, it can be seen as the radius of the smallest ball around the origin, which contains i linearly independent lattice vectors. Based on this definition it is possible to state the Gaussian heuristic $\lambda_i(L) \approx \sqrt{\frac{m}{2\pi e}}\,\text{vol}(L)^{\frac{1}{m}}$.

Lattice reduction is a method to find a new basis of a given lattice, such that the basis vectors are short and nearly orthogonal to each other. Since lattice reduction is used by many of the algorithms described below, it is necessary to introduce the Hermite factor δ_0^m, which is defined by $\delta_0^m = \frac{\|\mathbf{v}\|}{\text{vol}(L)^{\frac{1}{m}}}$, where $\mathbf{v}$ is the shortest non-zero vector in the basis returned by the lattice reduction algorithm. The Hermite factor describes the quality of a basis, which, for example, may be the output of a lattice reduction algorithm. Even though, strictly speaking δ_0 is called root-Hermite factor and $\log \delta_0$ the log root-Hermite factor, "Hermite-factor" also often refers to δ_0.

There exist several problems related to lattices, some of which we describe in the following.

Definition 2 (SVP)**.** Given a lattice $L(\mathbf{A})$, the Shortest Vector Problem (SVP) is the problem to find the shortest non-zero vector in $L(\mathbf{A})$.

Variants of the SVP are the γ-unique Shortest Vector Problem (γ-uSVP) and the κ-Hermite Shortest Vector Problem (κ-HSVP), which we define in the following:

Definition 3 (γ-uSVP)**.** Let $L(\mathbf{A})$ be a lattice such that $\lambda_2(L(\mathbf{A})) > \gamma\lambda_1(L(\mathbf{A}))$, the γ-unique SVP is the problem to find the shortest non-zero vector $\mathbf{v} \in L(\mathbf{A})$.

Definition 4 (κ-HSVP)**.** Let $L(\mathbf{A})$ be a lattice, the κ-Hermite SVP is the problem of finding a vector $\mathbf{v} \in L$ such that $0 < \|\mathbf{v}\| \leq \kappa \cdot \det(L)^{\frac{1}{n}}$

Furthermore, we define the Closest Vector Problem (CVP):

Definition 5 (CVP)**.** Given a lattice $L(\mathbf{A})$ and a target vector $\mathbf{t} \in \mathbb{R}^m$, which is not necessarily in the lattice, finding the lattice vector $\mathbf{v} \in L(\mathbf{A})$ closest to $\mathbf{t}$ solves the CVP.

2.2 Learning with Errors Problem

The Learning with Errors problem (LWE) is a generalization of the parity learning problem [12], introduced by Regev [38]. We first recall the definition of the Gap Shortest Vector Problem (GapSVP).

Definition 6 (GapSVP [13])**.** For an approximation ratio $\gamma \geq 1$, the GapSVP$_\gamma$ is the problem of deciding, given a basis $\mathbf{B}$ of an n-dimensional lattice $L(\mathbf{B})$ and a number d, between the case where $\lambda_1(L(\mathbf{B})) \leq d$ and the case where $\lambda_1(L(\mathbf{B})) > \gamma d$.

Regev [38], Peikert [37] and Brakerski et al. [13] show reductions from the worst-case hardness of the GapSVP problem to LWE. In the following we recall the definition of LWE.

Definition 7 (LWE [6])**.** Let n and q be positive integers. Additionally, let χ be a probability distribution over $\mathbb{Z}$ and let $\mathbf{s} \in \mathbb{Z}_q^n$. Then, $L_{\mathbf{s},\chi}$ denotes the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \in \mathbb{Z}$ according to χ (and considering it in $\mathbb{Z}_q$), and returning $(\mathbf{a}, c = \mathbf{a} \cdot \mathbf{s} + e \mod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.
Decision-LWE is the problem of deciding whether pairs $(\mathbf{a}, c) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ are sampled according to $L_{s,\chi}$ or the uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$.
Search-LWE is the problem of recovering $\mathbf{s}$ from $(\mathbf{a}, c = \mathbf{a} \cdot \mathbf{s} + e \mod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ sampled according to $L_{s,\chi}$.

Considering $L_{\mathbf{s},\chi}$ as oracle outputting samples at will is usually suitable for having arbitrarily many samples available. If the maximum number of samples to use is fixed, this can be seen as a fixed set of $m > 0 \in \mathbb{Z}$ samples $\{(\mathbf{a}_1, c_1 = \mathbf{a}_1 \cdot \mathbf{s} + e_1 \mod q), \ldots, (\mathbf{a}_m, c_m = \mathbf{a}_m \cdot \mathbf{s} + e_m \mod q)\}$, often written as matrix $(\mathbf{A}, \mathbf{c}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$. This matrix is called "sample matrix" in our work. It can be shown, that Decision-LWE and Search-LWE are equivalent (see Lemma 3 in [6]).

Adopting the choice in [6], we choose χ as a discrete Gaussian distribution $\mathfrak{D}_{\mathbb{Z},\alpha q}$ on $\mathbb{Z}$ with mean zero and width parameter αq, which samples elements with probability proportional to $\exp\left(-\pi \frac{x^2}{(\alpha q)^2}\right)$. For the cases considered in this work it can be assumed that the standard deviation of a continuous Gaussian distribution $\sigma = \frac{\alpha q}{\sqrt{2\pi}}$ with width parameter αq is roughly the same as the standard deviation of the used discrete Gaussian distribution. Other than in [6], only two characterizations of LWE are considered in this work: the generic characterization by n, α, q and the small secret case, where the components of $\mathbf{s}$ are small, i.e. chosen according to a distribution ψ such that $\mathbf{s}_{(i)} \in I$ with I being a set containing small numbers, e.g. $I = \{0, 1\}$. The third characterization shown by Albrecht et al. [6] with $q \approx n^c$ and $\alpha q = \sqrt{n}$ for a small constant c is left out, since considering the most generic characterization n, α, q is sufficient.

Let $\mathfrak{U}$ be the uniform distribution over $\mathbb{Z}_q$ and $2 \leq \omega < 3$ a constant such that there is an algorithm which multiplies matrices in $\mathcal{O}(n^\omega)$ operations for sufficiently large n. At loss of n samples, an LWE instance can be constructed, where the secret vector $\mathbf{s}$ follows the same distribution as the error:

Lemma 1 ([6, Lemma 1],[7])**.** Let $\mathfrak{D}_{\mathbb{Z}^n,\alpha q}$ be an n-dimensional extension of $\mathfrak{D}_{\mathbb{Z},\alpha q}$ to $\mathbb{Z}^n$ in the obvious way, i.e. each component is sampled according to $\mathfrak{D}_{\mathbb{Z},\alpha q}$. Then, given access an oracle $L_{\mathbf{s},\chi}$ returning samples of the form $(\mathbf{a}, c) = (\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e \mod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ with $\mathbf{a} \leftarrow_\$ \mathfrak{U}(\mathbb{Z}_q^n)$, $e \leftarrow_\$ \mathfrak{D}_{\mathbb{Z},\alpha q}$ and $\mathbf{s} \in \mathbb{Z}_q^n$, we can construct samples of the form $(\mathbf{a}, c) = (\mathbf{a}, \mathbf{a} \cdot \mathbf{e} + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ with $\mathbf{a} \leftarrow_\$ \mathfrak{U}(\mathbb{Z}_q^n)$, $e \leftarrow_\$ \mathfrak{D}_{\mathbb{Z},\alpha q}$ and $\mathbf{e} \leftarrow_\$ \mathfrak{D}_{\mathbb{Z}^n,\alpha q}$ in $2n^2$ operations in $\mathbb{Z}_q$ per sample, at loss of n samples overall and with $\mathcal{O}(n^\omega)$ operations for precomputation.

The two main problems leading to the basic strategies of solving LWE are Short Integer Solutions (SIS) and Bounded Distance Decoding (BDD). We describe these problems and strategies in the Sections 2.3 and 2.4 below.

2.2.1 Learning with Errors Problem with Small Secret

For the small secret variants of the described algorithms, the components of $\mathbf{s}$ are not chosen uniformly random from $\mathbb{Z}_q$, but instead $\mathbf{s}$ is chosen from a new distribution, where all components are small. In the following, let $[a, b]$ be the interval the components of $\mathbf{s}$ are sampled from. In general, this is easier to solve than the standard variant. Considering an LWE instance with dimension n, the corresponding variant using a binary secret $[a, b] = [0, 1]$ theoretically has to have dimension $n \log q$ to be as hard as the LWE instance with non-small secret [34].

Modulus Switching for Lattice Reduction

To solve LWE instances with small secret, some algorithms use modulus switching. Let $(\mathbf{a}, c = \mathbf{a} \cdot \mathbf{s} + e \mod q)$ be a sample of an n, α, q LWE instance. If $\mathbf{s}$ is small enough, this sample can be transformed into a sample $(\tilde{\mathbf{a}}, \tilde{c})$ of an n, α', p LWE instance, where p satisfies $p < q$ and $\left\| \left(\frac{p}{q} \cdot \mathbf{a} - \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil \right) \cdot \mathbf{s} \right\| \approx \frac{p}{q} \cdot \|e\|$. The transformed samples can be constructed such that $(\tilde{\mathbf{a}}, \tilde{c}) = \left(\left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \left\lfloor \frac{p}{q} \cdot c \right\rceil \right) \in \mathbb{Z}_p^n \times \mathbb{Z}_p$, where

$$p \approx \sqrt{\frac{2\pi n}{12}} \cdot \frac{\sigma_s}{\alpha} \tag{3}$$

with σ_s being the standard deviation of the elements of the secret vector $\mathbf{s}$ [6, Lemma 2]. With the components of $\mathbf{s}$ being uniformly distributed, the variance of the elements of the secret vector $\mathbf{s}$ is determined by $\sigma_s^2 = \frac{(b-a+1)^2-1}{12}$. It is assumed, that the distribution of the secret vector $\mathbf{s}$ has mean zero. The result is an LWE instance with errors having standard deviation $\frac{\sqrt{2}\alpha p}{\sqrt{2\pi}} + \mathcal{O}(1)$ and therefore $\alpha' = \sqrt{2}\alpha$. Even though, the distribution of the error is not exactly Gaussian anymore, it can be considered to be solved by algorithms solving LWE.

So, for lattice reduction with a small secret, applying modulus switching results in an LWE instance characterized by n, $\sqrt{2}\alpha$ and p. The required δ_0 is larger in this case than in non-small secret instances without modulus switching and therefore, the lattice reduction becomes easier. Every algorithm solving with this strategy can be combined with exhaustive search guessing g components of the secret at first. Then, the algorithm runs with dimension $n-g$. Therefore, all of these algorithms can be adapted to have at most the cost of exhaustive search and potentially have an optimal g somewhere in between zero and n.

2.3 Short Integer Solutions Problem

The Short Integer Solutions (SIS) problem is defined as follows:

Definition 8 (SIS)**.** Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ consisting of n vectors $\mathbf{a}_i \in \mathbb{Z}_q^m$ chosen uniformly at random, find a vector $\mathbf{v} \neq \mathbf{0} \in \mathbb{Z}^m$, such that $\|\mathbf{v}\| \leq \beta$ with $\beta < q \in \mathbb{Z}$ and $\mathbf{v}^T\mathbf{A} = \mathbf{0} \mod q$.

Solving the SIS problem solves Decision-LWE. Given m samples written as $(\mathbf{A}, \mathbf{c})$, which either satisfy $\mathbf{c} = \mathbf{As} + \mathbf{e} \mod q$ or $\mathbf{c}$ is chosen uniformly at random, the two cases can be distinguished by finding a vector $\mathbf{v}$ in the scaled (by q) dual lattice $L^{\perp}(\mathbf{A}) = \{\mathbf{w} \in \mathbb{Z}_q^m \mid \mathbf{w}^T\mathbf{A} = \mathbf{0} \mod q\}$, such that $\mathbf{v}$ satisfies the conditions of SIS. Then, $\mathbf{v} \cdot \mathbf{c}$ either results in $\mathbf{v} \cdot \mathbf{e}$, if $\mathbf{c} = \mathbf{As} + \mathbf{e} \mod q$, or is uniformly random over $\mathbb{Z}_q$. In the first case, $\mathbf{v} \cdot \mathbf{c} = \mathbf{v} \cdot \mathbf{e}$ follows a Gaussian distribution over $\mathbb{Z}$, inherited from the distribution of $\mathbf{e}$, and is usually small. Therefore, as long as the Gaussian distribution can be distinguished from uniformly random, Decision-LWE can be solved by this procedure. To ensure this, $\mathbf{v}$ has to be short enough, since otherwise, the Gaussian distribution becomes stretched and may be too flat to distinguish from random.

Furthermore, we define the related Inhomogeneous Short Integer Solutions (ISIS) problem:

Definition 9 (ISIS)**.** Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, a vector $\mathbf{b} \in \mathbb{Z}_q^n$ and a real β, find an integer vector $\mathbf{v} \in \mathbb{Z}_q^m$ such that $\mathbf{v}^T\mathbf{A} = \mathbf{b} \mod q$ and $\|\mathbf{v}\| \leq \beta$.

2.4 Bounded Distance Decoding Problem

The Bounded Distance Decoding (BDD) problem is defined as follows:

Definition 10 (μ-BDD)**.** Given a lattice $L(\mathbf{A})$ with basis $\mathbf{A} \in \mathbb{Z}^{m \times n}$, a target vector $\mathbf{c} \in \mathbb{Z}^m$ and a bound on the distance from the target vector to the lattice $\text{dist}(\mathbf{c}, L) < \mu\lambda_1(L)$ with $\mu \leq \frac{1}{2}$, find a lattice vector $\mathbf{x} \in L$ closest to $\mathbf{c}$.

The LWE problem given m samples written as $(\mathbf{A}, \mathbf{c} = \mathbf{As} + \mathbf{e} \mod q)$ can be seen as an instance of BDD. Let the columns of $\mathbf{A}$ form a basis of a lattice $L(\mathbf{A})$. Then the point $\mathbf{w} = \mathbf{As}$ is contained by the lattice. Since $\mathbf{e}$ follows the Gaussian distribution, over 99.7% of all encountered errors are within three standard deviations of the mean (which is zero). Therefore, $\mathbf{w}$ is the closest lattice point to $\mathbf{c}$ with a very high probability. Hence, finding $\mathbf{w}$ eliminates $\mathbf{e}$. The assumption that $\mathbf{A}$ is always invertible results in a lower bound for the hardness and is therefore acceptable. Hence, the secret $\mathbf{s}$ can be calculated by inverting $\mathbf{A}$.

3 Description of Lattice Reduction Algorithms

Lattice reduction is applied to a lattice L to find a basis $\{\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}\}$ of L, such that the basis vectors $\mathbf{b}_i$ are short and nearly orthogonal to each other. Following the convention of Albrecht et al. [6], the first non-zero vector $\mathbf{b}_0$ of the basis of the reduced lattice is the shortest vector in the basis. How and why lattice reduction algorithms work is out of scope of this thesis. In the following, we describe the process of lattice reduction only shortly and instead focus on the runtime estimations of lattice reduction, because the latter is the interesting part for the analysis of the considered attacks on LWE. For a deeper contemplation, see [27, 31, 40].

The Lenstra-Lenstra-Lovász (LLL) lattice reduction algorithm is a generalization from a 2-dimensional algorithm by Lagrange. Given a basis $\mathbf{B} = \{\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}\}$ for a lattice L, the Gram-Schmidt basis $\mathbf{B}^* = \{\mathbf{b}_0^*, ..., \mathbf{b}_{n-1}^*\}$ and the Gram-Schmidt coefficients $\mu_{i,j} = \frac{\mathbf{b}_i \cdot \mathbf{b}_j^*}{\mathbf{b}_j^* \cdot \mathbf{b}_j^*}$ (for $1 \leq j < i < n$) are defined. The process basically reduces basis vectors pairwise and checks for the size reduction and Lovász conditions after each reduction [30]. The size reduction condition is defined as $|\mu_{i,j}| \leq 0.5$ for $1 \leq j < i < n$, while the Lovász condition is given by $\delta \left\|\mathbf{b}_{\mathbf{k-1}}^*\right\|^2 \leq \left\|\mathbf{b}_k^*\right\|^2 + \mu_{k,k-1}^2 \left\|\mathbf{b}_{k-1}\right\|^2$ for $1 \leq k < n$, where $\delta \in (\frac{1}{4}, 1)$ is a parameter determining the quality of the reduced basis usually set to $\delta = \frac{3}{4}$. The runtime of the LLL algorithm is determined by $\mathcal{O}(n^{5+\epsilon} \log^{2+\epsilon} B)$ with $B > \|\mathbf{b}_i\|$ for $0 \leq i \leq n-1$. Additionally, an improved variant, called L2, exists, whose runtime is estimated to be $\mathcal{O}(n^{5+\epsilon} \log B + n^{4+\epsilon} \log^2 B)$ [36], and there is a heuristically version with runtime $\mathcal{O}(n^3 \log^2 B)$ [15]. The first vector of the output basis is guaranteed to satisfy $\|\mathbf{b}_0\| \leq \left(\frac{4}{3} + \epsilon\right)^{\frac{n-1}{2}} \cdot \lambda_1(L)$ with $\epsilon > 0$.

The Blockwise Korkine-Zolotarev (BKZ) algorithm employs an algorithm for solving SVP, which is seen as SVP-oracle here. This can be done by computing the Voronoi cell of the lattice, sieving or enumeration [25]. Given an LLL-reduced basis $\mathbf{B} = \{\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}\}$ and a block size k, the first block of basis vectors is $\{\mathbf{b}_0, \ldots, \mathbf{b}_{k-1}\}$. The SVP oracle is then used to find a small vector in the space spanned by these vectors. Finally, a new LLL-reduced basis is produced for the given lattice by calling LLL iteratively on blocks created by vectors from the SVP oracle and the basis of a projected lattice. This is called a BKZ-round. The new basis is the output of this step. The algorithm terminates at the first step, where the input basis remains unchanged by the process.

There are some improvements for BKZ, namely extreme pruning [21], early termination, limiting the enumeration radius to the Gaussian Heuristic and local block pre-processing [15]. The combination of these is called BKZ2.0.

The quality of the output basis is determined by k. Choosing a larger block size k entails a better quality of the output basis but at cost of an increased runtime. While $k = 2$ only produces an LLL-reduced basis, $k = n$ results in a Hermite-Korkine-Zolotarev(HKZ)-reduced output basis. The latter is in some sense an optimally reduced basis at cost of at least exponential runtime.

Definition 11 (Geometric Series Assumption (GSA) [41])**.** Let $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ be a basis and $\mathbf{B}^* = \{\mathbf{b}_1^*, ..., \mathbf{b}_n^*\}$ be the corresponding Gram-Schmidt basis, then $\frac{\left\|\mathbf{b}_i^*\right\|^2}{\|\mathbf{b}_1\|^2} = r^{i-1}$ for $i = 1, \ldots, n$ with quotient $r \in \left[\frac{3}{4}, 1\right)$.

A limiting value of the root-Hermite factor δ_0 for BKZ can be given, assuming Gaussian heuristic and Geometric Series Assumption (GSA) hold: $\lim_{n\to\infty} \delta_0 = \left(v_k^{\frac{-1}{k}}\right)^{\frac{1}{2(k-1)}} \approx \left(\frac{k}{2\pi e}(\pi k)^{\frac{1}{k}}\right)^{\frac{1}{2(k-1)}}$, where v_k is the volume of the unit ball in dimension k. As examples show, this estimation may also be applied when n is finite [6]. As a function of k, the 'lattice rule of thumb' approximates $\delta_0 = k^{\frac{1}{2k}}$, which is often simplified to $\delta_0 = 2^{\frac{1}{k}}$. Albrecht et al. [6] show that the simplified lattice rule of thumb is a lower bound to the expected behavior on the interval $[40, 250]$ of usual values for k. The simplified lattice rule of thumb is indeed closer to the expected behavior than the lattice rule of thumb, but it implies an subexponential algorithm for solving LWE.

In the following, we need the Hermite constant and therefore show its definition here:

Definition 12 (Hermite Constant)**.** Let δ_n be the maximum lattice packing density for hypersphere packing and V_n be the content of the n-hypersphere. The Hermite constant is then defined as $\gamma_n = 4\left(\frac{\delta_n}{V_n}\right)^{\frac{2}{n}}$.

Given an n-dimensional lattice, the runtime in clock cycles is estimated to be

$$\rho \cdot n \cdot t_k \ , \tag{4}$$

where ρ is the number of BKZ rounds and t_k is the time needed to find short enough vectors in lattices of dimension k. Even though, ρ is exponential upper bounded by $(nk)^n$ at best, in practice the results after $\rho = \frac{n^2}{k^2} \log n$ rounds provide a basis whose first vector satisfies $\|\mathbf{b}_0\| \leq 2\, v_k^{\frac{n-1}{2(k-1)} + \frac{3}{2}} \cdot \det(L)^{\frac{1}{n}}$, where $v_k \leq k$ is the maximum of Hermite constants in dimensions $\leq k$, and therefore are close to the final output. [26]

Table 2 shows the theoretical complexities of the three main methods of finding shortest vectors. The second value for sieving describes the heuristic variant. The two different values for enumeration are achieved by running enumeration on

Implementations of SVP-Oracle		Number of Operations [cycles]	Memory	$\log t_{BKZ}$
Voronoi Cell		$2^{2k+o(k)}$	$2^{k+o(k)}$	$\Omega\left(\frac{-\log\log\delta_0}{\log\delta_0}\right)$
Sieving	standard	$2^{k+o(k)}$	$2^{k+o(k)}$	
	heuristic	$2^{0.2972k+o(k)}$	$2^{0.2972k+o(k)}$	
Enumeration	LLL-reduced	$2^{\mathcal{O}(k^2)}$	$poly(k)$	$\Omega\left(\frac{\log^2\log\delta_0}{\log^2\delta_0}\right)$
	quasi-HKZ-reduced	$k^{\mathcal{O}(k)}$	$poly(k)$	$\Omega\left(\frac{-\log\left(\frac{-\log\log\delta_0}{\log\delta_0}\log\log\delta_0\right)}{\log\delta_0}\right)$

Table 2: Complexities of the methods used to find shortest vectors in lattices of dimension k; t_{BKZ} is the runtime of BKZ depending on the Hermite factor δ_0

an LLL-reduced lattice or a stronger reduced quasi-HKZ basis, which requires heavier preprocessing on the input lattice. Additionally, the logarithm of the runtime complexity of BKZ depending on δ_0 is shown in the last column.

