Quantum Augmented Dual Attack

Martin R. Albrecht and Yixin Shen

Royal Holloway, University of London.
{martin.albrecht,yixin.shen}@rhul.ac.uk

Abstract. We present a quantum augmented variant of the dual lattice attack on the Learning with Errors (LWE) problem, using classical memory with quantum random access (QRACM). Applying our results to lattice parameters from the literature, we find that our algorithm outperforms previous algorithms, assuming unit cost access to a QRACM. On a technical level, we show how to obtain a quantum speedup on the search for Fast Fourier Transform (FFT) coefficients above a given threshold by leveraging the relative sparseness of the FFT and using quantum amplitude estimation. We also discuss the applicability of the Quantum Fourier Transform in this context. Furthermore, we give a more rigorous analysis of the classical and quantum expected complexity of guessing part of the secret vector where coefficients follow a discrete Gaussian (mod q).

Keywords: Learning with Errors, Dual attack, Fast Fourier Transform, Quantum algorithms, Amplitude Estimation

1 Introduction

The Learning With Errors (LWE) problem was introduced by Regev [Reg05] and has since become a major ingredient for constructing basic and more advanced cryptographic primitives. It asks to find $\mathbf{s}$ given $(\mathbf{A}, \mathbf{b})$ with $\mathbf{b} \equiv \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \bmod q$ where both $\mathbf{s}$ and $\mathbf{e}$ have small entries. Its conjectured hardness against quantum computers further makes all these constructions supposedly post-quantum. In NIST's Post Quantum Standardization Process, three of the six finalists rely on the conjectured hardness of algebraic variants of Learning With Errors [SSTX09,LPR10] and its variant Learning With Rounding (LWR) [BPR12] problem.

From the perspective of a cryptanalyst equipped with a quantum computer, lattice problems such as LWE are frustrating. Known quantum speedups to solving these problems are tenuous at best [AGPS20]. That is, while Grover's search offers a near quadratic quantum speedup for breaking, say, AES [JNRV20] the gains against lattice problems are significantly more modest. This is due to the rich structure of the search space in a lattice reduction algorithm that has given rise to refined structured search algorithms for these problems, e.g. [BDGL16]. As a consequence of this, the current state-of-the-art is that quantum algorithms on lattice problems can effectively be ignored when setting parameters.

The most efficient cryptanalysis techniques against LWE(-like) problems are "primal" and "dual" lattice attacks, named after whether lattice reduction is performed on the "primal" lattice related to $\mathbf{A}$ or the "dual" lattice related to $\{\mathbf{x} \in \mathbb{Z}_q^m \mid \mathbf{x} \cdot \mathbf{A} \equiv \mathbf{0} \bmod q\}$. Up until recently, dual attacks were generally considered less efficient for secrets $\mathbf{s}$ drawn from a sufficiently wide distribution. Recent developments [GJ21,MAT22] of dual attacks, however, have shown their ability to surpass primal attacks. These performance improvements are derived from combining lattice reduction on the scaled dual of a target lattice with an exhaustive search on a space related to the underlying secret $\mathbf{s}$. Roughly speaking, spending more resources on the exhaustive search part allows us to spend fewer resources on the lattice reduction part of the overall algorithm and vice versa.

In [GJ21] the search over part of the secret vector is realised using a Fast Fourier Transform style algorithm and the search space is significantly reduced by roughly considering only the most significant bits of this part of the secret. In [MAT22] this last step is replaced by "modulus switching" which further provides significant performance gains. These newer iterations of the dual attack relate the search space to the underlying secret in such a way that large dimensions can now be covered even when the norm of the secret vector is not very small (previous versions of the dual attack relied on, say, coefficients $\mathbf{s}_i \in \{-1, 0, 1\}$).

Thus, with this new generation of dual attacks, unstructured search starts again to play a bigger role in costing attacks on LWE. It is therefore natural to ask what performance gains can be obtained by tackling this unstructured search using a Grover-like quantum algorithm. More precisely, [MAT22] relies on two different kinds of unstructured search:

- Secret guessing: part of the secret is exhaustively searched until a match is found. Since the secret is generated according to a discrete Gaussian of small width, a significant speedup can be obtained by starting the search with the most likely values of the secret first. The expected complexity of this step is known as the guessing complexity.
- FFT threshold: given a list of values in a n-dimensional array, and a threshold, the problem is to decide whether one of the coefficients of the Fourier transform of the array is above the threshold. This problem arises when trying to determine whether the secret guess was correct by distinguishing between a uniform distribution and a gaussian one.