There exist several practical estimations of the runtime of BKZ in literature. Some of these results are listed in the following. Lindner and Peikert's [31] estimation is given by $\log t_{BKZ}(\delta_0) = \frac{1.8}{\log\delta_0} - 78.9$ clock cycles. This result should be used carefully, since applying this estimation implies the existence of a subexponential algorithm for solving LWE [6]. The estimation shown by Albrecht et al. [2] $\log t_{BKZ}(\delta_0) = \frac{0.009}{\log^2\delta_0} - 4.1$, called delta-squared model, is non-linear in $\log\delta_0$ and it is claimed, that this is more suitable for current implementations. The logarithmic runtime using the model of lattice rule of thumb can be given as $\mathcal{O}(k)$. "fplll" is a project implementing several lattice algorithms including LLL, BKZ and BKZ2.0 improvements [14]. Albrecht et al. [6] use curve fitting based on BKZ runtime data given by fplll [14] ($fplll$), Chen and Nguyen [15] ($enum$), Laarhoven [28] ($sieve$) and Laarhoven et al. [29] ($q-sieve$) to find functions for t_k. These functions for $fplll$, $enum$, $sieve$ and $q-sieve$ are determined to be

$$t_{k,fplll} = 0.0135k^2 - 0.2825k + 21.02 \ , \tag{5}$$

$$t_{k,enum} = 0.270189k\log k - 1.0192k + 16.10 \ , \tag{6}$$

$$t_{k,sieve} = 0.3366k + 12.31 \ , \tag{7}$$

$$t_{k,q-sieve} = 0.265k \ . \tag{8}$$

Using $\rho = \frac{n^2}{k^2}\log n$ and the functions for t_k, the overall runtime of BKZ can be estimated by Equation (4).

4 Description of Algorithms to solve the Learning with Errors Problem

In this section we describe the algorithms used to estimate the hardness of LWE and analyze them regarding their computational cost. Each section is divided into general and small secret instances, except for Bai and Galbraith's algorithm, which itself is a small secret variant of the embedding attacks.

4.1 Exhaustive Search

The exhaustive search algorithm aims to find the secret $\mathbf{s}$ in order to solve LWE. In that process no reduction to underlying problems or employing of lattice reduction is necessary. Instead, all possible guesses $\mathbf{g}$ for $\mathbf{s}$ are enumerated and tested by evaluating $\|\mathbf{Ag}-\mathbf{c}\|$. If the guess is the correct secret $\mathbf{g}=\mathbf{s}$, the result is $\|\mathbf{As}-\mathbf{c}\| = \|\mathbf{e}\|$ and therefore small.

4.1.1 General Variant of Exhaustive Search

We need Lemma 4 from [6] to specify an interval every component of $\mathbf{e}$ falls into with overwhelming probability.

Lemma 2 ([6, Lemma 4])**.** Let χ denote the Gaussian distribution with standard deviation σ and mean zero. Then, for all $C > 0$, it holds that:

$$\Pr[e \leftarrow_{\$} \chi : |e| > C \cdot \sigma] \leq \frac{2}{C\sqrt{(2\pi)}} \exp\left(-\frac{C^2}{2}\right) \ . \tag{9}$$

We write $\mathbf{g}_i$ for possible guesses of the secret vector $\mathbf{s}$. From Lemma 2 it is known, that every component of $\mathbf{e}$ falls into $[-t\alpha q, \ldots, t\alpha q]$ with $t = \omega\left(\sqrt{\log n}\right)$ with overwhelming probability. By Lemma 1 the distribution of $\mathbf{e}$ and $\mathbf{s}$ can be made the same by sacrificing n samples. Therefore, there are $2t\alpha q + 1$ possibilities for each of the n components of $\mathbf{g}_i$ when guessing the secret vector $\mathbf{s}$. The cost of the computation of an inner product can be estimated to be $2n$ operations in $\mathbb{Z}_q$, since it requires n multiplications and $n-1$ additions. Each guessed vector has to be tested for all samples m by computing an inner product. Therefore, the runtime of exhausting all guesses $\mathbf{g}_i$ is given by the number of possible guesses $(2t\alpha q+1)^n$ and the cost of testing each of them for all samples by computing an inner product each time $2n \cdot m$:

$$(2t\alpha q+1)^n \cdot 2n \cdot m \ . \tag{10}$$

In the process, the currently enumerated guess has to be stored and so, the memory complexity is n. To ensure, that vectors $\mathbf{g}_i$, which are not the correct secret, are rejected with a probability $\geq \epsilon$, m has to satisfy $m \geq \frac{\log(1-\epsilon)-n\log(2t\alpha q+1)}{\log(2t\alpha)}$, where ϵ is the success probability [6]. Therefore, when given m, one can determine the success probability by simply rearranging:

$$\epsilon = 1-(2t\alpha)^m \cdot (2t\alpha q+1)^n \ . \tag{11}$$

The number of samples is given by the sum of the required samples n to match the distribution of $\mathbf{e}$ and $\mathbf{s}$ and the number of samples m needed to provide a given target success probability.

MITM:

Furthermore, a Meet-in-the-Middle (MITM) variant of this algorithm exists [11]. It is more efficient in terms of time complexity but requires more memory. The main ideas from above still apply, but each of the m samples $(\mathbf{a}_k, c_k = \mathbf{a}_k \cdot \mathbf{s} + e_k \mod q)$ is split in half at first: $\mathbf{a}_k = {\mathbf{a}_k}^l || {\mathbf{a}_k}^r$ with ${\mathbf{a}_k}^l, {\mathbf{a}_k}^r \in \mathbb{Z}_q^{\frac{n}{2}}$ being the first ("left") and second ("right") half. Likewise, the guesses $\mathbf{g}_i$ of the secret vector $\mathbf{s}$ are considered as halved: $\mathbf{g}_i = {\mathbf{g}_i}^l || {\mathbf{g}_i}^r$. Then, a table T is constructed, which maps $\mathbf{u}_{\mathbf{g}_i{}^l}$ to ${\mathbf{g}_i}^l$, where $\mathbf{u}_{\mathbf{g}_i{}^l} = \left({\mathbf{a}_0}^l \cdot {\mathbf{g}_i}^l, \ldots, {\mathbf{a}_{m-1}}^l \cdot {\mathbf{g}_i}^l\right)$. As above, the size of each component of $\mathbf{s}$ is at most $t\alpha q$. Therefore, the cost of generating said table T in number of operations is just the same as performing standard exhaustive search on half the dimension $\frac{n}{2}$, because only the first half of $\mathbf{s}$ is considered:

$$(2t\alpha q+1)^{\frac{n}{2}} \cdot 2\frac{n}{2} \cdot m \ . \tag{12}$$

Sorting the table into lexicographical ordering costs [6, Page 16]

$$\mathcal{O}\left(m(2t\alpha q+1)^{\frac{n}{2}} \cdot \frac{n}{2} \cdot \log(m(2t\alpha q+1))\right) \ . \tag{13}$$

With the other half ${\mathbf{g}_j}^r$ of $\mathbf{g}_j$, the vector $\mathbf{v}_{{\mathbf{g}_j}^r} = (c_0 - {\mathbf{a}_0}^r \cdot {\mathbf{g}_j}^r, \ldots, c_{m-1} - {\mathbf{a}_{m-1}}^r \cdot {\mathbf{g}_j}^r,)$ is built. This vector $\mathbf{v}_{{\mathbf{g}_j}^r}$ is sorted into the lexicographical ordered table T. This can be done by binary search in $\frac{n}{2}\log(m(2t\alpha q+1))$ operations. Therefore, sorting each of the $(2t\alpha q+1)^{\frac{n}{2}}$ possible vectors $\mathbf{v}_{{\mathbf{g}_j}^r}$ into the table costs $(2t\alpha q+1)^{\frac{n}{2}} \cdot \frac{n}{2} \cdot \log(m(2\alpha q+1))$ operations [6].

Each time, a vector $\mathbf{v}_{\mathbf{g}_j{}^r}$ is sorted into the table, the two vectors $\mathbf{u}_{\mathbf{g}_i{}^l}$ it has fallen between are considered. If the distance of the vectors $\mathbf{v}_{\mathbf{g}_j{}^r}$ and $\mathbf{u}_{\mathbf{g}_i{}^l}$ satisfies the requirement specified in the following, then $\mathbf{g}_i{}^l||\mathbf{g}_j{}^r$ is accepted as the correct secret. The correct secret $\mathbf{s} = \mathbf{g}_{i'}{}^l||\mathbf{g}_{i'}{}^r$ produces $\left\|\mathbf{v}_{\mathbf{g}_{i'}{}^r} - \mathbf{u}_{\mathbf{g}_{i'}{}^l}\right\| = \|\mathbf{e} \mod q\| \leq \sqrt{m}t\alpha q$. So, in order to accept the pair $\mathbf{g}_i{}^l||\mathbf{g}_j{}^r$ as the correct secret, the distance between $\mathbf{v}_{\mathbf{g}_j{}^r}$ and $\mathbf{u}_{\mathbf{g}_i{}^l}$ has to be lower than $\sqrt{m}t\alpha q$. Otherwise, the pair is rejected.

This algorithm fails if $\mathbf{v}_{\mathbf{g}_j{}^r} - \mathbf{u}_{\mathbf{g}_i{}^l} = \mathbf{e} \mod q$ produces a wrap around mod q on any component. This happens with probability $\frac{1}{C}$ for some constant $C > 1$, if m satisfies [6]

$$2t\alpha m < \frac{1}{C} \ . \tag{14}$$

The chance of accepting a wrong pair (false positive), i.e. a pair $\mathbf{g}_i{}^l, \mathbf{g}_j{}^r$ which does not form the correct secret $\mathbf{s}$, is determined by the probability of $\mathbf{v}_{\mathbf{g}_j{}^r}$ being near to $\mathbf{u}_{\mathbf{g}_i{}^l}$. More specifically, this is the probability, that the distance between $\mathbf{v}_{\mathbf{g}_j{}^r}$ and $\mathbf{u}_{\mathbf{g}_i{}^l}$ is $\sqrt{m}t\alpha q$ at most. In other words, it is the probability, that the difference of each of the m components is at most $\pm t\alpha q$. The difference in each component is smaller than $t\alpha q$ for $2t\alpha q + 1$ of the q elements of $\mathbb{Z}_q$. Therefore, the chance of a false positive can be estimated to be $\left(\frac{2t\alpha q+1}{q}\right)^m \approx (2t\alpha)^m$ [6]. Since there are $(2t\alpha q+1)^{\frac{n}{2}}$ wrong choices for $\mathbf{g}_i{}^l$, $(2t\alpha)^m \cdot (2t\alpha q+1)^{\frac{n}{2}}$ candidates per $\mathbf{g}_j{}^r$ are expected to be tested and therefore, it is required that the following holds:

$$(2t\alpha)^m \cdot (2t\alpha q+1)^{\frac{n}{2}} = poly(n) \ . \tag{15}$$

There are two constraints. First, Equation (14) must not be satisfied and second, the requirement from Equation (15) has to be fulfilled. Assuming that the two constraints hold, the overall runtime of the MITM-variant of exhaustive search is

$$\mathcal{O}\left(m(2t\alpha q+1)^{\frac{n}{2}} \cdot \left(2n + \left(\frac{n}{2} + poly(n)\right) \cdot \log(m(2t\alpha q+1))\right)\right) \ , \tag{16}$$

with a success probability of nearly 1. The memory complexity is determined by the size of the table T. For each of the m samples, T holds one entry for every possible guess of the first half of the secret. The number of possibilities of the latter is given by $(2t\alpha q+1)^{\frac{n}{2}}$ and therefore, the overall memory complexity is given by $m(2t\alpha q+1)^{\frac{n}{2}}$. The overall number of samples $n+m$ needed consists of the number of samples n sacrificed to equalize the distributions of $\mathbf{s}$ and $\mathbf{e}$ and the number of samples m required for testing the guesses. If the two constraints related to Equations (14) and (15) as described above cannot be satisfied, this algorithm is not applicable.

4.1.2 Small Secret Variant of Exhaustive Search

For the small secret variant, the components of $\mathbf{s}$ are chosen from a given interval $[a, b]$, which consists of small numbers, e.g. $[0, 1]$. Recall the complexity of standard exhaustive search given in Equation (10), where $2t\alpha q + 1$ is the suspected number of possible values of each component of $\mathbf{s}$. Choosing the components of $\mathbf{s}$ from the given interval $[a, b]$ automatically determines the number of possible values to be $b - a + 1$. Using this and for the same reasons as in standard exhaustive search, the runtime of the small secret variant of exhaustive search is given by

$$(b-a+1)^n \cdot 2n \cdot m \ . \tag{17}$$

Similarly, the success probability ϵ is derived as before except that the number of possible values for each component of $\mathbf{s}$ is given by $b - a + 1$. Substituting $b - a + 1$ for $2t\alpha q + 1$ in Equation (11) gives:

$$\epsilon = 1 - (2t\alpha)^m \cdot (b-a+1)^n \ . \tag{18}$$

The memory requirement is independent of the number of possible values. Therefore, it remains the same and hence, it is n. The same applies to the number of required samples $n + m$.

MITM:

Substituting $b - a + 1$ for $2t\alpha q + 1$ in the derivation of the runtime and memory requirement of the MITM variant of exhaustive search gives the runtime and memory requirement of the small secret variant of MITM. The constraint from Equation (15) has to be adapted as well, resulting in

$$(2t\alpha)^m \cdot (b-a+1)^{\frac{n}{2}} = poly(n) \ . \tag{19}$$

So, assuming Equation (14) does not hold and Equation (19) is satisfied, the runtime of the small secret variant of MITM is

$$\mathcal{O}\left(m(b-a+1)^{\frac{n}{2}} \cdot \left(2n + \left(\frac{n}{2} + poly(n)\right) \cdot \log(m(b-a+1))\right)\right) \ , \tag{20}$$

while the memory requirement is $m(b-a+1)^{\frac{n}{2}}$. As in the small secret variant of standard exhaustive search, the number of samples needed is independent of the number of possible values for one component of $\mathbf{s}$ and therefore remains $n + m$.

4.2 Blum-Kalai-Wasserman

The Blum-Kalai-Wasserman (BKW) algorithm [12] originally solves Learning Parity with Noise (LPN), but can be adapted to solve LWE. If the sample matrix were noise free, Gaussian elimination could be used to solve for $\mathbf{s}$. Although there actually is noise, the BKW algorithm solves LWE similar to Gaussian elimination using more rows at once but eliminating many variables with single additions of rows rather than just one. So, the main idea is to produce what we call "final samples" in the following. In contrast, we refer to unaltered samples from $L_{\mathbf{s},\chi}$ as "original samples". Final samples have their first components reduced to zeros by adding or subtracting partly reduced and original LWE-samples in multiple stages. In the style of Gaussian elimination, the reduction is also called addition of equations, since the sample matrix can be seen as system of equations. In the final step of BKW, the produced final samples are used to solve LWE.

Elimination Tables: The considered variants of BKW share the need of "elimination tables". Elimination tables are created as first step of all variants of BKW to ease sample reduction. For this, parameters b and $a = \lceil \frac{n}{b} \rceil$ are chosen to split the n components of a sample $\mathbf{a}$ into a blocks of size b each. The processing of each block is one stage (iteration) of the algorithm. For each stage i, there is one elimination Table T_i. The input for stage i is the output of stage $i-1$ or the original sample $\mathbf{a}$ in case of $i = 1$. The output of stage i is produced by reducing the i-th block of the input to zero by adding or subtracting elements of T_i to it. This procedure is called sample reduction. Elimination tables are created consecutively, beginning with the first elimination table T_1. To create an elimination table T_i, original samples are requested, reduced using the tables $T_j (1 \leq j < i)$ according to the algorithm above and put in the i-th elimination table, until this table is complete. Then, the elimination table T_{i+1} is built, until all a elimination tables exist. An elimination table T_i is considered complete, if it contains all possible combinations of components on the i-th block. Therefore, the vectors considered in the process of filling table T_i are required to be distinct in the i-th block, that is the combination of the components $(i-1) \cdot b$ to $i \cdot b$ has to be different to those of all other vectors stored in T_i. Since elimination tables T_i $(1 < i \leq a)$ are populated by obtaining samples from $L_{s,\chi}$ and eliminating the first $(i-1) \cdot b$ components of these by addition and subtraction of elements of the elimination tables T_j $(1 \leq j < i)$, the first $(i-1) \cdot b$ components of all elements of T_i are guaranteed to be zero. Taking advantage of the symmetry of both $\mathbb{Z}_q$ and the noise distribution and the fact, that a table does not need to store entries with only zeros on a block, each T_i contains $t = \frac{q^b-1}{2}$ distinct (in the i-th block) vectors. So, for example, the first elimination table T_1 consists of all t possible vectors obtained by calling the sample-oracle $L_{s,\chi}$, which are distinct in the first b components. After a stages, there are a elimination tables, where the last one T_a contains only vectors with the first $(a-1) \cdot b$ components being 0. The remaining $n' = n - (a-1) \cdot b$ components, which is $n' = n \mod b$, if b does not divide n, and $n' = b$ otherwise, form vectors in $\mathbb{Z}_q^{n'}$ distinct from each other.

4.2.1 General Variant of Blum-Kalai-Wasserman

We consider three variants of BKW, namely the search and decision variants and Coded-BKW. BKW can be seen as a variant of the Gaussian elimination method, where a system of noise-free linear systems of equations is transformed into triangular shape, a possible solution for the resulting univariate equation is found and then this partial solution is back substituted in the original LWE sample. Since the given system of equations is not noise-free, building the triangular shape increases the standard deviation of the noise by $\sqrt{2}$ for each stage of adding pairs of equations [24].

Decision-BKW

Decision-BKW solves LWE similar to the strategy of solving SIS described in Section 2.3. This consists of two steps: first, elimination tables are created to use in the sample reduction algorithm like described above and then, these are used to construct final samples, which are used to solve Decision-LWE. In the following, we describe the second step. To solve Decision-LWE, m final samples are constructed using the elimination tables. Similar to deciding whether a noise-free linear system of equations has a common solution, it has to be tested if these final samples can be reduced to zero by sample reduction using a triangular shaped basis, represented by the elimination tables.

Let $n \geq 1$ be the dimension of the LWE secret vector, q be a positive integer and define $a = \lceil \frac{n}{b} \rceil$. As stated in the proof of Lemma 2 presented by Albrecht et al. [2] the number of operations needed to create an elimination table T_i is determined by the following: To create one of the t elements of T_i one sample reduction has to be executed on the i-th stage. One sample reduction on stage i requires $i-1$ additions of $n+1$ components (n components of a sample $\mathbf{a}$ and 1 because of c). Since on stage $j < i$ the first $j \cdot b$ components are guaranteed to be zero, these can be ignored, resulting in a decrease of additions. So, the number of operations to create the i-th elimination table T_i is given by $t \cdot \left((i-1) \cdot (n+1) - \sum_{j=1}^{i-1} j \cdot b \right)$. Summing over all T_i gives the number of operations in $\mathbb{Z}_q$ needed to create all a elimination tables:

$$\frac{q^b-1}{2} \left(\frac{a(a-1)}{2} \cdot (n+1) - \frac{ba(a-1)}{4} - \frac{b}{6} \left((a-1)^3 + \frac{3}{2}(a-1)^2 + \frac{1}{2}(a-1) \right) \right) . \tag{21}$$

The number of elements of $\mathbb{Z}_q$ needed to be stored in memory for the elimination tables is $\sum_{i=1}^{a} t \cdot (n+1-(i-1)b)$, since each table T_i has to store t entries each of which holds $n+1-(i-1)b$ elements of $\mathbb{Z}_q$ [2, Lemma 3]. Rearranging gives

$$\frac{q^b}{2} \cdot a \cdot \left(n+1-b\frac{a-1}{2}\right) \ . \tag{22}$$

To produce one final sample, i.e. a sample reduced by a stages, $\sum_{i=1}^{a}(n+1-ib)$ operations in $\mathbb{Z}_q$ are needed. Rearranging this and multiplying with m, the number of operations needed to produce m final samples is given by

$$m \cdot \left(\frac{a}{2} \cdot (n+2)\right) \ . \tag{23}$$

The total number of required operations in $\mathbb{Z}_q$ is the sum of Equations (21) and (23). The total number of original samples (calls to $L_{\mathbf{s},\chi}$) required is composed of the number of samples used to construct the elimination tables $m_{tables} = a \cdot \left\lceil \frac{q^b}{2} \right\rceil$ (see [2, Lemma 2]) and the number of samples m needed to distinguish, i.e.

$$a \cdot \left\lceil \frac{q^b}{2} \right\rceil + m \ . \tag{24}$$

Next we determine the number of samples m. To get one final sample, 2^a original samples (calls to $L_{\mathbf{s},\chi}$) are necessary, which are added or subtracted to achieve the final sample. This can be expressed as a multiplication of a 2^a-dimensional vector $\mathbf{v}$ and the matrix $\mathbf{A}$ consisting of all original samples, where the components of $\mathbf{v}$ are in $\{-1,0,1\}$ (corresponding to: -1: subtraction; 0: not part of the 2^a samples; 1: addition). This can be seen as an SIS-Problem with small vector $\mathbf{v}$ of length $\sqrt{2^a}$. Being an instance of the SIS problem, the success probability ϵ of distinguishing $L_{s,\chi}$ from uniform random is given by [6]

$$\epsilon = \left(e^{-\pi(\|\mathbf{v}\|\cdot\alpha)^2}\right)^2 \ . \tag{25}$$

Using $\|\mathbf{v}\| = \sqrt{2^a}$, the number of samples required to produce a distinguishing advantage of ϵ is [31]:

$$m = \frac{\epsilon}{\exp(-2\pi\alpha^2 2^a)} \ . \tag{26}$$

The functions for costs and required samples can be narrowed down to functions of a or b. In the following, we determine the parameter a and thus $b = \frac{n}{a}$ is known as well. Balancing the runtimes of the two steps "creation of elimination tables" and "constructing final samples" yields the optimal runtime at $a_{runtime}$ [6]. On the other hand, balancing the number of samples required by each of the two steps yields the fewest samples requirement at $a_{samples}$. The latter can be seen by examining the terms m_{tables} and m, which describe the number of samples depending on a. First, $m_{tables} = a \cdot \left\lceil \frac{q^b}{2} \right\rceil$ is basically of the form $a \cdot (q^n)^{\frac{1}{a}}$ with $q \gg 1$. So, one can see, that on the interval $(0,n]$ the term $(q^n)^{\frac{1}{a}}$ dominates, which means, that when a increases, m_{tables} decreases and eventually approaches $n \cdot \frac{q}{2}$ at $a = n$. Second, $m = \epsilon \cdot \exp(2\pi\alpha^2 2^a)$ is essentially equivalent to $\epsilon \cdot \exp(C)^{2^a}$ with $0 < \epsilon < 1$ and $\exp(C) > 1$. Therefore, when a increases, m increases rapidly. So, the fewest samples required ($m_{tables} + m$) are given at the value $a_{samples}$, for which both steps approximately require the same number of samples. Given fewer samples than this, the algorithm is not applicable. On the other hand, given more samples than needed to achieve the best runtime, allows to use the results of the unlimited samples variant of this algorithm. Therefore, if the given number of samples does not satisfy any of the two cases, the values a between $a_{samples}$ and $a_{runtime}$ have to be searched through for the best runtime, for which enough samples are provided. Given a is found, $b = \frac{n}{a}$ is determined, too.