Contributions. After some preliminaries in Section 2, we provide a quantum version of the dual attack of [MAT22]. Specifically, our improvements are twofold.

First, in Section 3 we show how to obtain a quantum speedup on the search for Fast Fourier Transform (FFT) coefficients above a given threshold. This was left as an open problem in [MAT22]. Here, we leverage the relative sparseness of the FFT and use amplitude estimation to estimate the Fourier coefficients.

Second, in Section 4 we give a more rigorous analysis of the (classical and quantum) expected complexity of guessing a vector (whose coefficients are) drawn from a modular discrete Gaussian. In [MAT22], the authors estimated this complexity as the exponential of the entropy which is known not to be correct in general [Mas94]. We show that this complexity is indeed related to the entropy in the case of a (modular) discrete Gaussian, albeit up to a non-negligible exponential factor in the dimension.

In Section 5, we then measure the impact of our algorithm on the cost of solving lattice parameters from the literature. Following the literature, we evaluate the complexity of our algorithm under the assumption of unit-cost access to a classical memory with quantum random access (QRACM). Our algorithm provides a significant performance improvement over previous work in this model, but does not achieve a quadratic speedup overall. As a consequence, security parameters do not need to be updated in response to our findings.

In Section 6, we discuss the FFT threshold problem and its quantum complexity. Any significant speedup on this problem would yield major improvements in the complexity of the dual attack. We argue that the Quantum Fourier Transform (QFT) does not seem applicable in this context, despite being the natural approach.

2 Preliminaries

Recall that $e^{ix} = \cos(x) + i\,\sin(x)$. We write $[x, y]$ for the interval $\{x, x + 1, \ldots, y\} \subset Z$. We denote matrices by bold uppercase letters, e.g. $\mathbf{A}$, and vectors by bold lowercase letters, e.g. $\mathbf{v}$. We treat vectors as column matrices. We write $\mathbf{v}^T$ for the transpose of $\mathbf{v}$.

2.1 Lattices

A lattice $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^d$. We can represent it as $\{\sum x_i \cdot \mathbf{b}_i | x_i \in \mathbb{Z}\}$ and where $\mathbf{b}_i$ are the columns of a matrix $\mathbf{B}$, we may write $\mathcal{L}(\mathbf{B})$. If $\mathbf{B}$ has full column rank, we call $\mathbf{B}$ a basis.

While the central object of this work, the dual attack, critically relies on lattice reduction, such as the BKZ algorithm, we mostly make blackbox use of these algorithms here. Thus, we refer the reader to e.g. [GJ21,MAT22] for details. In particular, the blackbox use we make of lattice reduction algorithms and, critically, lattice sieving algorithms is captured in Algorithm 1.

In Algorithm 1 the BKZ-β_0 call performs lattice reduction with parameter β_0 where the cost of the algorithm scales at least exponentially with β_0. The BKZ algorithm proceeds by making polynomially many calls to an SVP oracle. In this work, this oracle is instantiated using a lattice sieving algorithm which is also called explicitly in Algorithm 1 with parameter β_1. Such a sieving algorithm outputs $N_{\mathsf{sieve}}(\beta_1)$ many short vectors in the lattice $\mathcal{L}(\mathbf{B})$ and has a cost exponential in β_1. The magnitude $N_{\mathsf{sieve}}(\beta_1)$ also grows exponentially with β_1 but slower than the cost of sieving. We may instantiate the lattice sieve with a classical algorithm [BDGL16] which has a cost of $2^{0.292\,\beta_1 + o(\beta_1)}$. We may also instantiate the lattice sieve with a quantum augmented variant of sieving [LMv13,Laa15,AGPS20,CL21] which as a cost of $2^{0.257\,\beta_1 + o(\beta_1)}$.