Search-BKW

Search-BKW solves LWE by iteratively recovering parts of the secret $\mathbf{s}$. The main idea is, to reduce m samples until the $a-1$-th stage, other than in Decision-LWE, where the samples are fully reduced to zero using a stages. The result is a single block of components of $\mathbf{s}$ of size $n' = n-(a-1)\cdot b$ [1], which can be recovered by exhaustive search. Albrecht et al. [2] present a generalization of the original method, which was later improved in [19] using discrete Fourier transform (DFT). In the following, we focus our analysis on the more recent version using DFT [19]. It consists of three phases: sample reduction, hypothesis testing and back substitution. The sample reduction phase employs a creation of elimination tables as shown above. The method described in [2] introduces an additional step at the end to split the last block, such that the new last block has size $0 < d \leq n'$. Therefore, this algorithm needs a instead of $a-1$ steps and recovers d components of $\mathbf{s}$ at once by exhaustive search over q^d possibilities. The improvement in [19], on the other

[1] Recall: the last block is of size $n-(a-1)\cdot b$, which is b, if b divides n, and $n \mod b$ if not.

hand, uses a discrete Fourier transform to recover n' components of $\mathbf{s}$ in one step. The reduction phase when using DFT requires $a-1$ steps, which is one step less than the analysis given by Albrecht et al. [2]. Substituting $a-1$ for a in the results of [2] gives the number of operations in $\mathbb{Z}_q$ needed to create $a-1$ elimination tables [19]:

$$\frac{q^b-1}{2}\left(\frac{(a-1)(a-2)}{2}\cdot(n+1)-\frac{ba(a-1)(a-2)}{6}\right) . \tag{27}$$

Hypothesis testing is the second phase of Search-BKW. In this phase, parts of $\mathbf{s}$ are recovered block-wise. While Albrecht et al. [2] use maximum likelihood to find parts of $\mathbf{s}$ with high probability, in [19] DFT is used. After the sample reduction phase, m samples exist with all but n' components equal to zero. Let $\mathbf{A}\in\mathbb{Z}_q^{m\times n'}$ and $\mathbf{c}\in\mathbb{Z}_q^m$ represent these samples. Let $f(\mathbf{x})$ be a function of $\mathbf{x}\in\mathbb{Z}_q^{n'}$ defined as $f(\mathbf{x})=\sum_{j=1}^{m}\pi(j,\mathbf{x})\cdot\theta_q^{\mathbf{c}(j)}$, where $\mathbf{A}_{(j)}$ is the j-th row, $\theta_q=\exp\left(\frac{2\pi i}{q}\right)$ and $\pi(j,\mathbf{x})=\begin{cases}1, & \text{if } \mathbf{A}_{(j)}=\mathbf{x}\\ 0, & \text{otherwise}\end{cases}$. Then, the discrete Fourier transform of $f(\mathbf{x})$ is given by [19]

$$\hat{f}(\boldsymbol{\alpha})=\sum_{j=1}^{m}\theta^{-\left(\mathbf{A}_{(j)}\cdot\boldsymbol{\alpha}-\mathbf{c}_{(j)}\right)} . \tag{28}$$

The block of $\mathbf{s}$ is recovered by finding the maximum of the real part of the discrete Fourier transform of $f(\mathbf{x})$ [19]. The number of samples needed to recover the j-th block with a success probability of ϵ is [19]:

$$m_{j,\epsilon}=8\cdot b\cdot\log\left(\frac{q}{\epsilon}\right)\cdot\left(1-\frac{2\pi^2\sigma^2}{q^2}\right)^{-2^{a-j}} . \tag{29}$$

Let $\epsilon'=\frac{1-\epsilon}{a}$. The number of operations in $\mathbb{Z}_q$ required to create m samples of the $a-1$-th stage (fully reduced except for the last block) is upper bounded by $m\cdot\frac{a-1}{2}\cdot(n+2)$ as in Equation (23). After solving the j-th block, the elimination table T_j is not used anymore and can be dropped. Therefore, the number of operations in $\mathbb{Z}_q$ required to create the samples needed to recover all blocks of $\mathbf{s}$ with probability ϵ is

$$\sum_{j=0}^{a-1}m_{j,\epsilon'}\cdot\frac{a-1-j}{2}\cdot(n+2) , \tag{30}$$

while the number of operations in $\mathbb{C}$ needed for the computation of the DFTs amounts to

$$2\sum_{j=0}^{a-1}m_{j,\epsilon'}+C_{FFT}\cdot n\cdot q^b\cdot\log q , \tag{31}$$

where C_{FFT} is a small constant related to fast Fourier transform.

The result from hypothesis testing consists of some components of $\mathbf{s}$ and is used in the back substitution phase. Back substituting is the third and last phase of Search-BKW used to eliminate just as much components in each sample as there are components of $\mathbf{s}$. Since a complete block of $\mathbf{s}$ is recovered, back substituting would completely zero-out the corresponding table. So, the table can be dropped as soon as elimination makes it obsolete and therefore, fewer operations are needed in the subsequent reduction steps. Each substitution is basically an inner product of vectors of dimension b and therefore costs about $2b$ operations. There are $\frac{q^b-1}{2}$ vectors in each of the $a-1$ tables, but, since after the first block is recovered, T_{a-1} can be dropped, only $a-2$ tables have to undergo back substitution. So, the number of operations in $\mathbb{Z}_q$ required to perform back substitution is

$$(a-1)\cdot(a-2)\cdot b\cdot\frac{q^b-1}{2} . \tag{32}$$

The total number of operations in $\mathbb{Z}_q$ and $\mathbb{C}$, respectively, is given by the sum of Equations (27), (30), (31) and (32).

Since samples for hypothesis testing and back substitution can be reused for each block and $m_{0,\epsilon}>m_{j,\epsilon}$ for $j>0$, it is sufficient to create $m_{0,\epsilon}$ samples once. The memory needed to store all tables T_i follows from Equation (22) with $a-1$ instead of a stages. Additionally, q^b elements of $\mathbb{C}$ for the complex function to apply DFT on and $m_{0,\epsilon}$ samples have to be stored. So, the total memory complexity adds up to

$$\frac{q^b-1}{2}\cdot(a-1)\cdot\left(n+1-b\frac{a-2}{2}\right)+m_{0,\epsilon} \text{ in } \mathbb{Z}_q \text{ and } q^b \text{ in } \mathbb{C} . \tag{33}$$

The total number of samples required is composed of the number of samples needed to fill the tables $m_{tables} = (a-1)\cdot\frac{q^b-1}{2}$ and $m_{0,\epsilon}$ samples to apply hypothesis testing and back substitution, added up:

$$(a-1)\cdot\frac{q^b-1}{2}+m_{0,\epsilon} \ . \tag{34}$$

In case of unlimited number of samples, the best runtime can be found by choosing a and b such that the runtimes of the creation of elimination tables and recovering elements of $\mathbf{s}$ are balanced, since a small a produces reduced samples with low noise, so that it is easier to recover components of $\mathbf{s}$, but for the cost of a larger b, which induces a higher complexity of finding collisions of samples. The determination of a and b in case of a fixed number of samples is similar to Decision-BKW, since again the total number of samples consists of two parts. First, m_{tables} can be approximated with the same term as in Decision-BKW as follows: $m_{tables} = (a-1)\cdot\frac{q^b-1}{2} \approx a\cdot(q^n)^{\frac{1}{a}}$ with $q \gg 1$. Second, even though $m(0,\epsilon)$ has a minimum at some a_{min}, the fewest samples are required at the value $a_{samples}$, for which the number of samples of the two steps are roughly equal, since $m(0,\epsilon)$ at $a < a_{min}$ is negligible compared to m_{tables} and increases rapidly for $a > a_{min}$. Therefore, just like in Decision-BKW, there are three cases when given a fixed number of samples. If there are fewer samples than needed to reach $a_{samples}$, the algorithm fails. Given more samples than required to achieve the best runtime, given by balancing the runtimes of the two steps, the unlimited samples variant of this algorithm can be used. If the given number of samples is in between those two limits, values a in between have to be checked for the best runtime, at which enough samples are present. As soon as a is found, $b = \frac{n}{a}$ is determined, too.

Coded-BKW

Like the other variants of BKW presented above, Coded-BKW as described by Guo et al. [24] requires sample reduction and therefore elimination tables. It is a variant of Search-BKW with a modified sample reduction phase. Recall the sample reduction phase of Search-BKW: a large number of samples $\mathbf{a}_k$ is used to find block-wise collisions and then, these are used to reduce the dimension of $\mathbf{s}$. In Coded-BKW on the i-th iteration, a q-ary linear lattice code $\mathscr{C}_i$ with parameters (N_i, b) is introduced at first, so that the i-th block $(\mathbf{a}_k)_i$ of $\mathbf{a}_k$ can be expressed by the codeword $(\mathbf{C}_k)_i \in \mathscr{C}_i$ and the corresponding error $(\mathbf{E}_k)_i \in \mathbb{Z}_q^{N_i}$, where the euclidean norm of $(\mathbf{E}_k)_i$ is minimal:

$$(\mathbf{a}_k)_i = (\mathbf{C}_k)_i + (\mathbf{E}_k)_i \ . \tag{35}$$

Given samples $\mathbf{a}_k$ on the i-th step, the corresponding block is split into codeword and error by a decoding procedure. Samples are then reduced by subtracting vectors mapped to the same codeword. Let $(\mathbf{s})_i$ be the corresponding block of $\mathbf{s}$ and consider $(\mathbf{s})_i\cdot(\mathbf{a}_k)_i = (\mathbf{s})_i\cdot(\mathbf{C}_k)_i + (\mathbf{s})_i\cdot(\mathbf{E}_k)_i$. Then, the subtraction of two vectors mapped to the same codeword removes $(\mathbf{s})_i\cdot(\mathbf{C}_k)_i$, so that only the coding error is left. In the i-th step, N_i components are removed. Therefore, after t modified BKW steps $\sum_{i=1}^{t} N_i$ components are removed. Since $N_i \geq b$, more components are removed per step than in Search-BKW but at cost of an additional noise term. This noise is increased exponentially in the subsequent iteration steps. So, a mixture of standard BKW steps and Coded-BKW steps should be used with standard BKW steps at the beginning. In the following, let t_1 be the number of standard BKW steps performed at the beginning and t_2 be the number of Coded-BKW steps after.

The analysis of the algorithm given by Guo et al. in [24] splits the algorithm into five steps. Let n be split up into $n = n_{top} + n_{cod} + n_{test} + n_{standard}$. First, the distribution of the secret vector $\mathbf{s}$ has to be transformed into the distribution of the error, which can be estimated (in number of operation in $\mathbb{Z}_q$) to be [24]

$$C_0 = (m-n')\cdot(n+1)\cdot\left\lceil\frac{n'}{b-1}\right\rceil \ , \tag{36}$$

where $n' = n - t_1 b$.

Second, the t_1 standard BKW reductions are applied, like described in Search-BKW. The result is a new set of LWE samples with dimension $n - n_{standard}$, since $n_{standard} = t_1\cdot b$ components are reduced to zero. The number of operations in $\mathbb{Z}_q$ to do this is estimated to be

$$C_1 = \sum_{i=1}^{t_1}(n+1-ib)\left(m-\frac{i(q^b-1)}{2}\right) \ . \tag{37}$$

Third, the t_2 coded BKW reductions are applied, where each step uses a (N_i, b) code. The value N_i is given by $N_i = \left\lfloor\frac{b}{1-\frac{1}{2}\log_q(12\frac{\sigma_{set}}{2^t})}\right\rfloor$, where σ_{set} is determined to be $\frac{q^{2(1-\frac{l}{n_{test}})}}{12}$, because of the error of the (n_{test}, l) code used in the

fifth phase (see below). The number of components reduced to zero by coded BKW steps are given by $n_{cod} = \sum_{i=1}^{t_2} N_i$. The decoding cost are upper bounded by

$$C_2' = \sum_{i=1}^{t_2} 4\left(M + \frac{i(q^b-1)}{2}\right)N_i \ , \tag{38}$$

where M is the number of required samples for testing, i.e. the result after the last coded BKW step. The total cost for the coded BKW steps is estimated to be

$$C_2 = C_2' + \sum_{i=1}^{t_2}(n_{top} + n_{test} + \sum_{j=1}^{i} N_j)\left(M + \frac{(i-1)(q^b-1)}{2}\right) \ . \tag{39}$$

In the fourth step, n_{top} components of $\mathbf{s}$ are guessed by exhaustively searching the $(2d+1)^{n_{top}}$ possibilities, where d is the suspected number of possibilities of each component of $\mathbf{s}$. Setting $d = 3 \cdot \sigma$ gives a high probability, that every component of $\mathbf{s}$ is between $-d$ and d. The estimation of the cost of this step is determined as in Equation (10) presented in the Section Exhaustive Search (Section 4.1). Here, the dimension is n_{top} instead of n and each component may be one of $2d+1$ possibilities instead of $2t\alpha q + 1$:

$$C_3 = M \cdot n_{top} \cdot (2d+1)^{n_{top}} \ . \tag{40}$$

Fifth, for each such guess, subspace hypothesis testing is performed using a (n_{test}, l) linear code with $l = b-1$ and FFT. The reduced samples from the previous phases are grouped according to their nearest codeword in the (n_{test}, l) systematic linear code. This is used to build q^l polynomials, which have to be evaluated q times, and these values are stored. Then another polynomial is evaluated using q FFTs, which records occurrences of Gaussian distributed errors. The resulting candidates are tested using a Neyman-Pearson test, which assigns belief levels to the candidates. The final result is then the one with the highest rank. The decoding cost are upper bounded by

$$C_4' = 4Mn_{test} \ . \tag{41}$$

The total number of operations in $\mathbb{Z}_q$ needed is estimated to be

$$C_4 = C_4' + (2d+1)^{n_{top}}\left(C_{FFT} \cdot q^{l+1}(l+1)\log q + q^{l+1}\right) \ , \tag{42}$$

where C_{FFT} is a small constant related to fast Fourier transform.

Let $P(d)$ be the probability, that, when guessing one component of $\mathbf{s}$, the interval $[-d, d]$ is sufficient. $P(d)$ can be lower bounded by $\mathrm{erf}\left(\frac{d}{\sqrt{2}\sigma}\right) < P(d)$. The testing phase may fail with probability $P_{test} = \left(\gamma e^{\frac{1-\gamma^2}{2}}\right)^{n_{cod}+n_{test}}$ due to the information subvector to be tested being larger than $\gamma\sqrt{n_{cod}+n_{test}}\sigma$ and therefore being to large to distinguish. Guo et al. [24] set $\gamma = 1.2$ and claim a probability in most of the applications larger than 97.5%.

The total number of operations in $\mathbb{Z}_q$ is given by

$$C = \frac{C_0 + C_1 + C_2 + C_3 + C_4}{P(d)^{n_{top}} \cdot P_{test}} \ . \tag{43}$$

The memory requirement is determined by the storage of elimination tables, which is given by

$$(t1 + t2) \cdot q^b \ . \tag{44}$$

The total number of samples required is given by the number of samples M needed for testing and the number of samples used for reduction $\frac{(t1+t2)(q^b-1)}{2}$. M has to be large enough to distinguish between uniform and Gaussian distribution, where $\sigma_{final}^2 = 2^{t1+t2}\sigma^2{}^2 + \gamma^2\sigma^2\sigma_{set}^2(n_{cod} + n_{test})$. The results from Equation (26) in Decision-BKW can be reused to estimate

$$M = \frac{\epsilon}{\left(\exp\left(-\pi\left(\frac{\sigma_{final}}{q}\right)^2\right)\right)^2} \ . \tag{45}$$

So, the total number of samples required add up to

$$m = \frac{(t1 + t2)(q^b - 1)}{2} + M \ . \tag{46}$$

The parameter t_1 can be determined by observing, that any $N_i \leq b$ gives no advantage over Search-BKW and therefore, t_1 is the number of times N_i is lower or equal b. So, for a given LWE instance (n, α, q) and a given total number of available samples $m_{available}$, parameters t_2 and b have to be found, such that the total number of operations C in $\mathbb{Z}_q$ (see Equation (43)) is minimal and the number of samples m required does not exceed $m_{available}$. This can be done by evaluating this process at varying t_2 and b according to a evaluating order similar to binary search.

To prevent confusion of secret bounds $[a, b]$ and block width b with number of blocks a, the parameters determining the interval, from which elements of the secret vector $\mathbf{s}$ are sampled, are named $[s_{min}, s_{max}]$ in this section.

Decision-BKW

The algorithm proposed by Albrecht et al. [3] solves Decision-LWE with small secret. Like with non-small secret, this consists of two steps: sample reduction and hypothesis testing. For this, only the sample reduction phase is modified using lazy modulus switching. Basically, this means to only switch modulus from q to p with $p < q$ when necessary. Therefore, while searching for collisions, only the first $\log p$ bits are considered instead of all $\log q$ bits. If there is a collision on the p most significant bits of the corresponding block of the two vectors, the elimination is applied in $\mathbb{Z}_q$ as before. Considering $p = 2^\kappa$, the elimination tables store only κ bits for each entry and only $\frac{p^b-1}{2}$ vectors per table are required. Let $\mathbf{a}_{(b\cdot l, b\cdot l+b)}$ be the block on stage l, then these vectors in the elimination tables are of the form $\left\lfloor \frac{p}{q} \cdot \mathbf{a}_{(b\cdot l, b\cdot l+b)} \right\rceil$. Since the elimination of a block takes place in $\mathbb{Z}_q$, but the collision affected only the p most significant bits, there is a new noise term in the fully reduced samples $(\tilde{\mathbf{a}}_i, \tilde{c}_i = \tilde{\mathbf{a}}_i\mathbf{s} + \tilde{e}_i \mod q)$ of $|\tilde{\mathbf{a}}_i \cdot \mathbf{s}|$. This is called "rounding error" by Albrecht et al. [3] and it should be of the same size as the noise $\tilde{e}_i$, such that neither dominates, i.e.

$$|\tilde{\mathbf{a}}_i \cdot \mathbf{s}| \approx \sqrt{2^a}\alpha q \ . \tag{47}$$

Albrecht et al. [3] propose a technique, where additional samples are reduced at first to produce "good" tables. As described, the sample reduction procedure finds collisions with entries of the elimination tables. For this, the entries of the tables are always retained. For the proposed technique, each time a collision is found, either the existing entry is kept in the table or the sample, with which a collision was found, replaces it. To do this, a chosen number of m^* additional samples have to be reduced using existing elimination tables and potentially inserted into them, replacing the old entries. Since lazy modulus switching is used, the reduced components are not necessarily zero. So, the decision, whether to keep the table entry or replace it, is based on the size of the previously "eliminated" elements of the two vectors. This is beneficial, because the sizes of these elements are carried on through all subsequent stages. The final samples used to solve Decision-LWE are sampled as before.

For the analysis of Decision-BKW with small secret using lazy modulus switching, Albrecht et al. [3] use two assumptions:

Assumption 1 ([3, Assumption 1])**.** The samples after every reduction step are assumed to be independent.

Assumption 2 ([3, Assumption 2])**.** Let the vectors $\mathbf{x}_0, \ldots, \mathbf{x}_{n-1} \in \mathbb{Z}_q^\tau$ be sampled from some distribution $\mathfrak{D}$ such that $\sigma^2 = \mathrm{Var}(\mathbf{x}_{i,(j)})$ where $\mathfrak{D}$ is any distribution on (sub-)vectors occurring in this attack. Let $\mathbf{x}^* = \min_{abs}(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1})$ where $\min_{abs}$ picks that vector $\mathbf{x}^*$ with $\sum_{j=0}^{b\cdot l-1} |\mathbf{x}^*_{(j)}|$ minimal. The stddev $\sigma_n = \sqrt{\mathrm{Var}(\mathbf{x}^*_{(0)})} = \cdots = \sqrt{\mathrm{Var}(\mathbf{x}^*_{(\tau-1)})}$ of components in $\mathbf{x}^*$ satisfies

$$\frac{\sigma}{\sigma_n} \geq c_\tau n^{\frac{1}{\tau}} + (1 - c_\tau) \tag{48}$$

with $c_\tau = 0.20151418166952917\sqrt{\tau} + 0.32362108131969386 \approx \frac{1}{5}\sqrt{\tau} + \frac{1}{3}$ for $\tau > 10$.

To estimate the standard deviation of the total noise, a matrix $\mathbf{M}$ is constructed, where the entry $\mathbf{M}_{(i,j)}$ represents the variance of entries $(b\cdot j, \ldots, b\cdot j + b - 1)$ in the table T_i. Albrecht et al. [3] propose an algorithm calculating this variance matrix $\mathbf{M}$ using Assumption 2. Let σ_r be the standard deviation of uniformly random elements in $Z_{\lfloor q/p \rceil}$ and let $(\tilde{\mathbf{a}}_i, \tilde{c}_i)$ be samples after a stages of elimination, then $\mathbf{v}$ is defined as a vector, where $\mathbf{v}_{(k)}$ is the variance of the components b_k through $b_k + b - 1$ of $\tilde{\mathbf{a}}_i$. Albrecht et al. [3] show that this can be expressed like in the following lemma.

Lemma 3 ([3, Lemma 2])**.** Let $n \geq 1$, q be a modulus, $b \in \mathbb{Z}$ with $1 \leq b \leq n$ and σ_r be the standard deviation of the uniform distribution over $\mathbb{Z}_{\lfloor q/p \rceil}$. Define $a = \left\lceil \frac{n}{b} \right\rceil$ and pick some $p < q$ and let $\mathbf{M}$ be the output of the algorithm proposed by Albrecht et al. under these parameters. Let $(\tilde{\mathbf{a}}_i, \tilde{c}_i)$ be samples after a reduction steps modulus p. Finally, define $\mathbf{v}$ as the a-vector of variances of the components of $\tilde{\mathbf{a}}$ where $\mathbf{v}_{(k)}$ holds the variance of the components $\tilde{\mathbf{a}}_{(b\cdot k)}$ to $\tilde{\mathbf{a}}_{(b\cdot k+b-1)}$. Under Assumption 2, the components of $\mathbf{v}$ satisfy:

$$\mathbf{v}_{(i)} = \sigma_r^2 + \sum_{j=i+1}^{a} \mathbf{M}_{(j,i)} \ . \tag{49}$$

Let σ_s be the standard deviation of the secret $\mathbf{s}$. The distribution of $\tilde{c}_i$ can then be approximated as Gaussian distribution with standard deviation σ_{total}.

Lemma 4 ([3, Lemma 3])**.** Let $n \geq 1$ be the dimension of the LWE secret vector, q be a modulus, $b \in \mathbb{Z}$ with $1 \leq b \leq n$. Define $a = \lceil \frac{n}{b} \rceil$ and pick some $p < q$ and let $\mathbf{v}$ be as in Lemma 3. Let $(\tilde{\mathbf{a}}_i, \tilde{c}_i)$ be samples after a reduction steps. We assume that Assumptions 1 and 2 hold. Then as a increases the distribution of $\tilde{c}_i$ approaches a discrete Gaussian distribution modulo q with standard deviation

$$\sigma_{total} = \sqrt{2^a\sigma + b\sigma_r^2\sigma_s^2 \sum_{i=0}^{a-1} \mathbf{v}_{(i)}} \leq \sqrt{2^a\sigma + (2^a - 1) \cdot b \cdot \sigma_r^2\sigma_s^2} \ . \tag{50}$$

Using Lemma 4, the number of samples needed to distinguish LWE samples from samples chosen uniformly at random can be calculated. Similar to other algorithms solving Decision-LWE, the advantage of distinguishing LWE from uniformly random is estimated as $\exp\left(-\pi\left(\frac{\sigma_{total}\cdot\sqrt{2\pi}}{q}\right)^2\right)$.

Let p and m^* be chosen, such that the requirement in Equation (47) is satisfied, i.e. $b\sigma_r^2\sigma_s^2 \sum_{i=0}^{a-1} \mathbf{v}_{(i)} \leq 2^a\sigma$. Recall the required number of operations needed for Decision-BKW with non-small secret. It is composed of the number of operations needed to create the elimination tables (see Equation (21)) and the number of operations required to produce m final (reduced) samples for the final distinguishing (see Equation (23)). The same applies to Decision-BKW with lazy modulus switching on small secret instances of LWE, but additionally, m^* samples have to be reduced. So, the number of operations needed to create the elimination tables remains as before, but instead of only reducing m samples, $m + m^*$ samples are reduced. Therefore, the cost in number of operations in $\mathbb{Z}_q$ can be estimated by

$$\begin{aligned} &\frac{p^b}{2} \cdot \left(\frac{a(a-1)}{2} \cdot (n+1) - \frac{ba(a-1)}{4} - \frac{b}{6}\left((a-1)^3 + \frac{3}{2}(a-1)^2 + \frac{1}{2}(a-1)\right)\right) + (m+m^*) \cdot \left(\frac{a}{2} \cdot (n+2)\right) \\ <&\frac{p^b}{2} \cdot \left(\frac{a(a-1)}{2} \cdot (n+1)\right) + (m+m^*)na \ . \end{aligned} \tag{51}$$

The memory requirement is the same as in Decision-BKW, given by $\frac{p^b}{2} \cdot a \cdot \left(n + 1 - b \cdot \frac{a-1}{2}\right) < \frac{p^b}{2} \cdot a \cdot (n+1)$. The required number of samples consists of the number of samples needed to fill the elimination tables $a \cdot \frac{p^b}{2}$ and the number of reduced samples $m + m^*$. In total, $a \cdot \frac{p^b}{2} + m + m^*$ initial samples are required.

Coded-BKW

To adapt Coded-BKW to work on LWE instances with small secret, only slight changes have to be made to its procedure. First, the suspected number d of possibilities of each component of $\mathbf{s}$ does not have to be estimated like in the fourth step of Coded-BKW with non-small secret. Instead it is determined by the bounds $[s_{min}, s_{max}]$ of the secret vector $\mathbf{s}$. To be precise, it is given by $d = s_{max} - s_{min} + 1$.

Second, σ_{final} has to calculated slightly different. Let $\sigma_s = \frac{(s_{max} - s_{min} + 1)^2 - 1}{12}$ be the variance for uniform distribution given bounds $[s_{min}, s_{max}]$. Then, the second term of σ_{final}^2, being the "coding variance", has to be changed to $\sigma_s\sigma_{set}^2(n_{cod} + n_{test})$. So, in total, the new σ_{final} is determined as follows: $\sigma_{final}^2 = 2^{t1+t2}\sigma^2 + \sigma_s\sigma_{set}^2(n_{cod} + n_{test})$.

Since, starting from Coded-BKW on LWE instances with non-small secret, these are the only changes to be made for solving instances with small secret, the steps of the algorithm and the estimations remain the same, but with the new values d and σ_{final} discussed above.

4.3 Using Lattice Reduction to Distinguish

The using lattice reduction to distinguish algorithm or "distinguishing attack" solves Decision-LWE via the SIS strategy by only employing lattice reduction. For this, the dual lattice defined by the sample matrix is considered. Lattice reduction is applied to this lattice to find a short vector in the lattice. The result is used as short vector $\mathbf{v}$ in the SIS problem to distinguish the Gaussian distribution from uniformly random. By doing so, the Decision-LWE problem is solved.

To find a short vector $\mathbf{v}$ in the context of solving LWE using the SIS strategy, lattice reduction on the scaled dual lattice $L^\perp(\mathbf{A}) = \{\mathbf{w} \in \mathbb{Z}_q^m \mid \mathbf{w}^T\mathbf{A} = \mathbf{0} \mod q\}$ can be used. To construct the basis of this dual lattice given $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$, one extends the matrix $\mathbf{A}$ by $q\mathbf{I} \in \mathbb{Z}^{m\times m}$ and reduces this using LLL. After deleting all columns only consisting of zeros, the result is the basis of L. The dimension of the dual lattice $L^\perp$ is m, with high probability the rank is m and the volume $\text{vol}(L) = q^n$ [35].

Recall the two conditions of SIS, the non-zero vector $\mathbf{v}$ has to satisfy as described in Section 2.3: First, the length of $\mathbf{v}$ has to be short enough ($\|\mathbf{v}\| \leq \beta$ with $\beta < q \in \mathbb{Z}$) and second $\mathbf{v}^T\mathbf{A} = \mathbf{0} \mod q$. Applying lattice reduction to L sets the shortest non-zero vector $\mathbf{b}_0$ found as the first vector of the reduced basis. So, $\mathbf{b}_0$ satisfies $\mathbf{b}_0^T\mathbf{A} = \mathbf{0} \mod q$. Therefore, $\mathbf{b}_0$ fulfills the second requirement for the vector $\mathbf{v}$. If $\mathbf{b}_0$ is also short enough to satisfy the first requirement of SIS, then it can be used as the wanted, short vector $\mathbf{v}$. If $\mathbf{b}_0$ is not short enough, a better lattice reduction has to be applied to $L^\perp$. Usually, instead of $\mathbf{b}_0$, every vector of the reduced basis can be used, since they mostly do not vary in length considerably.

4.3.1 General Variant of Using Lattice Reduction to Distinguish

The vector $\mathbf{v}$ is found like described above by simply applying lattice reduction to $L^{\perp}$. Therefore, it is set to $\mathbf{v} = \mathbf{b}_0$. As described in Section 2.3, $\mathbf{v} \cdot \mathbf{c}$ is considered. By deciding, whether this product is Gaussian distributed or uniformly random, Decision-LWE is solved.

The success probability ϵ is the advantage of distinguishing $\mathbf{v} \cdot \mathbf{e}$ from uniformly random and can be approximated by standard estimates [6]

$$\epsilon = e^{-\pi(\|\mathbf{v}\| \cdot \alpha)^2} \ . \tag{52}$$

The length of $\|\mathbf{v}\| = \frac{1}{\alpha}$ results in an advantage of about $\epsilon = \frac{1}{23}$. But to achieve a fixed success probability ϵ, a vector $\mathbf{v}$ of length $\|\mathbf{v}\| = \frac{1}{\alpha}\sqrt{\ln\left(\frac{1}{\epsilon}\right)/\pi}$ is needed, as one can see by Equation (52). To shorten equations, let $f(\epsilon) = \sqrt{\ln\left(\frac{1}{\epsilon}\right)/\pi}$. As one can see by rearranging Equation (52), the length of $\mathbf{v}$ for a fixed success probability ϵ is

$$\|\mathbf{v}\| = \frac{1}{\alpha} f(\epsilon) \ . \tag{53}$$

Recall the definitions of the volume of a lattice $\mathrm{vol}(L) = q^n$ and the Hermite factor $\delta_0^m = \frac{\|\mathbf{v}\|}{\mathrm{vol}(L)^{1/m}}$. Substituting these into Equation (53) gives the log root-Hermite factor δ_0 required to achieve a success probability of ϵ to distinguish $\mathbf{v} \cdot \mathbf{e}$ from uniformly random:

$$\log \delta_0 = \frac{1}{m} \log\left(\frac{1}{\alpha} f(\epsilon)\right) - \frac{n}{m^2} \log q \ , \tag{54}$$

where m is the total number of samples used in the process. The determination of δ_0 is sufficient to specify the runtime of this attack, since it solely depends on lattice reduction. In Table 3 the runtimes for the cases from Section 3 are shown.

Model	Runtime
rule of thumb	$\mathcal{O}(k)$, for k determined by $\frac{k}{\log k} = \frac{1}{2} \cdot \frac{m^2}{m \log\left(\frac{1}{\alpha} f(\epsilon)\right) - n \log q}$
simpl. rule of thumb	$\mathcal{O}\left(\frac{m^2}{m \log\left(\frac{1}{\alpha} f(\epsilon)\right) - n \log q}\right)$
Lindner & Peikert	$\frac{1.8 \cdot m^2}{m \log\left(\frac{1}{\alpha} f(\epsilon)\right) - n \log q} - 78.9$
delta-squared	$\frac{0.009 \cdot m^4}{\left(m \log\left(\frac{1}{\alpha} f(\epsilon)\right) - n \log q\right)^2} + 4.1$

Table 3: Logarithmic runtimes of the using lattice reduction to distinguish algorithm for different models introduced in Section 3

However, the algorithm is significantly faster when working with a less restricted (in terms of length), hence longer, vector. On the other hand, using a longer vector reduces the success probability. To achieve an overall success probability of about 1 the algorithm has to be run multiple times. The number of repetitions is determined to be $\frac{1}{\epsilon^2}$ by Chernoff bound [16]. So, let $T(\epsilon, m)$ be the runtime of a single execution of the algorithm shown in Table 3. Then, the best overall runtime is the minimum of $\frac{1}{\epsilon'^2} \cdot T(\epsilon', m)$. Though, it has to mentioned, that, when reusing the fixed number of samples m in every run instead of having available an unlimited number of samples, the runtime of a single run is not independent of each other anymore, since the same error vector $\mathbf{e}$ is used in every run. It is not clear, what the influence of this is. Since it is not known exactly, it has to be considered conservatively in order to present a lower bound to the runtime.

4.3.2 Small Secret Variant of Using Lattice Reduction to Distinguish

The using lattice reduction to distinguish algorithm for small secrets works the same as in the non-small secret case, but it exploits the smallness of the secret $\mathbf{s}$ by applying modulus switching at first. Recall the derivation of $\log \delta_0$ in Equation (54). In the small secret case with interval $[a, b]$, applying modulus switching technique results in an LWE instance characterized by $n, \sqrt{2}\alpha, p$. Using the same reasoning as in the standard case, the required $\log \delta_0$ for said LWE instance is given by

$$\log \delta_0 = \frac{1}{m} \log\left(\frac{1}{\sqrt{2}\alpha} f(\epsilon)\right) - \frac{n}{m^2} \log p \ , \tag{55}$$

Model	Runtime
rule of thumb	$\mathcal{O}(k)$, for k determined by $\frac{k}{\log k} = \frac{1}{2} \cdot \frac{m^2}{m \log\left(\frac{1}{\sqrt{2}\alpha} f(\epsilon)\right) - n \log p}$
simpl. rule of thumb	$\mathcal{O}\left(\frac{m^2}{m \log\left(\frac{1}{\sqrt{2}\alpha} f(\epsilon)\right) - n \log p}\right)$
Lindner & Peikert	$\frac{1.8 \cdot m^2}{m \log\left(\frac{1}{\sqrt{2}\alpha} f(\epsilon)\right) - n \log p} - 78.9$
delta-squared	$\frac{0.009 \cdot m^4}{\left(m \log\left(\frac{1}{\sqrt{2}\alpha} f(\epsilon)\right) - n \log p\right)^2} + 4.1$

Table 4: Logarithmic runtimes of the small secret variant of the using lattice reduction to distinguish algorithm for different models introduced in Section 3

where p can be estimated by Equation (3). The rest of the algorithm remains the same as in the standard case. Therefore, the overall runtime is still determined by the runtime of the lattice reduction, but with the $\log \delta_0$ from Equation (55). This is shown in Table 4. Also, the variation of success probabilities like in the standard case should be considered. Combining this algorithm with exhaustive search like described in Section 2.2.1 may improve the runtime.

4.4 Decoding Approach

The decoding approach solves LWE via the BDD strategy described in Section 2.4. The procedure considers the lattice $L = L(\mathbf{A})$ formed by the sample matrix $\mathbf{A}$ and consists of two steps: the reduction step and the decoding step. In the reduction step, lattice reduction on L is employed to get "good" Gram-Schmidt vectors. Then, these are used in the decoding phase to find the close lattice vector $\mathbf{w} = \mathbf{As}$ used in the BDD strategy to eliminate the error vector $\mathbf{e}$. In the following let the "target success probability" be the overall success probability of the attack, chosen by the attacker (usually close to 1). In contrast, the success probability refers to the success probability of a single run of the algorithm. The target success probability is achieved by running the algorithm potentially multiple times with a certain success probability for each single run.

Also, in the following the fundamental parallelepiped is defined as follows:

Definition 13 (Fundamental Parallelepiped)**.** Let $\mathbf{X}$ be a set of n vectors $\mathbf{x}_i$ and $\alpha_i \in \left[-\frac{1}{2}, \frac{1}{2}\right)$. Then $P_{1/2}(\mathbf{X}) = \sum_{i=0}^{n-1} \alpha_i \mathbf{x}_i$ is the fundamental parallelepiped of $\mathbf{X}$.

4.4.1 General Variant of Decoding Approach

To solve BDD, and therefore LWE, the most basic algorithm is Babai's Nearest Plane algorithm [9]. Given a BDD instance $(\mathbf{A}, \mathbf{c} = \mathbf{As} + \mathbf{e} \mod q)$ from m samples, it consists of two steps. First, lattice reduction on the lattice $L = L(\mathbf{A})$ is used, which results in a new basis $\mathbf{B} = \{\mathbf{b}_0, ..., \mathbf{b}_{n-1}\}$ for L with root-Hermite factor δ_0, and then the decoding step is applied. In the decoding step, the sublattice L' with basis $\mathbf{B}' = \{\mathbf{b}_0, ..., \mathbf{b}_{n-2}\}$ and the Gram-Schmidt basis $\mathbf{B}^* = \{\mathbf{b}_0^*, ..., \mathbf{b}_{n-1}^*\}$ are considered. These are used to recursively find the closest vector $y \in L'$ to a new target vector $\mathbf{c}'$, which is formed by orthogonally projecting $\mathbf{c}$ onto the affine subspace $U + v$, where U is the linear subspace spanned by $\mathbf{b}_0, ..., \mathbf{b}_{n-2}$ and $v \in L$, such that the distance between $U + v$ and $\mathbf{c}$ is minimal. For the orthogonal projection, $\mathbf{c}$ is considered as linear combination of $\mathbf{B}^*$, for which then the last coefficient is rounded to its nearest integer.

The result of the algorithm is the lattice point $\mathbf{w} \in L$, such that $\mathbf{c} \in \mathbf{w} + P_{1/2}(\mathbf{B}^*)$. Therefore, the algorithm is only able to recover $\mathbf{s}$ correctly from $\mathbf{c} = \mathbf{As} + \mathbf{e} \mod q$ if and only if $\mathbf{e}$ lies in $P_{1/2}(\mathbf{B}^*)$. The success probability of the Nearest Plane algorithm is the probability of $\mathbf{e}$ falling into $P_{1/2}(\mathbf{B}^*)$:

$$\Pr\left[\mathbf{e} \in P_{1/2}(\mathbf{B}^*)\right] = \prod_{i=0}^{m-1} \Pr\left[|\mathbf{e} \cdot \mathbf{b}_i^*| < \frac{\mathbf{b}_i^* \cdot \mathbf{b}_i^*}{2}\right] = \prod_{i=0}^{m-1} \operatorname{erf}\left(\frac{\left\|\mathbf{b}_i^*\right\| \sqrt{\pi}}{2\alpha q}\right) , \tag{56}$$

For this, it is assumed that sampling from the discrete Gaussian is approximately the same as from a continuous Gaussian, which is a valid assumption as long as the discrete Gaussian is sufficiently wide. So, for Babai's Nearest Plane algorithm, an attacker can only adjust his overall runtime according to the trade-off between the quality of the lattice reduction and the success probability.

Lindner and Peikert [31] present a modification of the Nearest Plane algorithm named Nearest Planes. They declare, that the Gram-Schmidt vectors $\mathbf{B}^*$ of a reduced basis $\mathbf{B}$ often vary in length. More specifically, the first vectors tend to be

rather long, while the last ones usually are very short. This results in a "long and skinny" parallelepiped $P_{1/2}(\mathbf{B}^*)$, which does not match the structure of the Gaussian distributed error $\mathbf{e}$. This means, that the error vector $\mathbf{e}$ is unlikely to fall into the parallelepiped and therefore, the success probability of the Nearest Plane algorithm is rather low. To compensate this problem, the Nearest Planes algorithm of Lindner and Peikert adds an additional parameter $d_i \geq 1$ to the decoding step, which describes how many nearest planes the algorithm takes into account on the i-th level of recursion. The effect of the values d_i is essentially, that the parallelepiped $P_{1/2}(\mathbf{B}^*)$ is stretched in the direction of $\mathbf{b}_i^*$ with factor d_i. This means, that instead of only finding the one, nearest plane, d_i nearest planes are considered. Then, the recursion of the next level is started with each of the d_i possibilities. This leads to not only one, but $\prod_{i=0}^{n-1} d_i$ distinct possible lattice points, which has to be searched through exhaustively for the correct result.

The success probability of the Nearest Planes algorithm is again the probability of $\mathbf{e}$ falling into the parallelepiped. For the Nearest Planes algorithm, the parallelepiped to be considered is stretched with factors $\mathbf{d} = \{d_0, \ldots, d_{n-1}\}$. Formally written, the fundamental parallelepiped to be considered is $P_{1/2}(\mathbf{B}^* \cdot \mathrm{diag}(\mathbf{d}))$. Therefore, the success probability considering the stretching factors d_i is given as follows:

$$\Pr\left[\mathbf{e} \in P_{1/2}(\mathbf{B}^* \cdot \mathrm{diag}(\mathbf{d}))\right] = \prod_{i=0}^{m-1} \Pr\left[|\mathbf{e} \cdot \mathbf{b}_i^*| < d_i \cdot \frac{\mathbf{b}_i^* \cdot \mathbf{b}_i^*}{2}\right] = \prod_{i=0}^{m-1} \mathrm{erf}\left(\frac{d_i \left\|\mathbf{b}_i^*\right\| \sqrt{\pi}}{2\alpha q}\right) \ . \tag{57}$$

Although, there is no method known to determine the optimal d_i, Lindner and Peikert's way of choosing d_i is intuitive: To cover the Gaussian distribution of $\mathbf{e}$ best possible, the shortest Gram-Schmidt vectors $\left\|\mathbf{b}_i^*\right\|$ should be stretched most and so, they suggest to maximize $\min(d_i \left\|\mathbf{b}_i^*\right\|)$. As long as the values d_i are powers of 2, this can be shown to be optimal [6]. For a fixed success probability, the optimal values d_i can be found iteratively. In each iteration, the value d_i, for which $d_i \left\|\mathbf{b}_i^*\right\|$ is currently minimal, is increased by one step size (usually step size = 1). Then, the success probability given by Equation (57) is calculated again. If the result is better or equal than the chosen, fixed success probability, the iteration stops. [31]

An attacker can choose the parameters δ_0 and d_i, which determine the success probability ϵ of the algorithm. He will presumably try to minimize his overall runtime

$$T = \frac{T_{BKZ} + T_{NP}}{\epsilon} \ , \tag{58}$$

where T_{BKZ} is the runtime of the lattice reduction with chosen target quality δ_0, T_{NP} is the runtime of the decoding step with chosen d_i multiple recursions and ϵ is the success probability achieved by δ_0 and d_i. The runtime of a single execution of Nearest Planes is only the numerator $T_{BKZ} + T_{NP}$. For a fixed success probability, a more reduced basis means it is sufficient to choose smaller values d_i, which results in a smaller T_{NP}. However, to achieve a more reduced basis, the lattice reduction algorithm takes significantly more time. The other way round, with a less reduced basis much larger values d_i are required to satisfy the fixed success probability due to the low quality of the resulting basis. Although this means that the lattice reduction algorithm needs less time, the time spent by the decoding step is significantly higher. This means, that T_{BKZ} and T_{NP} are monotonously in δ_0. Using this, it can be shown, that the overall runtime T is less or equal than two times the best runtime. Therefore, to estimate the overall runtime it is reasonable to assume that the lattice reduction and the decoding step are balanced. To give a more precise and conservative estimation, one bit has to be subtracted from the number of operations, since the estimation is up to a factor of 2 worse than the optimal runtime.

The runtime of the lattice reduction is determined by δ_0 as described in Section 3. The values d_i cannot be expressed by a formula and therefore, there is also no closed form of δ_0. As a consequence, the runtime of the lattice reduction step cannot be explicitly given here. It has to be found by iteratively varying values for δ_0 until the runtimes of the two steps are balanced as described above.

The runtime of the decoding step for Babai's Nearest Plane algorithm is determined by the complexity of the Gram-Schmidt orthogonalization, which is about $2mn^2$ floating point operations or $\mathcal{O}(mn^2)$ in asymptotic notation [23]. For Lindner and Peikert's Nearest Planes algorithm, this is still a factor, but it gets dominated by the number of points considered, when analyzing the runtime of the algorithm. So, the runtime of the decoding step of the Nearest Planes algorithm is determined by the number of points $\prod_{i=0}^{m-1} d_i$ that have te be exhausted and the time t_{node} it takes to process one point:

$$T_{NP} = t_{node} \cdot \prod_{i=0}^{m-1} d_i \ . \tag{59}$$

Since no closed formula is known to calculate the values d_i, they are computed by step-wise increasing like described above until the success probability calculated by Equation (57) reaches the fixed success probability. Albrecht et al. [6], using an estimation of Lindner and Peikert [31], determine t_{node}, which is equivalent to the runtime of a single execution

of Babai's Nearest Plane algorithm, to be $t_{node} \approx 10^{15.1}$ clock cycles. So, the runtime T_{BKZ} depends on δ_0, whereas T_{NP} depends on d_i, which is equivalent to depending on δ_0 [2] and the fixed success probability.

The runtime of the lattice reduction is determined by the desired quality δ_0 of the lattice, where higher quality entails longer runtime. For the decoding step, the runtime is also influenced by δ_0, where higher quality means less runtime, since lower values d_i can be used. So, it is possible to find a δ_0 for which these two steps are somewhat balanced.

Since this contemplation only considers a fixed success probability, the best trade-off between success probability and the runtime of a single execution described above must be found by repeating the process above with varying values of the fixed success probability.

In difference to having as many samples as needed, being restricted to a limited number of samples m, in general, does not yield the optimal number of samples $m_{optimal}$ for a specific δ_0. Nevertheless, if given $m < m_{optimal}$ samples, as much samples as possible should be used in the procedure described above. Otherwise, $m_{optimal}$ samples should be used.