Algorithm 1: Short Vectors Sampling Procedure [GJ21]

```
  Input: A basis $\mathbf{B} = [\mathbf{b}_0 \ldots \mathbf{b}_{d-1}]$ for a lattice and integers $\beta_0, \beta_1 \leqslant d$ and $D$.
  Output: A list of $D$ vectors from the lattice.
1 Let $N_{\mathsf{sieve}}(\cdot)$ denote a function that returns the number of vectors returned by
    one call to a lattice sieving oracle.
2 for $i \in [0, \lceil D/N_{\mathsf{sieve}}(\beta_1)\rceil - 1]$ do
3     Randomise the basis $\mathbf{B}$.
4     Run BKZ-$\beta_0$ to obtain a reduced basis $\mathbf{b}'_0, \ldots, \mathbf{b}'_{d-1}$.
5     Run a sieve in dimension $\beta_2$ on the sublattice spanned by $\mathbf{b}'_0, \ldots, \mathbf{b}'_{\beta_1-1}$ to
        obtain a list of $N_{\mathsf{sieve}}(\beta_1)$ vectors and add them to $L$.
6 return $L$
```

2.2 Learning with Errors

The Learning with Errors problem (LWE) is defined as follows.

Definition 1 (LWE). *Let $n, m, q \in \mathbb{N}$, and let χ_s, χ_e be distributions over $\mathbb{Z}_q$. Denote by $\mathrm{LWE}_{n,m,\chi_s,\chi_e}$ the probability distribution on $\mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$ obtained by sampling the coordinates of the matrix $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ independently and uniformly over $\mathbb{Z}_q$, sampling the coordinates of $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{e} \in \mathbb{Z}_q^m$ independently from χ_s and χ_e respectively, and outputting $(\mathbf{A}, \mathbf{A}\cdot\mathbf{s}+\mathbf{e})$.*

We define two problems:

- Decision-LWE. Distinguish the uniform distribution over $\mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$ from $\mathrm{LWE}_{n,m,\chi_s,\chi_e}$.
- Search-LWE. Given a sample from $\mathrm{LWE}_{n,m,\chi_s,\chi_e}$, recover $\mathbf{s}$.

Dual Attack. Dual-lattice attacks, or simply "dual attacks" on LWE and related problems were introduced in [MR09]. In its simplest form it proceeds as follows. Given either $(\mathbf{A}, \mathbf{A}\cdot\mathbf{s}+\mathbf{e})$ or $\mathbf{A}, \mathbf{u}$ where $(\mathbf{A}, \mathbf{u})$ are uniform and wlog $\mathbf{s}, \mathbf{e}$ are short [ACPS09], the attack finds short $\mathbf{x}$ s.t. $\mathbf{x}^T \cdot \mathbf{A} \equiv \mathbf{0}$. Then, we either obtain $\mathbf{x}^T\cdot\mathbf{A}\cdot\mathbf{s} + \langle\mathbf{x},\mathbf{e}\rangle = \langle\mathbf{x},\mathbf{e}\rangle$ or $\langle\mathbf{x},\mathbf{u}\rangle$. The former follows a distribution with small elements, the latter follows a uniform distribution.

In [ADPS16], the "normal form" of the dual attack was introduced which finds short $\mathbf{x}$ such that $\mathbf{x}^T\cdot\mathbf{A} \equiv \mathbf{y} \bmod q$ with $\mathbf{y}$ short. We then obtain $\mathbf{x}^T\cdot\mathbf{A}\cdot\mathbf{s} + \langle\mathbf{x},\mathbf{e}\rangle = \langle\mathbf{y},\mathbf{s}\rangle + \langle\mathbf{x},\mathbf{e}\rangle$, which follows a distribution with small entries when $\mathbf{y}, \mathbf{s}, \mathbf{x}$ and $\mathbf{e}$ are short.

In [Alb17] a composition of the dual attack with a guessing phase (and some scaling) was introduced with a focus on vectors $\mathbf{s}$ that are sparse and small compared to $\mathbf{e}$. The idea is to split $\mathbf{A} = [\mathbf{A}_0|\mathbf{A}_1]$ such that $\mathbf{b} \equiv \mathbf{A}_0\cdot\mathbf{s}_0 + \mathbf{A}_1\cdot\mathbf{s}_1 + \mathbf{e} \bmod q$. Then the dual attack is run on $\mathbf{A}_0$ s.t.

$$\langle\mathbf{x},\mathbf{b}\rangle \equiv \mathbf{x}^T\cdot\mathbf{A}_0\cdot\mathbf{s}_0 + \mathbf{x}^T\cdot\mathbf{A}_1\cdot\mathbf{s}_1 + \langle\mathbf{x},\mathbf{e}\rangle = \langle\mathbf{y},\mathbf{s}_0\rangle + \mathbf{x}^T\cdot\mathbf{A}_1\cdot\mathbf{s}_1 + \langle\mathbf{x},\mathbf{e}\rangle\,.$$

Thus, guessing $\mathbf{s}_1$ and computing $\langle\mathbf{x},\mathbf{v}\rangle - \mathbf{x}^T\cdot\mathbf{A}_1\cdot\mathbf{s}_1$ produces a elements that follow a distribution with small elements. In [EJK20] this was generalised to more general secret distributions paired with additional improvements on the exhaustive search over $\mathbf{s}_1$.