4.4.2 Small Secret Variant of Decoding Approach

The decoding approach for small secrets works the same as in the non-small secret case, but it exploits the smallness of the secret $\mathbf{s}$ by applying modulus switching at first. Switching from the usual n, α, q instance to $n, \sqrt{2}\alpha, p$ allows for larger δ_0 and thus the lattice reduction becomes easier. Therefore, a better lattice reduction can be applied. As in the standard case, the two steps reduction and decoding have to be balanced. Also, the runtime of the decoding step decreases when applying better lattice reduction in the reduction step. Therefore after balancing the runtimes of the two steps, the runtimes of both steps are lower for the small secret case than in the general variant. Combining this algorithm with exhaustive search like described in Section 2.2.1 may improve the runtime.

4.5 Standard Embedding

Standard Embedding, also called "Kannan attack" or "reducing BDD to uSVP", is an algorithm to solve Search-LWE using embedding and lattice reduction. For that, the usual lattice formed by the sample matrix is embedded into a lattice of higher dimension. By applying lattice reduction on the embedding lattice, uSVP can be solved in the higher-dimensional lattice. Since the BDD problem can be reduced to the uSVP problem, solving uSVP in the embedding lattice solves Search-LWE.

4.5.1 General Variant of Standard Embedding

The $(\frac{1}{2\gamma})$-BDD problem can be reduced to the γ-uSVP by embedding the usual lattice of the given BDD instance into a higher-dimensional lattice. That is, given a sample matrix $(\mathbf{A}, \mathbf{c} = \mathbf{As} + \mathbf{e} \mod q)$ and a lattice $L(\mathbf{A})$ from a BDD instance, the matrix of the higher-dimensional lattice $L(\mathbf{A}')$ is defined as

$$\mathbf{A}' = \begin{pmatrix} \mathbf{A} & \mathbf{c} \\ \mathbf{0}^T & t \end{pmatrix} , \tag{60}$$

where t is the embedding factor defined as the distance from $\mathbf{c}$ to $L(\mathbf{A})$, i.e. $t = \text{dist}(\mathbf{c}, L(\mathbf{A})) = \|\mathbf{c} - \mathbf{x}\|$ where $\mathbf{x} \in L(\mathbf{A})$, such that $\|\mathbf{c} - \mathbf{x}\|$ is minimized. If $t < \frac{\lambda_1(L(\mathbf{A}))}{2\gamma}$, the higher-dimensional lattice $L(\mathbf{A}')$ has a γ-unique shortest vector $\mathbf{c}' = \begin{pmatrix} -\mathbf{e} \\ t \end{pmatrix} \in \mathbb{Z}_q^{m+1}$ [33] with length $\|\mathbf{c}'\| \approx \sqrt{m}\alpha q$ [17], since $\mathbf{c}' = \begin{pmatrix} -\mathbf{e} \\ t \end{pmatrix} = \mathbf{A}' \begin{pmatrix} -\mathbf{s} \\ 1 \end{pmatrix}$. Therefore, $\mathbf{e}$ can be extracted from $\mathbf{c}'$ and thus, $\mathbf{As}$ is known, which can be solved for $\mathbf{s}$ as stated in Section 2.4.

Theoretically, the γ-uSVP can be solved by solving the corresponding κ^2-HSVP with $\gamma = \kappa^2$. This would lead to a gap of $\lambda_2(L) > \kappa^2 \lambda_1(L)$, while Gama and Nguyen [20] show that uSVP instances can be solved with some probability depending on τ by applying a lattice reduction algorithm, if

$$\lambda_2(L) > \tau \delta_0^m \lambda_1(L) \ . \tag{61}$$

Here, τ is a constant, experimentally determined depending on the class of the lattice, the lattice reduction algorithm and the target success probability.

For determining the success probability and analyzing the runtime, τ has to be found, which depends on the embedding factor t. There are two cases to be considered: $t = \|\mathbf{e}\|$ and $t < \|\mathbf{e}\|$ (especially including $t = 1$).

[2] Remember: the Gram-Schmidt basis $\mathbf{B}^*$ needed in Equation (57) depends on the quality δ_0 of the reduced basis.

Case $t = \|\mathbf{e}\|$:
For the analysis, the following lemma is needed:

Lemma 5 ([4, Lemma 2]). Let $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, let $\alpha q > 0$ and let $\epsilon' > 1$. Let $\mathbf{e} \in \mathbb{Z}_q^m$ such that each component is drawn from χ and considered mod q. Under the assumption that the Gaussian heuristic holds for $L(\mathbf{A})$ and that the rows of $\mathbf{A}$ are linearly-independent over $\mathbb{Z}_q$, we can create an embedding lattice with λ_2/λ_1-gap greater than

$$\frac{\lambda_2}{\lambda_1} \geq \frac{\min\left(q, q^{1-\frac{n}{m}}\sqrt{\frac{m}{2\pi e}}\right)}{\epsilon' \alpha q \sqrt{m} \frac{1}{\sqrt{\pi}}} \tag{62}$$

with probability greater than $1 - \left(\epsilon' \cdot \exp\left(\frac{1-\epsilon'^2}{2}\right)\right)^2$.

The dimension of the embedding lattice is $\dim(L(\mathbf{A}')) = m + 1$, but for simplicity and to use Lemma 5 unchanged, we assume $\dim(L(\mathbf{A}')) = m$. This is reasonable, since m is large.

Using Lemma 5 and assuming $q^{1-\frac{n}{m}}\sqrt{\frac{m}{2\pi e}} < q$ gives the gap-requirement $\frac{\lambda_2}{\lambda_1} \leq \frac{q^{1-\frac{n}{m}}\sqrt{\frac{1}{2e}}}{\epsilon' \alpha q}$. Substituting this into the relation from Equation (61) gives the following equation, which δ_0 has to satisfy: $q^{1-\frac{n}{m}}\sqrt{\frac{1}{2e}} \geq \tau \delta_0^m \epsilon' \alpha q$. By simple rearrangement, the requirement for δ_0 is given as follows:

$$\delta_0 \leq \left(\frac{q^{1-\frac{n}{m}}\sqrt{\frac{1}{2e}}}{\tau \epsilon' \alpha q}\right)^{\frac{1}{m}} , \tag{63}$$

where m is the total number of samples used in the process. Here, τ is experimentally determined to be $\tau \leq 0.4$ for a success probability of $\epsilon_\tau = 0.1$ [4]. The probability ϵ to successfully solve LWE follows from the multiplication of ϵ_τ and the probability, that the requirement for $\frac{\lambda_2}{\lambda_1}$ from above is fulfilled:

$$\epsilon = \epsilon_\tau \cdot \left(1 - \left(\epsilon' \cdot \exp\left(\frac{1-\epsilon'^2}{2}\right)\right)^m\right) . \tag{64}$$

From there, δ_0 can be computed and thus all necessary parameters are known to calculate the runtime estimations of the underlying lattice reduction algorithm. In Table 5 the runtimes for the cases from Section 3 are shown.

Model	Runtime
rule of thumb	$\mathcal{O}(k)$, for k determined by $\frac{k}{\log k} = \frac{1}{2} \cdot \frac{m}{\log\left(q^{1-\frac{n}{m}}\sqrt{\frac{1}{2e}}\right) - \log(\tau\epsilon'\alpha q)}$
simpl. rule of thumb	$\mathcal{O}\left(\frac{m}{\log\left(q^{1-\frac{n}{m}}\sqrt{\frac{1}{2e}}\right) - \log(\tau\epsilon'\alpha q)}\right)$
Lindner & Peikert	$\frac{1.8 \cdot m}{\log\left(q^{1-\frac{n}{m}}\sqrt{\frac{1}{2e}}\right) - \log(\tau\epsilon'\alpha q)} - 78.9$
delta-squared	$\frac{0.009 \cdot m^2}{\left(\log\left(q^{1-\frac{n}{m}}\sqrt{\frac{1}{2e}}\right) - \log(\tau\epsilon'\alpha q)\right)^2} + 4.1$

Table 5: Logarithmic runtimes of standard embedding for different models introduced in Section 3

As discussed above, the success probability ϵ of a single run depends on τ and thus does not necessarily yield the desired target success probability ϵ_{target}. For example, $\tau = 0.3$ (as in the case $t < \|\mathbf{e}\|$ below) in a usual lattice leads to a success probability of $\epsilon \approx 0.1$, while one usually wants to have a target success probability close to 1. Therefore, if the success probability is lower than the target success probability, the algorithm has to be repeated ρ times, until the desired target success probability is reached. To be more precise, the following equation has to be fulfilled:

$$\epsilon_{target} = 1 - (1 - \epsilon)^\rho \tag{65}$$

Consequently, for estimating the hardness, it has to be considered, that ρ executions of this algorithm have to be done. Most important, the runtime has to multiplied by ρ. The samples, on the other hand, may be reused in each run, so that - as conservative lower bound - the number of samples needed to execute this algorithm does not have to be changed.

Case $t < \|\mathbf{e}\|$:

Albrecht, Fitzpatrick and Göpfert [4] observe, that choosing $t < \|\mathbf{e}\|$ leads to a more efficient attack, especially when choosing $t = 1$, since the gap λ_2/λ_1 usually increases. However, choosing $t = 1$ implies, that $L(\mathbf{A}')$ contains a vector shorter than $\mathbf{c}'$ with a non-zero probability and, as a consequence, the method sometimes fails to extract the error vector $\mathbf{e}$.

Since there is no method known to determine the gap λ_2/λ_1 in an efficient way, Albrecht, Fitzpatrick and Göpfert [4] assume the gap to be the same as in the case of choosing $t = \|\mathbf{e}\|$. To fix this, the value τ is modified to $\tau' \approx 0.3$.

4.5.2 Small Secret Variant of Standard Embedding

There are two approaches to solve a small secret LWE instance based on embedding. First, modulus switching can be applied to exploit the smallness of the secret vector $\mathbf{s}$. The standard embedding attack on LWE with small secret using modulus switching works the same as standard embedding in the non-small secret case, except that it operates on instances characterized by $n, \sqrt{2}\alpha, p$ instead of n, α, q with $p < q$. This allows for larger δ_0 and therefore for an easier lattice reduction. To be more precise, the requirement for δ_0 from Equation (63) changes as follows. The reasoning for the derivation of δ_0 remains the same as in the general variant, but the transformed parameters have to be used. Thus, δ_0 has to satisfy

$$\delta_0 \leq \left(\frac{p^{1-\frac{n}{m}} \sqrt{\frac{1}{2e}}}{\tau\epsilon' \sqrt{2}\alpha p} \right)^{\frac{1}{m}} , \tag{66}$$

where p can be estimated by Equation (3). As stated in the description of the standard case, the overall runtime of the algorithm is determined by the runtime of the lattice reduction, which can be calculated as soon as δ_0 is known. In Table 6 the runtimes for the models described in Section 3 are shown. The success probability remains the same.

Model	Runtime
rule of thumb	$\mathcal{O}(k)$, for k determined by $\frac{k}{\log k} = \frac{1}{2} \cdot \frac{m}{\log\left(q^{1-\frac{n}{m}} \sqrt{\frac{1}{2e}}\right) - \log\left(\tau\epsilon' \sqrt{2}\alpha p\right)}$
simpl. rule of thumb	$\mathcal{O}\left(\frac{m}{\log\left(q^{1-\frac{n}{m}} \sqrt{\frac{1}{2e}}\right) - \log\left(\tau\epsilon' \sqrt{2}\alpha p\right)} \right)$
Lindner & Peikert	$\frac{1.8 \cdot m}{\log\left(q^{1-\frac{n}{m}} \sqrt{\frac{1}{2e}}\right) - \log\left(\tau\epsilon' \sqrt{2}\alpha p\right)} - 78.9$
delta-squared	$\frac{0.009 \cdot m^2}{\left(\log\left(q^{1-\frac{n}{m}} \sqrt{\frac{1}{2e}}\right) - \log\left(\tau\epsilon' \sqrt{2}\alpha p\right)\right)^2} + 4.1$

Table 6: Logarithmic runtimes of the small secret variant of standard embedding for different models introduced in Section 3

Combining this algorithm with exhaustive search like described in Section 2.2.1 may improve the runtime.

Second, Bai and Galbraith's embedding [11] can be used. Even though this is based on dual lattices, it is somewhat similar to standard embedding and therefore can be seen as a variant solving LWE with small secret. The description can be found in Section 4.7.

4.6 Dual Embedding

Dual embedding [10] is very similar to standard embedding shown in Section 4.5. Here, LWE is also solved by reducing BDD to uSVP, but with a different lattice. Again, the embedding technique together with lattice reduction is used. The usual lattice formed by the sample matrix is embedded into a higher-dimensional dual lattice. By applying lattice reduction on the embedding lattice, uSVP can be solved in the higher-dimensional lattice. Since the BDD problem can be reduced to the uSVP problem, solving uSVP in the embedding lattice solves Search-LWE. Compared to the standard embedding attack, the dual embedding algorithm runs in dimension $n + m + 1$ instead of $m + 1$, while the number of required samples remains m. Therefore, it is more suitable for instances, which are restricted to a fixed number of samples [17].

4.6.1 General Variant of Dual Embedding

Let $L(\mathbf{A})$ be a lattice and $\mathbf{c} = \mathbf{As} + \mathbf{e} \mod q$ be a vector from a given BDD instance. Furthermore, let the matrix $\mathbf{A}_o \in \mathbb{Z}_q^{m \times (n+m+1)}$ be defined as

$$\mathbf{A}_o = \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \mathbf{c} \end{pmatrix} , \tag{67}$$

where $\mathbf{I}_m \in \mathbb{Z}^{m\times m}$ is the identity matrix. Then, $L^{\perp}(\mathbf{A}_o) = \{\mathbf{v} \in \mathbb{Z}^{n+m+1} \mid \mathbf{A}_o\mathbf{v} = \mathbf{0} \mod q\}$ is the lattice, in which the uSVP operates. Considering $\mathbf{v} = (\mathbf{s}, \mathbf{e}, -1)^T$ leads to $\mathbf{A}_o\mathbf{v} = \mathbf{A}\mathbf{s} + \mathbf{e} - \mathbf{c} = \mathbf{0} \mod q$ and therefore $\mathbf{v} \in L^{\perp}(\mathbf{A}_o)$. The length of $\mathbf{v}$ is small and can be estimated to be $\|\mathbf{v}\| \approx \sqrt{n+m}\sigma$ [17].

Since this attack is similar to standard embedding, its basic estimations of the success probability and the runtime can be applied for this attack as well. Taking into account the change in dimension and using the same simplification as in standard embedding, the root-Hermite delta has to be

$$\delta_0 = \left(\frac{q^{\frac{m}{m+n}} \sqrt{\frac{1}{2e}}}{\tau \epsilon' \alpha q} \right)^{\frac{1}{n+m}}, \tag{68}$$

while the number of samples used in the process is m.

The success probability is determined by the same characteristics as in standard embedding. That is, the probability ϵ to successfully solve LWE is given by the multiplication of ϵ_τ, which is the probability that the process determined by τ is successful, and the probability, that the requirement for the gap $\frac{\lambda_2}{\lambda_1}$ is fulfilled. The result can be seen in Equation (64).

Similar to standard embedding, the runtime of this attack is determined by the runtime of the lattice reduction achieving the root-Hermite-factor δ_0 described in Equation (68). In Table 7 the runtimes for the cases from Section 3 are shown.

Model	Runtime
rule of thumb	$\mathcal{O}(k)$, for k determined by $\frac{k}{\log k} = \frac{1}{2} \cdot \frac{n+m}{\log\left(q^{\frac{m}{m+n}} \sqrt{\frac{1}{2e}}\right) - \log(\tau\epsilon'\alpha q)}$
simpl. rule of thumb	$\mathcal{O}\left(\frac{n+m}{\log\left(q^{\frac{m}{m+n}} \sqrt{\frac{1}{2e}}\right) - \log(\tau\epsilon'\alpha q)} \right)$
Lindner & Peikert	$\frac{1.8 \cdot (n+m)}{\log\left(q^{\frac{m}{m+n}} \sqrt{\frac{1}{2e}}\right) - \log(\tau\epsilon'\alpha q)} - 78.9$
delta-squared	$\frac{0.009 \cdot (n+m)^2}{\left(\log\left(q^{\frac{m}{m+n}} \sqrt{\frac{1}{2e}}\right) - \log(\tau\epsilon'\alpha q)\right)^2} + 4.1$

Table 7: Logarithmic runtimes of dual embedding for different models introduced in Section 3

Since this algorithm is not mentioned by Albrecht et al. [6] and therefore the analysis in the case of unlimited number of samples is not done, this analysis is shown here. The case, where the number of samples is not limited and thus, the optimal number of samples can be used, is a special case of the discussion above. To be more precise, it is the case, where $m = m_{optimal}$ with $m_{optimal}$ being the number of samples for which the runtime is minimal. For this, the parameter m, for which δ_0 determined by Equation (68) is maximal, has to be found. This yields the lowest runtime using dual-embedding. The success probability is determined just like in the general case discussed above.

Similar to standard embedding, the success probability of a single run may be - depending on τ - rather low in both the general and the optimal case. To compensate this, the algorithm has to be repeated several times, so that the desired target success probability is reached. For details, see the discussion for Equation (65) in standard embedding (Section 4.5).

4.6.2 Small Secret Variant of Dual Embedding

Just like for the small secret variant of standard embedding shown in Section 4.5.2, there are two approaches to solve a small secret LWE instance based on embedding. First, modulus switching can be applied to exploit the smallness of the secret vector $\mathbf{s}$. The dual embedding attack on LWE with small secret using modulus switching works the same as dual embedding in the non-small secret case, except that it operates on instances characterized by $n, \sqrt{2}\alpha, p$ instead of n, α, q with $p < q$. This allows for larger δ_0 and therefore for an easier lattice reduction. This means, that Equation (68) has to be adapted taking the transformed parameters into account. This can be done by simply substituting the respective parameters, because the derivation of the equation remains the same. This results in the following equation for δ_0:

$$\delta_0 = \left(\frac{p^{\frac{m}{m+n}} \sqrt{\frac{1}{2e}}}{\tau \epsilon' \sqrt{2}\alpha p} \right)^{\frac{1}{n+m}}, \tag{69}$$

where p can be estimated by Equation (3). Since the runtime of the algorithm is determined by the runtime of the lattice reduction, it can be calculated when knowing δ_0. The runtimes of the different models of lattice reduction discussed

Model	Runtime
rule of thumb	$\mathcal{O}(k)$, for k determined by $\frac{k}{\log k} = \frac{1}{2} \cdot \frac{n+m}{\log\left(q^{\frac{m}{m+n}}\sqrt{\frac{1}{2e}}\right) - \log\left(\tau\epsilon'\sqrt{2\alpha p}\right)}$
simpl. rule of thumb	$\mathcal{O}\left(\frac{n+m}{\log\left(q^{\frac{m}{m+n}}\sqrt{\frac{1}{2e}}\right) - \log\left(\tau\epsilon'\sqrt{2\alpha p}\right)}\right)$
Lindner & Peikert	$\frac{1.8\cdot(n+m)}{\log\left(q^{\frac{m}{m+n}}\sqrt{\frac{1}{2e}}\right) - \log\left(\tau\epsilon'\sqrt{2\alpha p}\right)} - 78.9$
delta-squared	$\frac{0.009\cdot(n+m)^2}{\left(\log\left(q^{\frac{m}{m+n}}\sqrt{\frac{1}{2e}}\right) - \log\left(\tau\epsilon'\sqrt{2\alpha p}\right)\right)^2} + 4.1$

Table 8: Logarithmic runtimes of the small secret variant of dual embedding for different models introduced in Section 3

in Section 3 are adapted for the small secret variant of dual embedding in Table 8. The success probability remains the same. Combining this algorithm with exhaustive search like described in Section 2.2.1 may improve the runtime.

The other approach is Bai and Galbraith's embedding algorithm [11]. This is very similar to dual embedding due to the usage of a dual lattice. The major difference lies in the exploitation of the smallness of $\mathbf{s}$ and the resulting changes to the volume of the lattice used in the estimation of δ_0. The description can be found in Section 4.7.

4.7 Bai and Galbraith's Embedding

The algorithm of Bai and Galbraith [11] solves Search-LWE with a small secret vector $\mathbf{s}$ by embedding. The main idea is to reduce an ISIS instance to CVP in a lattice and solve this by solving uSVP with the embedding technique. Similar to dual embedding described in Section 4.6, a dual lattice embedding the usual lattice formed by the sample matrix is defined. Let $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ and $\mathbf{c} \in \mathbb{Z}_q^m$ be the given samples. Furthermore, let $\mathbf{I}_m$ be the m-dimensional identity matrix and let $m' = n+m$. Then, the dual lattice is defined as $L^\perp(\mathbf{A}') = \{\mathbf{v} \in \mathbb{Z}^{m'} \mid \mathbf{A}'\mathbf{v} = \mathbf{0} \mod q\}$, where $\mathbf{A}' = \begin{pmatrix}\mathbf{A} & \mathbf{I}_m\end{pmatrix} \in \mathbb{Z}_q^{m\times m'}$. Consider the ISIS instance $\mathbf{c} = \mathbf{A}'\mathbf{y} \mod q$ with target vector $\mathbf{y} = \begin{pmatrix}\mathbf{s}\\ \mathbf{e}\end{pmatrix}$. Let then $\mathbf{w} \in \mathbb{Z}^{n+m}$ be $\mathbf{w} = \begin{pmatrix}\mathbf{0}\\ \mathbf{c}\end{pmatrix}$ so that $\mathbf{A}'\mathbf{w} = \mathbf{c} \mod q$ and let $\mathbf{v} \in L$ be close to $\mathbf{w}$, found by solving the corresponding closest vector problem (CVP). Then, the target vector $\mathbf{y}$ is given by the subtraction of $\mathbf{w}$ and $\mathbf{v}$, i.e.

$$\mathbf{y} = \begin{pmatrix}\mathbf{s}\\ \mathbf{e}\end{pmatrix} = \mathbf{w} - \mathbf{v} \ . \tag{70}$$

The vector $\mathbf{y}$ is a short vector, which satisfies $\mathbf{A}'\mathbf{y} = \mathbf{A}'(\mathbf{w}-\mathbf{v}) = \mathbf{c}$. Since $\|\mathbf{s}\| \ll \|\mathbf{e}\|$, the CVP algorithm has to find an unbalanced solution. To tackle this, the lattice should be scaled so that it is more balanced. This can be done by multiplying the first n rows of the basis of L by some factor depending on σ [11]. This results in an increase of the volume of the lattice without significantly increasing the norm the error vector. Additionally, it can be further balanced by rearranging the interval around 0. By using these balancing techniques the volume of the lattice is increased by a factor of $(\xi\sigma)^n$, where $\xi = \frac{2}{b-a}$ with $[a,b]$ being the interval the secret vector $\mathbf{s}$ is sampled from. This increases the δ_0 needed to successfully execute this algorithm.

The required δ_0 can be determined similar to the steps used for standard embedding in Section 4.5. The requirement for the gap $\frac{\lambda_2}{\lambda_1}$ is not exactly like in Lemma 5, since there are differences in the dimension and the volume of the lattice. Taking these into account, the gap is given by: $\frac{\lambda_2}{\lambda_1} > \frac{\min\left(q, (q^m\cdot(\xi\sigma)^n)^{\frac{1}{n+m}}\sqrt{\frac{n+m}{2\pi e}}\right)}{\sigma\sqrt{2m+n}}$ [11]. Also, consider the assumption $(q^m\cdot(\xi\sigma)^n)^{\frac{1}{n+m}}\sqrt{\frac{n+m}{2\pi e}} \le q$ and the relation $\lambda_2(L) > \tau\delta_0^{m'}\lambda_1(L)$ from Equation (61). Assuming equality for simplification and putting all three equations together results in the following requirement for δ_0:

$$\log\delta_0 = \frac{m'\left(\log\left(\frac{q}{\sigma}\right) - \log(2\tau\sqrt{\pi e})\right) + n\log\xi - n\log\left(\frac{q}{\sigma}\right)}{m'^2} \ . \tag{71}$$

Just like for dual embedding, m samples are used in the process. The runtime is determined like for the other embedding attacks, which only depend on lattice reduction. In Table 9 the runtimes for the cases from Section 3 are shown.

The contemplation of τ in standard embedding in Section 4.5 applies here, too. So, for the cases $t = \|\mathbf{e}\|$ and $t = 1$, τ and τ' are used, respectively. Likewise, the success probability is determined by the probability ϵ_τ corresponding to the chosen τ and the probability, that the gap $\frac{\lambda_2}{\lambda_1}$ is as described above (see Equation (64)). Therefore, the success probability of a single run may be - depending on τ - rather low. To compensate this, the algorithm has to be repeated several times,

Model	Runtime
rule of thumb	$\mathcal{O}(k)$, for k determined by $\frac{k}{\log k} = \frac{1}{2} \cdot \frac{m'^2}{m'\left(\log\left(\frac{q}{\sigma}\right) - \log(2\tau\sqrt{\pi e})\right) + n\log\xi - n\log\left(\frac{q}{\sigma}\right)}$
simpl. rule of thumb	$\mathcal{O}\left(\frac{m'^2}{m'\left(\log\left(\frac{q}{\sigma}\right) - \log(2\tau\sqrt{\pi e})\right) + n\log\xi - n\log\left(\frac{q}{\sigma}\right)}\right)$
Lindner & Peikert	$\frac{1.8 \cdot m'^2}{m'\left(\log\left(\frac{q}{\sigma}\right) - \log(2\tau\sqrt{\pi e})\right) + n\log\xi - n\log\left(\frac{q}{\sigma}\right)} - 78.9$
delta-squared	$\frac{0.009 \cdot m'^4}{\left(m'\left(\log\left(\frac{q}{\sigma}\right) - \log(2\tau\sqrt{\pi e})\right) + n\log\xi - n\log\left(\frac{q}{\sigma}\right)\right)^2} + 4.1$

Table 9: Logarithmic runtimes of the Bai-Galbraith-Embedding attack for different models introduced in Section 3

so that the desired target success probability is reached (see standard embedding, Equation (65)). Combining this algorithm with exhaustive search like described in Section 2.2.1 may improve the runtime.

In contrast to the other algorithms using lattice reduction, Bai and Galbraith state, that applying modulus switching to their algorithm does not result in an improvement. The reason for this is, that modulus switching reduces q by a larger factor than it reduces the size of the error. Therefore, a smaller rescaling factor is used und thus, a smaller gap is produced, which is a crucial property.

5 Implementation

In this section, we describe our implementation of the results presented in Sections 4.1 to 4.7 as an extension of the LWE-Estimator introduced by Albrecht et al. [5, 6]. Also, we give an explanation of the usage of our software on the basis of examples and describe the structure of the code. Furthermore, we compare exemplary results of our implementation to those of the existing LWE-Estimator and present a comparison of the considered algorithms, focusing on the behavior when limiting the number of available samples.

The LWE-Estimator by Albrecht et al. is available at `https://bitbucket.org/malb/lwe-estimator`. We base our extension on the version of the LWE-Estimator lastly updated at 22-October-2016 (commit-id: 9c95373). Our software is written in Sage [18] and will be publicly available at `https://www.cdc.informatik.tu-darmstadt.de/cdc/personen/nina-bindel/`. We adapt each algorithm the LWE-Estimator implements to take a fixed number of samples into account instead of assuming unlimited many samples were available, except for Arora and Ge's algorithm based on Gröbner bases. Instead, our implementation includes dual-embedding (described in Section 4.6) in both the optimal number of samples version and the version adapted to fixed numbers of samples. The reason for this decision is, that Arora and Ge's algorithm requires too many samples while dual-embedding is much more suitable for LWE instances with limited number of samples. Therefore, in the implementation of our work there exist seven algorithms together with their respective small secret variants. Following the notation of Albrecht et al. [6], we assign an abbreviation to each algorithm to refer to when using the implementation:

- "mitm": Exhaustive Search, discussed in Section 4.1
- "bkw": Coded-BKW, discussed in Section 4.2 (Decision-BKW and Search-BKW are separated from this and are not assigned an abbreviation)
- "sis": Using Lattice Reduction to Distinguish, discussed in Section 4.3
- "dec": Decoding, discussed in Section 4.4
- "kannan": Standard Embedding, discussed in Section 4.5
- "dual": Dual Embedding, discussed in Section 4.6
- "baigal": Bai and Galbraith's Embedding, being a small secret variant of standard- and dual-embedding, discussed in Section 4.7
- "arora-gb": Arora and Ge's algorithm based and Gröbner bases, not discussed in this thesis and not adapted to fixed numbers of samples in the implementation

The shorthand symbol "bkw" solely refers to Coded-BKW and its small secret variant. Decision-BKW and Search-BKW are not assigned an abbreviation and are not used by the main method estimate_lwe, because Coded-BKW is the newest version of these three. Nevertheless, the two excluded algorithms can be called separately via the function bkw, which is a convenience method for the functions bkw_search and bkw_decision, and its corresponding small secret variant bkw_small_secret.

5.1 Explanation of Usage and Example

In the following, we show an example, which explains how the software is used and what the output means. For this, we choose the parameters like Regev [38] as implemented by Albrecht et al. [1]. This method depends solely on n and results in a parameter set $n, \alpha = \frac{1}{\sqrt{2\pi n}\log^2 n}, q \approx n^2$. In the following, n is set to $n = 128$. Given this parameter set and a fixed number of samples $m = 256$, calling the estimator to calculate the costs for exhaustive search, Coded-BKW, using lattice reduction to distinguish, decoding, standard embedding and dual embedding in the non-small secret case looks like shown in Listing 1. The first two lines of Listing 1 define the parameters n, α, q and the number of samples. The third line does the actual call to the program and stores the calculated estimations in the variable costs.

```
1  sage: n,alpha,q = unpack_lwe(Regev(128))
2  sage: m = 256
3  sage: costs = estimate_lwe(n, alpha, q, samples=m, skip="arora-gb")
```

Listing 1: Basic example of calling the LWE-Estimator using the LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n}\log^2 n}$, $q \approx n^2$ and $m = 256$

To exclude more algorithms, the parameter skip has to be extended with "," being the delimiter. To calculate the costs of the same algorithms like above in the small secret case, parameters small=True and secret_bounds=[lower,upper] have to be specified, with lower and upper being the lower and upper bound of the secret, respectively, for example secret_bounds=[−1,1]. As mentioned above, the functions bkw_search, bkw_decision and bkw_small_secret have to be called separately. In Listing 2 we present an example of calling these separate functions. For this, we choose $m = 2^{87.3}$ because the BKW algorithm in any variant requires a huge amount of samples.

```
sage: n, alpha, q = unpack_lwe(Regev(128))
sage: m = 2**87.3
sage: cost = bkw_decision(n, alpha, q, samples=m)
```

Listing 2: Example of calling the estimation for Decision-BKW separately for the LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n \log^2 n}}$, $q \approx n^2$ and $m = 2^{87.3}$

As one can see, both ways of calling functions of the estimator need the parameters of the LWE instance n, alpha, q and (optional) the fixed number of samples samples. If instead of alpha only the Gaussian width parameter (sigma_is_stddev=False) or the standard deviation (sigma_is_stddev=True) is known, alpha can be calculated by calling alphaf(sigma, q, sigma_is_stddev). The function alphaf basically calculates sigma / q, but in case of sigma being the standard deviation the factor $\sqrt{2\pi}$ is considered.

The result of Listing 1 is a dictionary (in other programming languages named 'map' or 'table') with the abbreviations associated to the algorithms as keys and a dictionary holding the results of the cost calculation of the respective algorithm as value. So, for example, the cost of standard embedding can be printed by the statement cost_str(costs["kannan"]) and looks like shown in Listing 3. In contrast, the calculation of the costs of Decision-BKW in Listing 2 returns the cost-dictionary directly, so that the results can be printed by the statement cost_str(cost).

```
sage: print cost_str(costs["kannan"])
______________________________________
sieve:  ≈2^77.1, oracle:      256, delta_0: 1.0071049, bkz2:  ≈2^91.4, k:      164, lp:  ≈
    2^102.8, m:      256
```

Listing 3: Example of outputting the cost of the standard embedding calculated by the call shown in the basic example (Listing 1)

Depending on the algorithm, the minimum number of required operations may be given by bop, rop, sieve or bkz2. However, the list in the output is always sorted such that the first value shows the minimum number of required operations. Instead of bit-operations, rop measures the number of required operations in $\mathbb{Z}_q$. Usually, this is an acceptable lower bound, since the number of bit-operations can be approximated by bop $\approx \log q \cdot$ rop. In Table 10 we show the meanings of all parts, which may occur in the output. To show a full example with output and an example of a small secret variant, we present the estimation of Bai and Galbraith's embedding in Listing 4 with the same parameters as above. Again, as before, first the parameters are defined, then the estimations of the costs are calculated and at the end these costs are printed.

```
sage: n, alpha, q = unpack_lwe(Regev(128))
sage: m = 256
sage: costs = estimate_lwe(n, alpha, q, samples=m, small=True, secret_bounds=[-1,1], skip="mitm,bkw,sis
    ,dec,kannan,dual,arora-gb")
sage: print cost_str(costs["baigal"])
______________________________________
bkz2:  ≈2^40.2, oracle:      256, delta_0: 1.0111634, k:       65, lp:  ≈2^38.9, sieve:  ≈
    2^59.1, repeat:        1
```

Listing 4: Example of Bai and Galbraith's embedding algorithm solving the small secret LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n \log^2 n}}$, $q \approx n^2$, $m = 256$ and $[a, b] = [-1, 1]$

5.2 Explanation of Structure of Code

Our Figures 2 and 3 show the high-level structure of the implementation in the general and small secret cases, respectively, and especially illustrate the connections of algorithms and subroutines. We show both entry points estimate_lwe and bkw with its small secret variant bkw_small_secret. We illustrate, which functions are called if the corresponding abbreviation is specified using normal arrows, while dashed arrows represent the usage of a subroutine and double arrows indicate, that one function is the wrapper function of another.

Ignoring the separate functions bkw and bkw_small_secret for the moment, everything starts with estimate_lwe. There the decision is made, whether the general or small secret implementation has to be used, based on the input parameters. Then, the methods for every algorithm except those listed in parameter skip are called. The order, in which they are called, is only relevant for the order of the output, since at the end, an ordered dictionary containing the results for every considered algorithm is returned.

First, we describe the general case as shown in Figure 2. The functions for exhaustive search and Coded-BKW can be called directly, while the other functions belong to algorithms using lattice reduction. These are wrapped in the function sieve_or_enum, which takes such a function as input and ensures, that for the BKZ lattice reduction the SVP-oracle yielding the best runtime is chosen.

Value	Explanation
bop	total number of bit-operations of an attack
rop	total number of operations in $\mathbb{Z}_q$ of an attack
delta_0	root-Hermite factor δ_0 used by lattice reduction
k	block size used by BKZ
bkz2	number of operations of running BKZ2.0 with enumeration as SVP-oracle
sieve	number of operations of running BKZ2.0 with sieving as SVP-oracle
fplll	number of operations of running BKZ2.0 with fplll 4.0.4 as SVP-oracle
lp	number of operations of running BKZ2.0 as estimated by Lindner and Peikert
oracle	number of samples required to run an attack
m/dim	internal number of samples/dimension, e.g., the dimension the algorithm is run on in a single execution without repeating
repeat	number of times the algorithm has to be repeated, e.g., to achieve a target success probability
mem	storage requirements of elements in $\mathbb{Z}_q$
log(eps)	negative logarithm of the success probability eps of a single run
enum	number of decoding steps
enumop	number of (real-field-)operations needed to execute enum decoding steps
|v|	length of the vector $\mathbf{v}$, which is searched for in the SIS problem
b, t1, t2, l, d, γ, σ_set, ncod, ntop, ntest, σ_final, C0(gauss), C1(bkw), C2(coded), C3(guess), C4(test)	parameters and partial results of Coded-BKW, corresponding to the theoretical parameters mentioned in this work

Table 10: Meanings of the abbreviations and values in the output of the function estimate_lwe

The discussions of using lattice reduction to distinguish and the decoding approach both are based on a fixed success probability. As already shown in the respective sections, this has to be compensated by wrapping the presented algorithm in a function called rinse_and_repeat, which takes the function to wrap and the LWE instance parameters as input and finds the best success probability, i.e. the success probability for which the lowest bit hardness is found. Since the found success probability is usually smaller than the target success probability, it has to be amplified to the target success probability by repetitively executing the algorithm. The number of repetitions is calculated by function amplify, which takes a success probability of a single run and a target success probability as input and returns the number of trials needed to amplify the success probability of a single run to the target success probability. The result can then be used to multiply the computational costs accordingly. In contrast, the embedding algorithms kannan and dual use amplify directly, since for them, a closed formula for the number of needed repetitions exists.

Naturally, the methods of the lattice-based algorithms all rely on bkz_runtime_delta, which takes the target δ_0 and the dimension of the lattice as input and calculates the runtime of BKZ. Also, except for dual-embedding, they calculate the optimal dimension $m_{optimal}$ according to the formula given by Albrecht et al. [6] $m_{optimal} = \sqrt{\frac{n \log q}{\log \delta_0}}$, implemented in lattice_reduction_opt_m(n,q,delta).

Decision-BKW and Coded-BKW employ distinguish_required_m, which takes the standard deviation, q and the success probability as input, to calculate the required amount of samples to distinguish between sampling from the sample-oracle and sampling from uniform random. The function binary_search basically takes a function f and an interval as input and returns the minimum of f on the given interval if f is convex. This is used by Coded-BKW to vary several parameters to get the best runtime.

Exhaustive search (mitm) on the other hand is completely independent of subroutines. Also, mitm and bkw_coded are implemented to handle both the general and the small secret case and therefore, they appear in both Figure 2 and 3. The other algorithms handling the small secret case are wrapped in the function small_secret_guess, which, given a the function to wrap and the LWE instance parameters, finds the best ratio between guessing components of $\mathbf{s}$ in style of exhaustive search and executing the given algorithm with the then reduced dimension. The methods for the other small secret algorithms except for Bai and Galbraith's algorithm also use modulus switching and actually are named after the pattern [algorithm]_embedding_small_secret_mod_switch_and_guess, which is shortened in Figure 3. Bai and

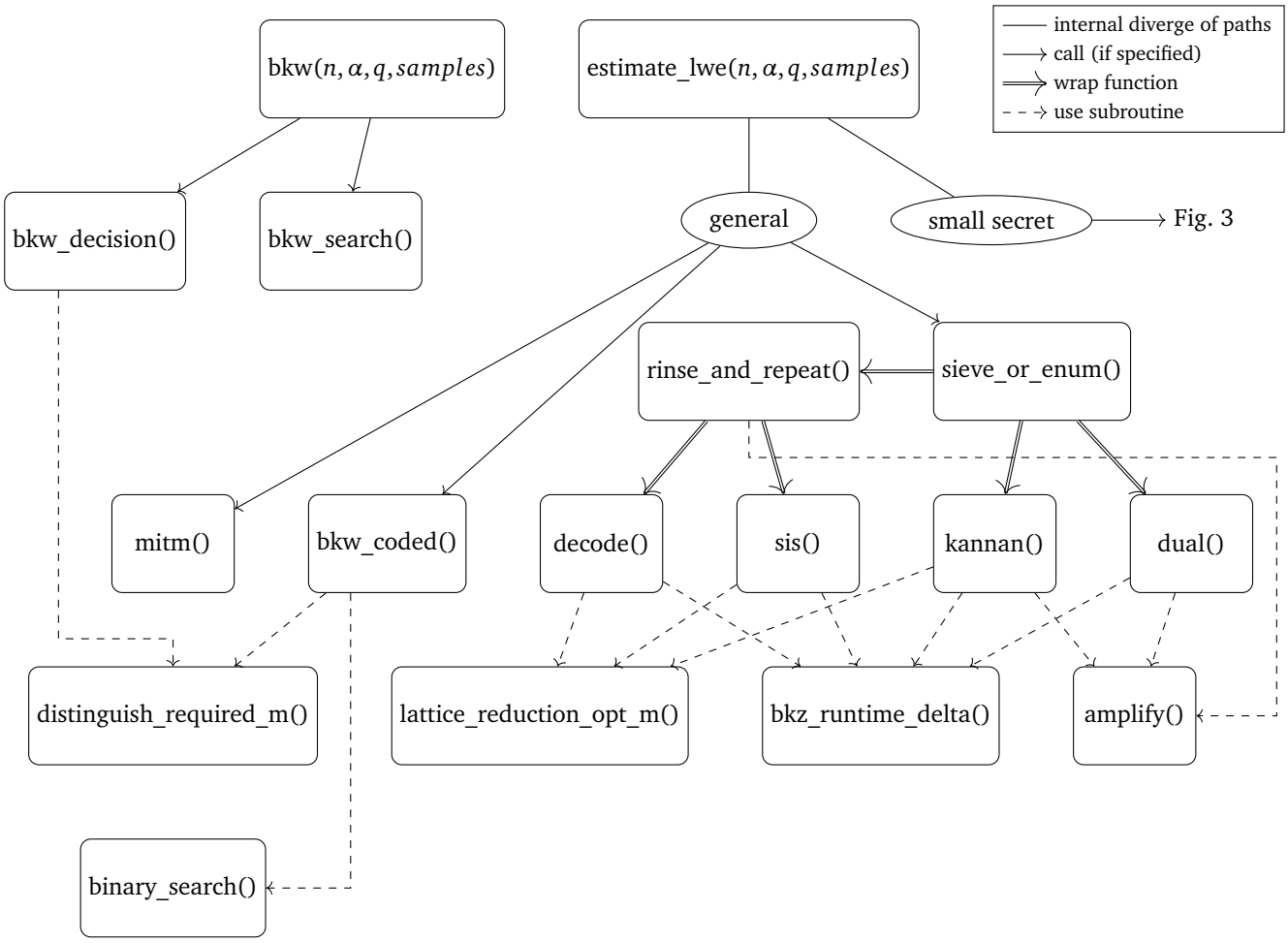


Figure 2: High-level structure of the implementation in the general case showing the connections of wrapping functions, subroutines and the functions estimating the costs of algorithms; estimate_lwe and bkw are the entry points for the general case

Galbraith's algorithm however is implemented similar to the other embedding algorithms, except, that it is wrapped in small_secret_guess as explained above.

The other entry point via the separate functions bkw and bkw_small_secret only employ distinguish_required_m like Coded-BKW or do not depend on any subroutine.

Implementation Details

For most of the specific methods implementing the estimation of the considered algorithms, the changes needed to adapt the previous implementation to take the number of samples into account are exact implementations of the results presented in the theory part of this work. Because of the amount of code, we do not present the implementation as a whole. However, there are some implementation details worth to mention.

In the method mitm, which corresponds to exhaustive search, the estimations of runtime and memory requirements were too simplified to the point, where they were independent of m. Together with a missing log in one of the constraint equations, this is fixed in the current version as suggested by us.

All implementations for the algorithms based on lattice reduction follow the same basic structure except for dual-embedding. In Figure 4a we show this basic structure as a flowchart. First, δ_0 and $m_{optimal}$ are calculated as before, then it is tested, if enough samples are available to use the optimal case. If not, these values are replaced by the results presented in Section 4 of this work. At the end, the computational cost for running this algorithm are calculated by calling bkz_runtime_delta. Our implementation of dual-embedding is slightly different. In Figure 4b we present the corresponding flowchart in contrast to the basic structure of the other algorithms. Since the previous implementation does not consider this algorithm, we provide a complete implementation of dual-embedding including the case of having the optimal number of samples available. For this, we use only the equation for δ_0 presented in this work and determine the maximum of δ_0 numerically to compute the optimal case. This yields the minimum runtime. Then, the check for enough samples and calculation of computational cost remains the same as in the other embedding attacks.

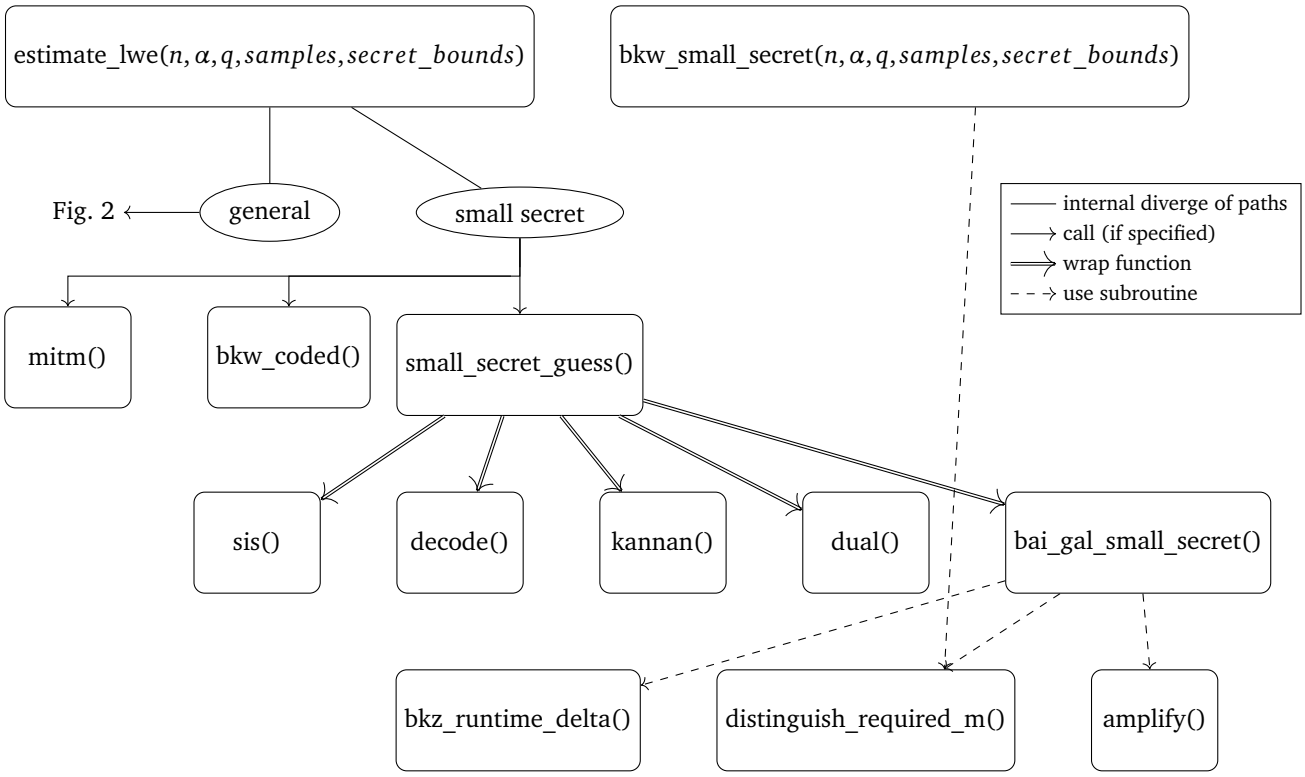


Figure 3: High-level structure of the implementation in the small secret case showing the connections of wrapping functions, subroutines and the functions estimating the costs of algorithms; estimate_lwe with set parameter secret_bounds and bkw_small_secret are the entry points for the small secret case

Another problem arises for decode when very strictly limiting the number of samples. It uses enum_cost to calculate the computational cost of the decoding step. For this, amongst other things, the stretching factors d_i of the parallelepiped are computed iteratively by step-wise increasing as described in Section 4.4. In this process, the success probability is used, which is calculated as a product of terms $\operatorname{erf}\left(\frac{d_i \left\|\mathbf{b}_i^*\right\| \sqrt{\pi}}{2\alpha q}\right)$ as described in Equation (57) of that section. Since the precision, the program can work with, is limited, this may lead to a success probability of 0, if one of these terms is close to 0. In this case, the loop never terminates. This problem can be avoided, but for reasonable step sizes doing so leads to unacceptable long runtimes of the loop finding the factors d_i. Since it is only an extreme case, where very few samples are available, we let the program throw an error, saying there are too few samples.