In [GJ21] further improvements were presented. In particular, the search over $\mathbf{s}_1$ is realised using a Fast Fourier Transform style algorithm and the search space is significantly reduced by roughly considering only the most significant bits of $\mathbf{s}_1$. In [MAT22] this last step is replaced by "modulus switching" [BV11,AFFP14] which provides significant performance gains.[1]

2.3 Discrete Gaussian distribution

Let $\sigma > 0$. For any $\mathbf{x} \in \mathbb{R}^d$, we let $\rho_\sigma(\mathbf{x}) = \exp(-\|\mathbf{x}\|^2/2\sigma^2)$. Note that this different from the other, also commonly used definition where $\frac{1}{2}$ is replaced by π in the exponent. This change is inconsequential to our results. For any lattice $\mathcal{L} \subset \mathbb{R}^d$, we denote by $D_{\mathcal{L},\sigma}$ the discrete Gaussian distribution over $\mathcal{L}$, define by $D_{\mathcal{L},\sigma}(\mathbf{x}) = \rho_\sigma(\mathbf{x})/\rho_\sigma(\mathcal{L})$ for all $\mathbf{x} \in \mathcal{L}$.

We will also make use of the modular discrete Gaussian. For any $q \in \mathbb{N}$, we denote by $D_{\mathbb{Z}_q^d,\sigma}$ the modular discrete Gaussian distribution over $\mathbb{Z}_q^d$, defined by

$$D_{\mathbb{Z}_q^d,\sigma}(\mathbf{x}) = \frac{\rho_\sigma(\mathbf{x} + q\mathbb{Z}^d)}{\rho_\sigma(\mathbb{Z}^d)}.$$

Note that the distribution $D_{\mathbb{Z}_q^d,\sigma}$ is isomorphic to the distribution $D^d_{\mathbb{Z}_q,\sigma}$, a fact that will use often implicitly.

2.4 Quantum Computing

Quantum Circuit Model. In the quantum circuit model, the time complexity is the circuit size, which is the total number of elementary quantum gates. The space complexity is the number of qubits used. We will assume that the elementary quantum gates come from a fixed universal set. Up to constant factors, the complexity does not depend on the universal set that we have chosen. Since all unitary transforms are invertible, any quantum circuit $\mathcal{A}$ is reversible and we denote by $\mathcal{A}^\dagger$ its inverse, which is also equal to its conjugate transpose when viewed as a matrix.

Given a function $f : \{0,1\}^n \to \{0,1\}^m$, we say that a quantum circuit, implementing a unitary U that acts on $n+\ell+m$ qubits, *computes f with probability α* if for every x, a measurement on the last m qubits of $U|x\rangle|0^\ell\rangle|0^m\rangle$ outputs $f(x)$ with probability at least α. The exact location of the qubits that we measure for the output actually does not matter, since we can also apply SWAP gates (implementable by elementary gates) to swap them to the m last positions. The extra ℓ qubits that are not part of the input/output are called *ancilla qubits* (or work space).

[1] Another significant gain reported in [MAT22] is due to an improvement to the lattice sieving algorithm from [BDGL16] but discussing this is out of scope of this work.

Quantum Query Model. We use the standard form of the *quantum query model*: given a unitary $\mathcal{O}$, we say that a circuit computes f with *oracle access* to $\mathcal{O}$ if by augmenting the model with the unitary $\mathcal{O}$, we can construct a circuit computing f. The number of *queries* on $\mathcal{O}$ is the number of unitary $\mathcal{O}$ in the circuit. If we find an efficient algorithm for a problem in query complexity and we are given an explicit circuit realizing the black-box transformation of the oracle $\mathcal{O}$, we will have an efficient algorithm for an explicit computational problem.