The function rinse_and_repeat is implemented by iteratively varying the success probability in a kind of numeric method, where one walks, starting from highest value, in the current direction with a pre-defined step size, until the results become worse; then one turns around with half step size. After the adaptation to a fixed number of samples, the problem arises, that there might be not enough samples to apply the algorithm at all. In our implementation, these exceptional cases, where an algorithm reports to not have enough samples available, are handled by estimating the costs for this set of parameters to be positive infinite. This may fail to find a solution, if none of the tested success probabilities result in a solvable parameter set. Like this, any algorithm shown will produce an exception when given too few samples, for example due to the small number of samples leading to a $\delta_0 < 1$.

Also, limiting the number of samples requires values δ_0 very near to 1 much more often than having available as many samples as needed, especially when varying success probabilities. The previous implementation k_chen however finds the value k for BKZ lattice reduction by iterating through possible values of k, starting at 40, until the resulting δ_0 is lower than the given target-δ_0. As shown in Listing 5, this iteration uses steps of multiplying k by 2 at maximum. When given a target-δ_0 very near to 1, only a high value k can satisfy the equation, which needs a long time to find. Therefore, the previous implementation of finding k for BKZ lattice reduction is not suitable for the case of limited numbers of samples. Thus, we replace this function in our implementation by finding k using the secant-method as presented in Listing 6.

```
f = lambda k: (k/(2*pi_r*e_r) * (pi_r*k)**(1/k))**(1/(2*(k-1)))
while f(2*k) > delta:
  k *= 2
while f(k+10) > delta:
  k += 10
  while True:
    if f(k) < delta:
      break
    k += 1
```

Listing 5: Iteration to find k in method k_chen of the previous implementation used in the LWE-Estimator

```
f = lambda k: RR((k/(2*pi_r*e_r) * (pi_r*k)**(1/k))**(1/(2*(k-1))) - delta)
k = newton(f, 100, fprime=None, args=(), tol=1.48e-08, maxiter=500)
k = ceil(k)
```

Listing 6: Implementation of method k_chen to find k using the secant-method

Many algorithms in the small secret case can be improved by guessing g components of n at first and then applying the respective algorithm on the smaller instance as described in Section 2.2.1. Similar to varying success probabilities, this is done by iteratively searching through varying values g until the optimal runtime is found. The previous implementation small_secret_guess evaluates the function calculating the costs for the respective algorithm at values i, such that this function is called with dimension $n-i$. The value i for the next step is determined based on a comparison between the costs of the current run and the best run until this point. This often leads to a wrong evaluation order and thus to not using the best g. The correct implementation would require a comparison depending on the last run instead of the best, which results in a form of binary search. It is preferable to consider this in future implementations of the LWE-Estimator.

5.3 Comparison of Implementations and Algorithms

In the following, we present results of all algorithms, evaluated using the example parameters from above. For comparison, we show the estimations of the implementation without considering a limitation of the number of samples, too. Furthermore, we compare the considered algorithms with focus on the behavior in case of a limitation of samples. This is done separately for the general variant and the small secret variant.

5.3.1 Comparison of Implementations for the General Variant

In our Tables 11a and 11b the logarithm of the hardness estimated by the implementation of this work can be seen. While exhaustive search, using lattice reduction to distinguish, decoding, standard-embedding and dual-embedding produce results on similar intervals, Coded-BKW needs approximately as many samples as in the optimal case to produce results and is therefore separated from the others. It can be seen, that the hardness decreases with increasing numbers of samples and remains the same after reaching the optimal number of samples, which amongst other values is shown in Table 12. If the estimator could not find a solution, mostly due to too few samples provided to apply the respective algorithm, the cell is filled with NaN.

samples	dual	mitm	sis	dec	kannan
0	NaN	NaN	NaN	NaN	NaN
50	NaN	NaN	NaN	NaN	NaN
100	103.24	NaN	NaN	NaN	NaN
150	67.91	NaN	NaN	NaN	NaN
200	60.32	395.89	163.59	132.63	282.94
250	58.54	395.89	97.34	75.01	80.62
300	58.54	395.89	79.54	61.8	64.53
350	58.54	395.89	75.13	58.08	59.57
400	58.54	395.89	74.74	58.07	59.14
450	58.54	395.89	74.74	58.07	59.14

(a)

samples	bkw
$1.18 \cdot 10^{21}$	NaN
$2.95 \cdot 10^{21}$	NaN
$4.72 \cdot 10^{21}$	NaN
$6.49 \cdot 10^{21}$	NaN
$8.26 \cdot 10^{21}$	NaN
$1 \cdot 10^{22}$	NaN
$1.18 \cdot 10^{22}$	88.87
$1.36 \cdot 10^{22}$	88.87
$1.53 \cdot 10^{22}$	88.87
$1.71 \cdot 10^{22}$	88.87

(b)

Table 11: Logarithmic hardness of the algorithms exhaustive search (mitm), Coded-BKW (bkw), using lattice reduction to distinguish (sis), decoding (dec), standard-embedding (kannan) and dual-embedding (dual) depending on the given number of samples for the LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n}\log^2 n}$ and $q \approx n^2$

In Table 12 we show the logarithmic hardness and the corresponding optimal number of samples estimated by the previous implementation for unlimited numbers of samples. It should be noted that, some algorithms rely on multiple

executions, for example to amplify the rather low success probability of a single run to a target success probability near to 1. In such a case, the previous implementation for optimal number of samples multiplies not only the runtime but also the number of oracle calls, while we assume for our adapted implementation for fixed numbers of samples, that samples may be reused in repeated runs of the same algorithm. Therefore, some of the optimal number of samples shown in column "optimal" in Table 12 are much bigger than estimated by the our implementation. To compensate this and to provide comparability, we introduce the column "optimal-recalculated", for which we recalculate the values for the optimal numbers of samples into the model of reusing samples, i.e. undo the amplification done in the first place.

Algorithm	optimal	optimal-recalculated	hardness
dual	10780	245	58.54
mitm	181	181	395.90
sis	192795128	376	74.74
dec	53436	366	58.07
kannan	16412	373	59.14
bkw	1.011798789e22	1.011798789e22	88.87

Table 12: Logarithmic hardness with optimal number of samples computed by the previous LWE-Estimator and the optimal number of samples recalculated according to the model used in this work for the LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n}\log^2 n}$ and $q \approx n^2$

We illustrate the shown results in the form of plots in Figures 5a to 5f. There is one algorithm per plot and the dotted line marks the number of optimal samples for the respective algorithm as specified in column "optimal-recalculated" of Table 12. Additionally, we show the hardness estimated by considering no limitation of the number of samples in each plot by "[algorithm]-optimal". As one can see, for numbers of samples lower than the optimal number of samples, the estimated hardness is either (much) bigger than the estimation using optimal number of samples or does not exist. In contrast, for numbers of samples greater or equal than optimal ($samples > optimal$), the hardness is exactly the same as in the optimal case, since the implementation falls back on the optimal number of samples when enough samples are given. Without this the hardness would increase again for $samples > optimal$. We show this behavior exemplary for dual-embedding in Figure 6.

5.3.2 Comparison of Algorithms for the General Variant

In Figure 7 we show the effect of limiting the available number of samples on the considered algorithms comparatively. We do not include BKW in this plot, since the number of required samples is too high (about 10^{22}) to show in the same graph as the other algorithms (about 300 samples). Instead, we present the results for Coded-BKW in Figure 5b. The first thing that strikes is, that the limitation of the number of samples leads to an exponentially increase of the logarithmic hardness, except for exhaustive search and Coded-BKW, which are basically not applicable for limited number of samples. Furthermore, while the algorithms labeled with mitm, sis, dec and kannan are applicable for roughly the same interval of samples, the algorithms dual-embedding and Coded-BKW stand out. That is, Coded-BKW needs a lot of samples; in fact, it is an other magnitude. Also, it does not allow a considerable limitation of the number of samples. Dual-embedding on the other hand is applicable for stronger limitations than any of the other algorithms. Though, it has to mentioned, that the optimal number of samples is also lower than that of the other algorithms. However, it is true, not only in absolute numbers, but even when setting the minimum number of samples, for which the algorithm is applicable, into relation to the optimal number. Also, the logarithmic hardness of dual-embedding is lower than for the other algorithms not only in absolute values, but in relation to the hardness in the optimal case as well. In other words, the logarithmic hardness of it increases slower than that of the other algorithms. This can be attributed to the usage of a lattice with dimension $n + m$, while only requiring m samples. The graphs of the algorithms labeled with sis and dec are nearly parallel in the sense of them having the same behavior in terms of increase of the hardness when limiting the number of samples. The logarithmic hardness of standard embedding (kannan) increases much more rapidly.

5.3.3 Comparison of Implementations for the Small Secret Variant

In Tables 13a and 13b we demonstrate the results of the logarithmic hardness in small secret case. For the same reason as above, Coded-BKW is shown in a separate table. Again, it can be seen, that with increasing numbers of samples the hardness decreases until the optimal number of samples is reached. From there, the hardness remains the same, since having more samples available than the optimal number allows to use the optimal runtime. Also, we present the results of the previous implementation with optimal numbers of samples in Table 14. As before, NaN means, the implementation returned no result and the column "optimal-recalculated" is introduced to compare the results of this

samples	baigal	dual	sis	kannan	mitm	dec
0	∞	NaN	NaN	NaN	NaN	NaN
50	67.15	120.85	NaN	NaN	NaN	NaN
100	37.26	71.41	149.59	143.28	NaN	NaN
150	34.57	55.79	118.92	106.97	117.33	NaN
200	34.57	52.27	100.75	85.76	117.33	NaN
250	34.57	52.31	79.05	63.97	117.33	59.89
300	34.57	52.31	69.8	53.82	117.33	53.13
350	34.57	52.31	68.47	53.01	117.33	53.05
400	34.57	52.31	68.47	53.01	117.33	53.05
450	34.57	52.31	68.47	53.01	117.33	53.05

(a)

samples	bkw
$4.4 \cdot 10^{12}$	NaN
$1.1 \cdot 10^{13}$	NaN
$1.76 \cdot 10^{13}$	NaN
$2.42 \cdot 10^{13}$	NaN
$3.08 \cdot 10^{13}$	NaN
$3.74 \cdot 10^{13}$	61.76
$4.4 \cdot 10^{13}$	61.76
$5.06 \cdot 10^{13}$	61.76
$5.72 \cdot 10^{13}$	61.76
$6.38 \cdot 10^{13}$	61.76

(b)

Table 13: Logarithmic hardness of the small secret variants of the algorithms exhaustive search (mitm), Coded-BKW (bkw), using lattice reduction to distinguish (sis), decoding (dec), standard-embedding (kannan), dual-embedding (dual) and Bai and Galbraith's embedding (baigal) depending on the given number of samples for the LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n}\log^2 n}$, $q \approx n^2$ and $[a, b] = [-1, 1]$

work and the estimations of the optimal case produced by the previous implementation, i.e. recalculated into the model, where samples are reused in each run. Also, we choose the parameters Regev [38] as implemented by Albrecht et al. [1] with $n = 128$ as above and we set the interval of the secret bounds as $[-1, 1]$.

Algorithm	optimal	optimal-recalculated	hardness
baigal	6600	150	34.57
dual	9020	205	52.31
sis	172285008	336	68.47
kannan	14652	333	53.01
mitm	145	145	117.33
dec	23760	330	53.05
bkw	3.314881294e13	3.314881294e13	61.76

Table 14: Logarithmic hardness in the small secret case with optimal number of samples computed by the previous LWE-Estimator and the optimal number of samples recalculated according to the model used in this work for the LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n}\log^2 n}$, $q \approx n^2$ and $[a, b] = [-1, 1]$

To illustrate results, we show plots for the small secret variants of shown algorithms in a similar way as above for the general case in Figure 8. The graph for Coded-BKW is omitted and instead, we present it in Figure 9b, since for the comparison, it cannot be shown in the same plot as the other algorithms due to the huge difference in numbers of samples. The graphs look similar to the ones presented in Figure 5. That is, exhaustive search (mitm) and Coded-BKW only reflect the results of the optimal case; the other algorithms show an exponentially increase of the logarithmic hardness, when limiting the number of samples, and a constant behavior, if more samples are available than the optimal number of samples.

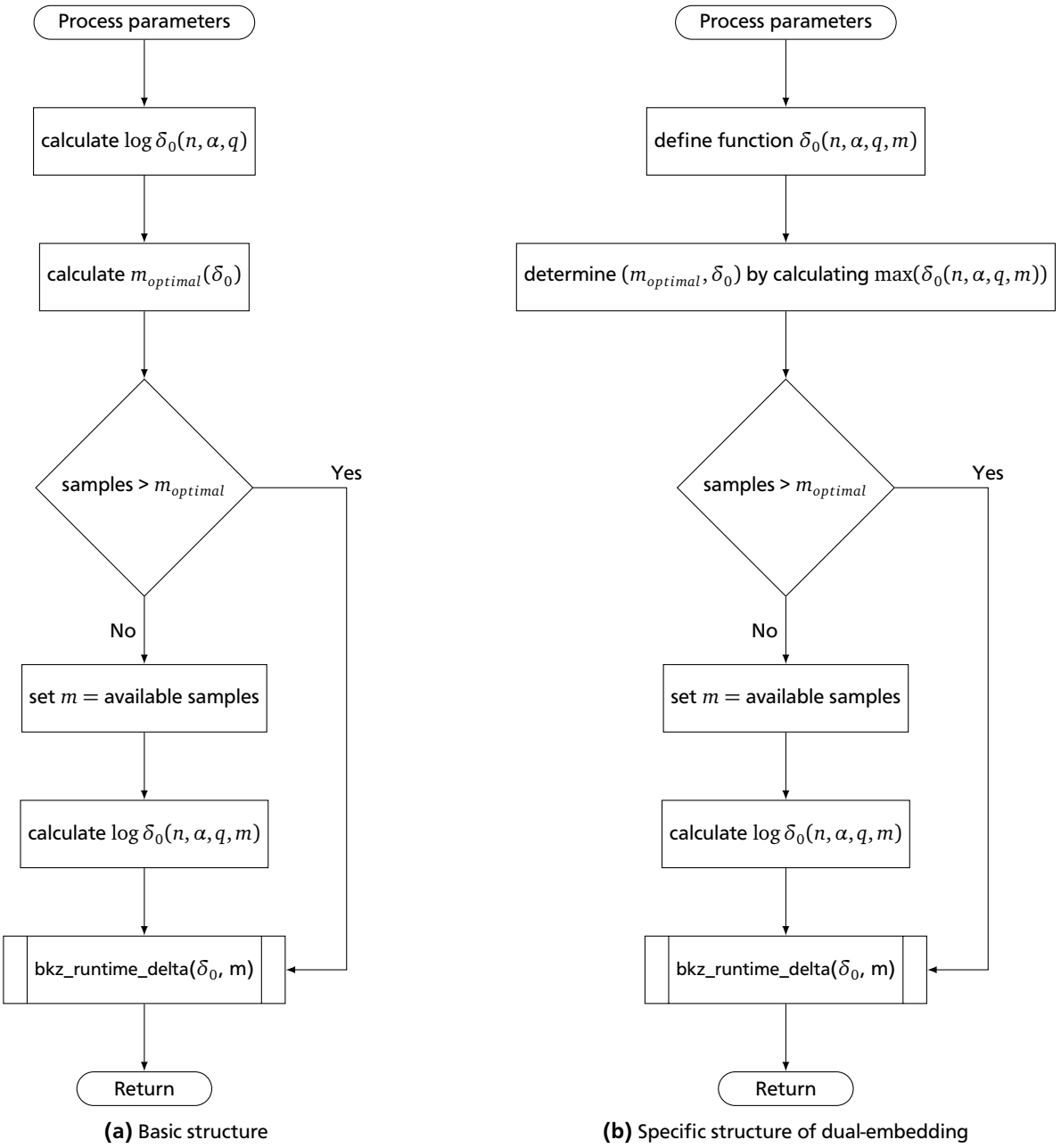
5.3.4 Comparison of Algorithms for the Small Secret Variant

In Figure 9 we show a comparison of the considered algorithms in the small secret case with a fixed number of available samples. As one can see, Bai and Galbraith's algorithm yields the best result for limited number of samples. However, dual-embedding is very similar in terms of behavior of the increase of the hardness. Again, like in the general case, standard embedding (kannan) shows the biggest percentage increase. For a certain limitation of the samples, the hardness of exhaustive search (mitm) becomes as good or even better than some of the algorithms, for example here, the hardness of the algorithm labeled with sis at 150 samples is higher than the hardness of exhaustive search. This shows, that the method of finding the best ratio between guessing components of $\mathbf{s}$ and actually applying the algorithm implemented in small_secret_guess is wrong as described above. In theory, this procedure ensures, that the algorithms have at most the costs of exhaustive search, but this example shows, that the actual implementation does not fulfill this requirement.

5.3.5 Conclusion of Comparisons

Comparing the general and the small secret cases with respect to the hardness when limiting the number of samples shows, that, in general, for the same parameters, a stronger limitation of the samples is possible for the small secret case than for the general case while the algorithms still remain applicable. Also, while the logarithmic hardness in the small secret case is lower than in the general case as expected, the increasing behavior roughly remains the same.

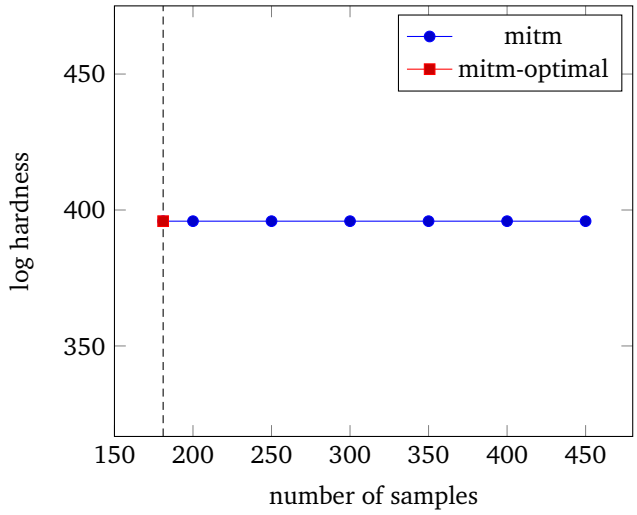
Also, it can be seen, that the implementation adapted to consider fixed number of samples is consistent in comparison to the previous implementation, since calling the estimator, setting the optimal number of samples as fixed number of samples, produces the same hardness as in case of not limiting the number of samples. Furthermore, the results show, that the optimal number of samples indeed yield the minimal number of operations.



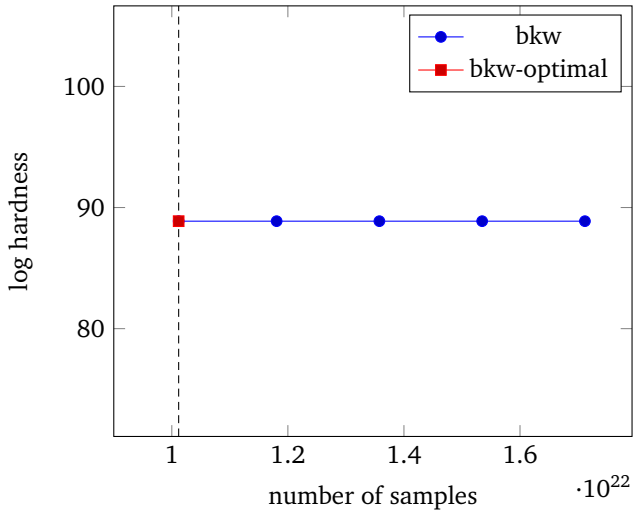
(a) Basic structure

(b) Specific structure of dual-embedding

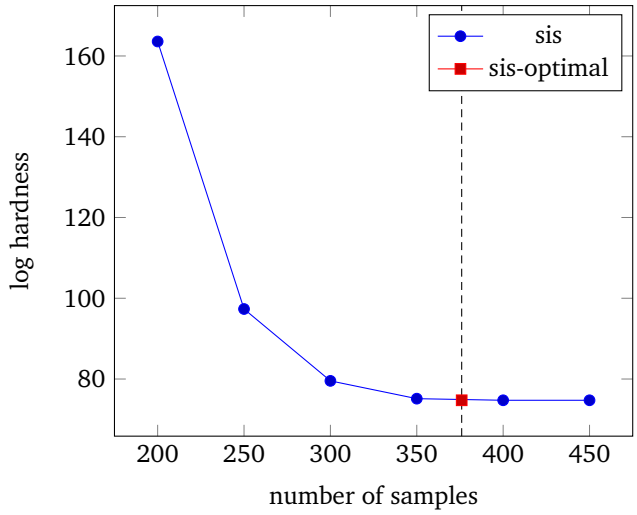
Figure 4: Flowcharts of the basic structure of estimating the computational cost of algorithms using lattice reduction and the specific structure of estimating the cost of dual-embedding



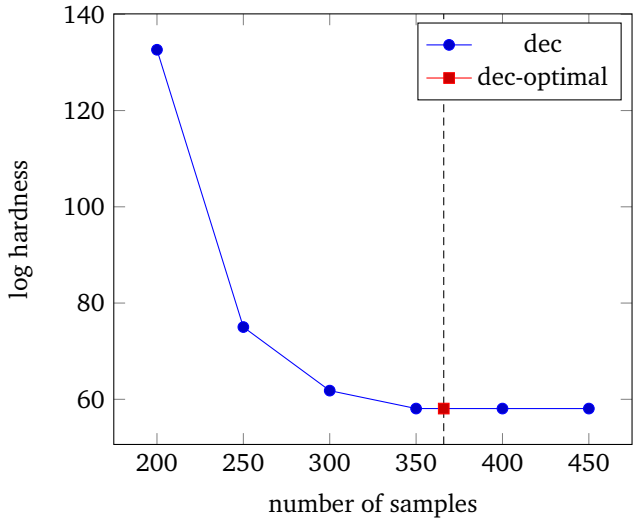
(a) Meet-in-the-middle (mitm)



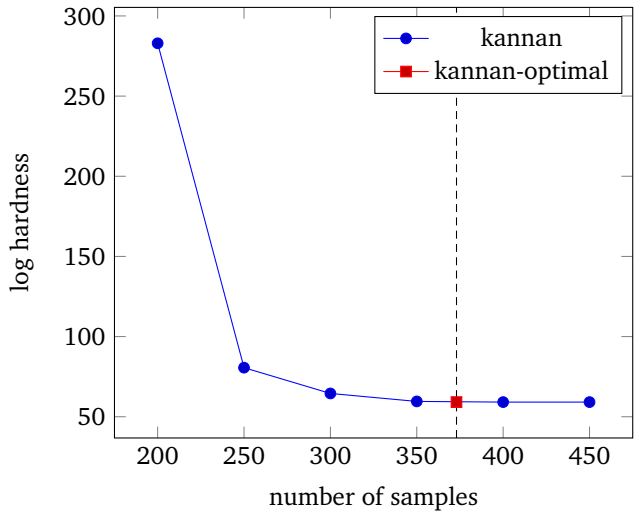
(b) BKW (bkw)



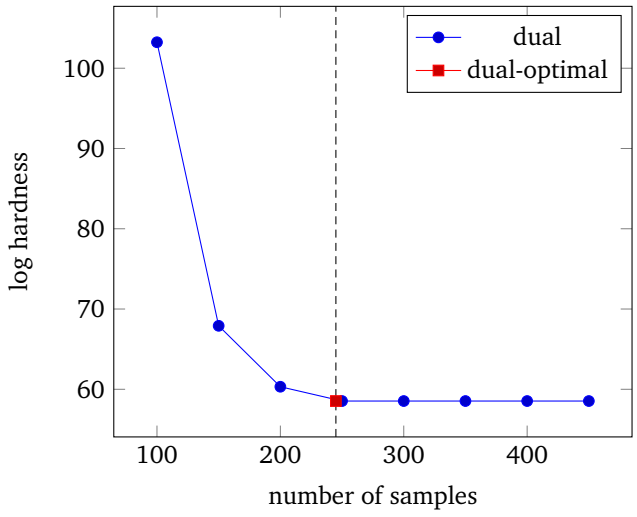
(c) using lattice reduction to distinguish (sis)



(d) decoding (dec)



(e) standard-embedding (kannan)



(f) dual-embedding (dual)

Figure 5: Logarithmic hardness of the algorithms Meet-in-the-middle, Coded-BKW, using lattice reduction to distinguish, decoding, standard embedding and dual embedding for the LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n \log^2 n}}$ and $q \approx n^2$; hardness estimations for each algorithm shown for both depending on a given number of samples and using the optimal number of samples marked by a dashed line

samples	dual
0	NaN
50	NaN
100	103.24
150	67.91
200	60.32
250	58.57
300	60.38
350	64.17
400	66.99
450	70.37

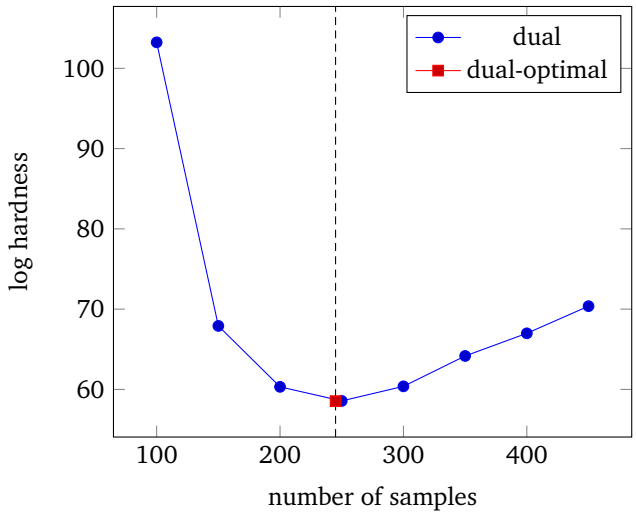


Figure 6: Logarithmic hardness of dual-embedding without falling back to optimal case for numbers of samples larger than the optimal number of samples for the LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n}\log^2 n}$ and $q \approx n^2$

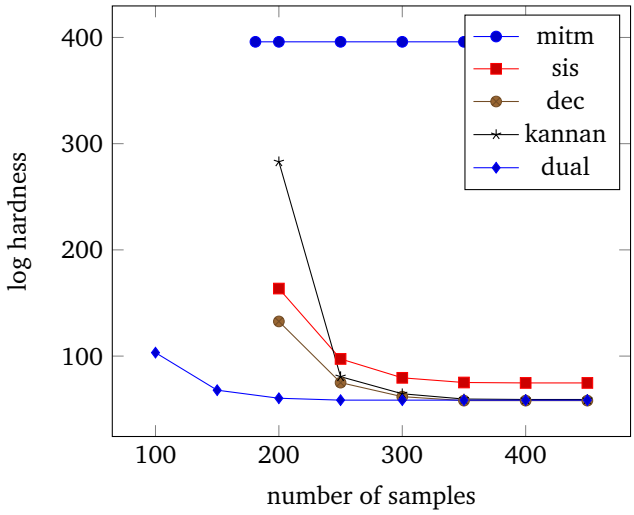
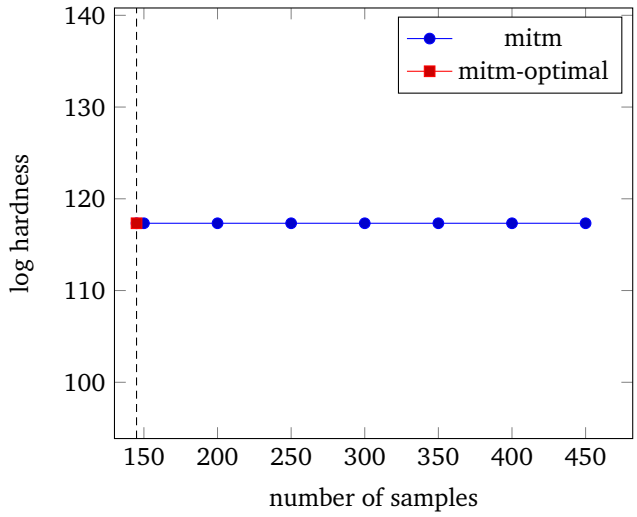
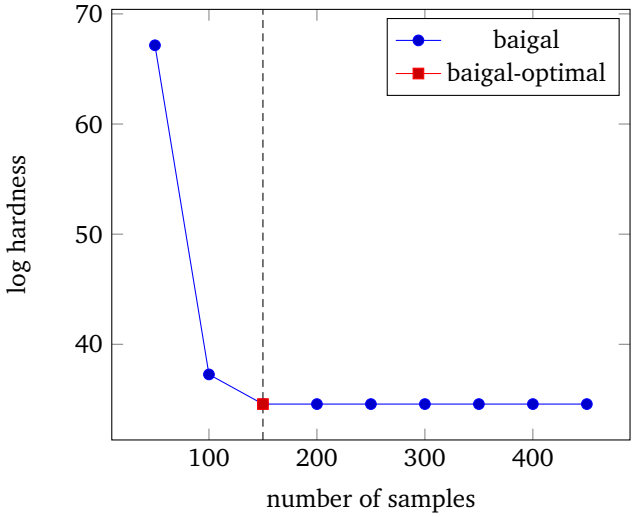


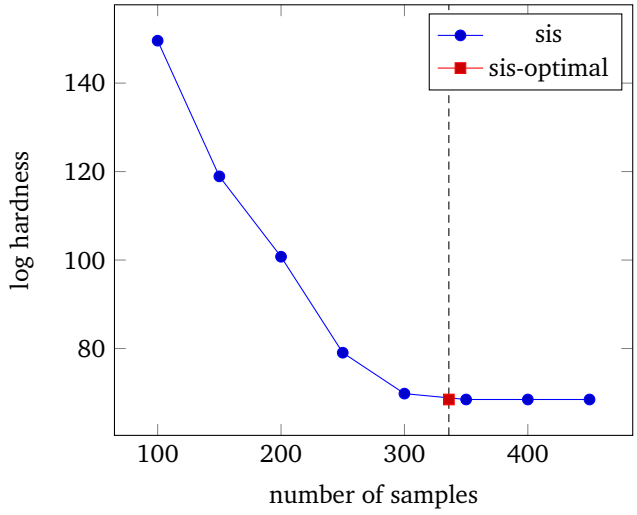
Figure 7: Comparison of the logarithmic hardness of the LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n}\log^2 n}$ and $q \approx n^2$ of the algorithms Meet-in-the-middle (mitm), using lattice reduction to distinguish (sis), decoding (dec), standard embedding (kannan) and dual embedding (dual), when limiting the number of samples



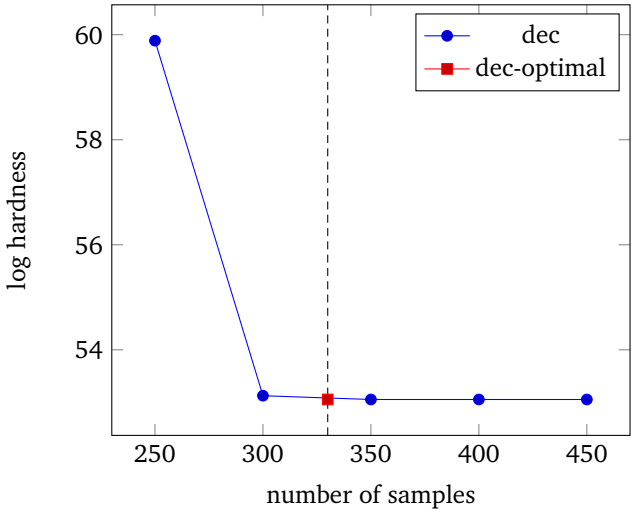
(a) Meet-in-the-Middle (mitm)



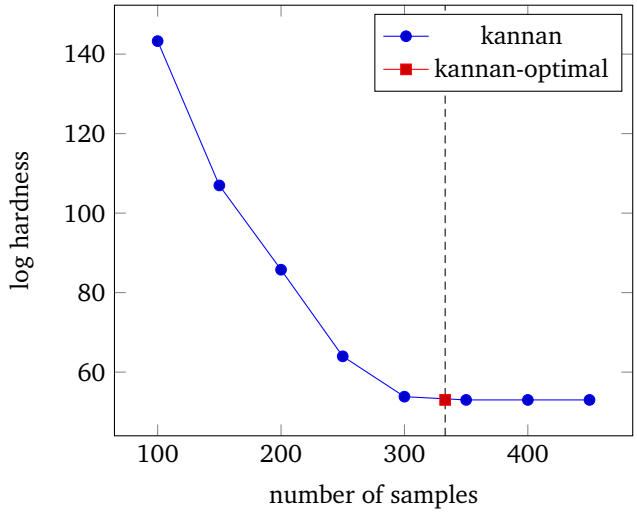
(b) Bai and Galbraith's algorithm (baigal)



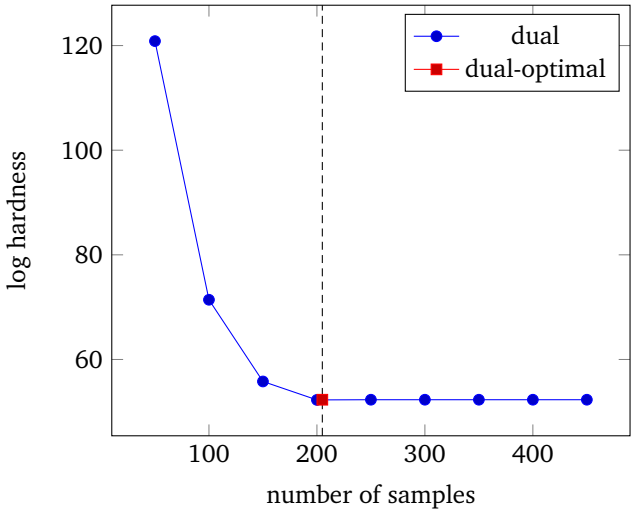
(c) using lattice reduction to distinguish (sis)



(d) decoding (dec)

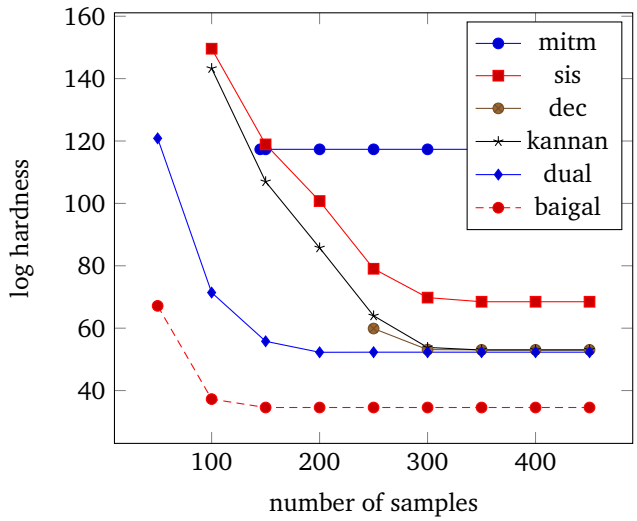


(e) standard-embedding (kannan)

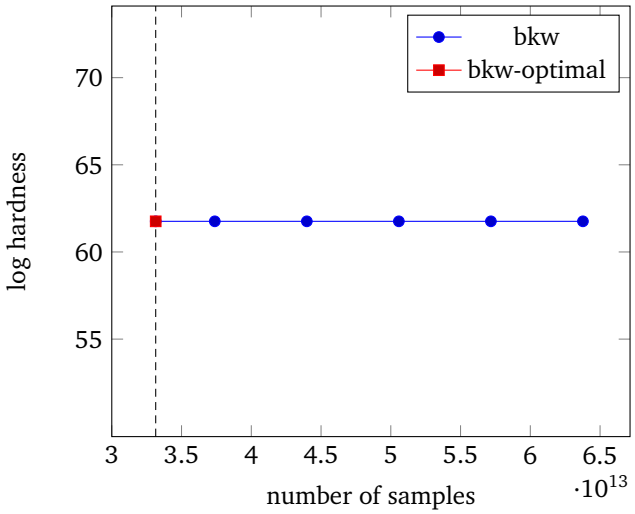


(f) dual-embedding (dual)

Figure 8: Logarithmic hardness of the algorithms Meet-in-the-middle, Bai-Galbraith-embedding, using lattice reduction to distinguish, decoding, standard embedding and dual embedding for the small secret LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n \log^2 n}}$, $q \approx n^2$ and $[a, b] = [-1, 1]$; hardness estimations for each algorithm shown for both depending on a given number of samples and using the optimal number of samples marked by a dashed line



(a) Comparison of algorithms except for BKW



(b) Graph for Coded-BKW including the optimal case

Figure 9: Comparison of the logarithmic hardness of the LWE instance $n = 128$, $\alpha = \frac{1}{\sqrt{2\pi n}\log^2 n}$, $q \approx n^2$ and $[a, b] = [-1, 1]$ of the small secret variants of the algorithms Meet-in-the-middle (mitm), Coded-BKW (bkw), using lattice reduction to distinguish (sis), decoding (dec), standard embedding (kannan), dual embedding (dual) and Bai and Galbraith's embedding (baigal) when limiting the number of samples

6 Summary

In this work, we present an analysis of the hardness of LWE for the case of a fixed given number of samples. For this, we describe the approaches exhaustive search, BKW, using lattice reduction to distinguish, decoding, standard embedding and dual embedding shortly and analyze them with regard to limiting the number of samples. Also, we analyze the short secret variants of the mentioned algorithms under the same restriction on samples. Furthermore, we present the dual embedding algorithm in case of using as many samples as required to run in optimal time. We adapt the existing "LWE-Estimator" software to take the results of these analyses into account. We show some of the necessary changes to the existing code. This includes direct changes to the methods calculating the hardness of the actual algorithm as well as slight adaptations to some helper functions. During this, we discovered some bugs in the previous software, namely a missing log, an imprecise estimation and a wrong evaluation order, some of which are fixed already as suggested by us. At the end, we present examples of using the software to show, how to use the estimator and what the several fields of the output mean. Also, we compare the results of the previous implementation always using the optimal number of samples and the implementation of this work, which takes a limitation of samples into account, based on example parameters. While the lattice reduction based algorithms decoding, using lattice reduction to distinguish, standard embedding and dual embedding show a similar behavior, the algorithms exhaustive search and BKW are somewhat different. The former group shows a hardness approaching the hardness of the optimal case, coming from positive infinity at small numbers of samples, as presented in the plots in Figures 5c to 5f. As mentioned, exhaustive search and BKW behave different for given fixed numbers of samples. That is, there are no results for the interval from zero to the optimal number of samples due to reasons discussed in the corresponding sections. So, the results show, that it is significantly harder or even impossible to use the considered attacks when using few number of samples, while using roughly as many samples as in the optimal case yields approximately the same hardness.

As mentioned above, the usage of a restricted set of samples has its limitations. If given too few samples, none of the presented algorithms are applicable and in case of exhaustive search and BKW, this applies even for roughly any limiting number of samples below the optimal number. Also, for very few samples, the runtime of the estimator may be prolonged significantly or the accuracy of the results may become imprecise. On the other hand, it is possible to construct an LWE instance without sample limit from an LWE instance with a given, fixed set of samples. For example, Duc, Tramér and Vaudenay [19] use an idea introduced by Lyubashevsky [32] to generate additional samples, at cost of them having higher noise, using an universal family of hash functions to combine samples. Other possibilities of accomplishing this are given in [22] and [39]. Further work could implement this into the estimator and compare the results to the estimations given by this work. This may lead to improvements of the estimation, especially for the algorithms exhaustive search and BKW.

The results of this work show the necessity of taking the number of samples into account, considering the sometimes huge impact on the hardness of the LWE-solving algorithms when limiting the number of available samples. The new estimations implemented into the LWE-Estimator help to prevent overly conservative parameter choices when working with cryptographic systems with limited numbers of samples.

References

[1] Martin R. Albrecht, Daniel Cabarcas, Robert Fitzpatrick, Florian Göpfert, and Michael Schneider. A generator for LWE and ring-LWE instances. available at: `http://www.iacr.org/news/files/2013-04-29lwe-generator.pdf`, 2013.

[2] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. Cryptology ePrint Archive, Report 2012/636, 2012. `http://eprint.iacr.org/2012/636`.

[3] Martin R. Albrecht, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Lazy modulus switching for the BKW algorithm on LWE. Cryptology ePrint Archive, Report 2014/019, 2014. `http://eprint.iacr.org/2014/019`.

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

[5] Martin R. Albrecht, Florian Göpfert, Cedric Lefebvre, Rachel Player, and Sam Scott. Estimator for the bit security of LWE instances. `https://bitbucket.org/malb/lwe-estimator`, 2016. [Online; accessed 18-November-2016].

[6] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. Cryptology ePrint Archive, Report 2015/046, 2015. `http://eprint.iacr.org/2015/046`.

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

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

[9] László Babai. On lovász' lattice reduction and the nearest lattice point problem. In K. Mehlhorn, editor, *STACS 85: 2nd Annual Symposium on Theoretical Aspects of Computer Science Saarbrücken, January 3–5, 1985*, pages 13–20. Springer Berlin Heidelberg, Berlin, Heidelberg, 1985.

[10] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *Topics in Cryptology – CT-RSA 2014*, volume 8366 of *Lecture Notes in Computer Science*, pages 28–47, San Francisco, CA, USA, February 25–28, 2014. Springer, Heidelberg, Germany.

[11] Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In Willy Susilo and Yi Mu, editors, *ACISP 14: 19th Australasian Conference on Information Security and Privacy*, volume 8544 of *Lecture Notes in Computer Science*, pages 322–337, Wollongong, NSW, Australia, July 7–9, 2014. Springer, Heidelberg, Germany.

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

[13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 575–584, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[14] David Cadé, Xavier Pujol, and Damien Stehlé. fplll 4.0.4. `http://perso.ens-lyon.fr/damien.stehle/fplll/`, 2013. [Online; accessed 18-November-2016].

[15] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

[16] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statist.*, 23(4):493–507, 12 1952.

[17] Özgür Dagdelen, Rachid El Bansarkhani, Florian Göpfert, Tim Güneysu, Tobias Oder, Thomas Pöppelmann, Ana Helena Sánchez, and Peter Schwabe. High-speed signatures from standard lattices. In Diego F. Aranha and Alfred Menezes, editors, *Progress in Cryptology - LATINCRYPT 2014: 3rd International Conference on Cryptology and Information Security in Latin America*, volume 8895 of *Lecture Notes in Computer Science*, pages 84–103, Florianópolis, Brazil, September 17–19, 2015. Springer, Heidelberg, Germany.

[18] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.0.0)*, 2016. `http://www.sagemath.org`.

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

[20] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.

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

[22] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.

[23] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 1996.

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

[25] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In *Proceedings of the Third International Conference on Coding and Cryptology*, IWCC'11, pages 159–190, Berlin, Heidelberg, 2011. Springer-Verlag.

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

[27] H. W. Lenstra jr. Integer programming with a fixed number of variables. *MATH. OPER. RES*, 8(4):538–548, 1983.

[28] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. Cryptology ePrint Archive, Report 2014/744, 2014. `http://eprint.iacr.org/2014/744`.

[29] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. Cryptology ePrint Archive, Report 2014/907, 2014. `http://eprint.iacr.org/2014/907`.

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

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

[32] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques: 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th International Workshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005. Proceedings*, pages 378–389. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[33] Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 577–594, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[34] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 21–39, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[35] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[36] Phong Q. Nguyen and Damien Stehlé. Floating-point LLL revisited. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 215–233, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

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

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

[39] Oded Regev. The learning with errors problem (invited survey). In *Proceedings of the 2010 IEEE 25th Annual Conference on Computational Complexity*, CCC '10, pages 191–204, Washington, DC, USA, 2010. IEEE Computer Society.

[40] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1):181–199, 1994.

[41] Claus Peter Schnorr. Lattice reduction by random sampling and birthday methods. In Helmut Alt and Michel Habib, editors, *STACS 2003: 20th Annual Symposium on Theoretical Aspects of Computer Science Berlin, Germany, February 27 – March 1, 2003 Proceedings*, pages 145–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.