Quantum Algorithms. We say that a *quantum algorithm computes a function* $F : \{0,1\}^* \to \{0,1\}^*$ *with probability* α if there is a classical algorithm $\mathcal{A}$ *with quantum evaluation* that outputs $F(w)$ with probability α on input w. By quantum evaluation we mean that the algorithm can, any number of times during the computation, build a quantum circuit and evaluate it, that is measure the state $U|0\rangle$ where U is the unitary implemented by the circuit. The *time complexity* $T(n)$ is the classical time complexity of $\mathcal{A}$ plus the time complexity of the circuits (that is the number of gates). The *classical space complexity* $S(n)$ is the space complexity of $\mathcal{A}$ (ignoring quantum evaluations). The *quantum space complexity* $Q(n)$ is the maximal space complexity of all circuits (that is the maximum number of qubits used). In the natural way, we say that a quantum algorithm has oracle access to $\mathcal{O}$ if it produces circuits with oracle access to $\mathcal{O}$. The query complexity of the algorithm is the sum of the query complexity of the circuits. Now that the quantum model of computation is properly defined, we can express the fact that every classical computation can be implemented by a quantum computer, although at a non-negligible cost.

Theorem 1 ([Ben89,LS90]). *Given any $\varepsilon > 0$ and any classical computation with running time T and space complexity S, there exists an equivalent reversible classical computation with running time $O(T^{1+\varepsilon}/S^{\varepsilon})$ and space complexity $O(S(1+\ln(T/S)))$.*

Corollary 1. *Given any $\varepsilon > 0$ and any classical computation with running time T and space complexity S, there exists an equivalent quantum circuit of size $O(T^{1+\varepsilon}/S^{\varepsilon})$ using $O(S(1+\ln(T/S)))$ qubits.*

In principle, it is always possible to turn a classical computation into a quantum one (Corollary 1) and combine all quantum algorithms into one quantum circuit by postponing all measurements until the very end of the computation, using the so-called the *principle of deferred measurement* [NC11]. We will use this fact implicitly in the rest of the paper and just assume that we can take any classical algorithm and turn into a quantum one with the same complexity.

Quantum Search. One of the most well-known quantum algorithms is Grover's unstructured search algorithm [Gro96]. Suppose we have a set of objects named $\{0, 1, \dots, N-1\}$, of which some are *targets*. We say that an oracle $\mathcal{O}$ *identifies the targets* if, in the classical (resp. quantum) setting, $\mathcal{O}(i) = 1$ (resp. $\mathcal{O}|i\rangle = -|i\rangle$) when i is a target and $\mathcal{O}(i) = 0$ (resp. $\mathcal{O}|i\rangle = |i\rangle$) otherwise. Given such an oracle $\mathcal{O}$, the goal is to find a target $j \in \{0, 1, \ldots, N-1\}$ by making queries to the oracle $\mathcal{O}$.

In the search problem, one tries to minimise the number of queries to the oracle. In the classical case, one needs $O(N)$ queries to solve such a problem. Grover, on the other hand, provides a quantum algorithm, that solves the search problem with only $O(\sqrt{N})$ queries [Gro96] when there is one target, and $O(\sqrt{N/t})$ when there are exactly t targets. We present here a generalisation of Grover's algorithm called amplitude amplification [BHMT02a].

Theorem 2 (Amplitude Amplification [BHMT02a]). *Suppose we have a set of N objects of which some are targets. Let $\mathcal{O}$ be a quantum oracle that identifies the targets. Let $\mathcal{A}$ be a quantum circuit using no intermediate measurements, ie $\mathcal{A}$ is reversible. Let a be the initial success probability of $\mathcal{A}$, that is the probability that a measurement of $\mathcal{A}\,|0\rangle$ outputs a target. There exists a quantum algorithm that calls $\mathcal{O}\left(\sqrt{1/a}\right)$ times $\mathcal{A}$, $\mathcal{A}^\dagger$ and $\mathcal{O}$, uses as many qubits as $\mathcal{A}$ and $\mathcal{O}$, and outputs a target with probability greater than $1 - a$.*

Grover's algorithm is a particular case of this theorem where $\mathcal{A}$ produces a uniform superposition of all objects, in which case $a = \frac{1}{N}$. The theorem then states that we can find a target with probability $1 - \frac{1}{N}$ using $O(\sqrt{N})$ calls to the oracle $\mathcal{O}_f$.

Theorem 3 (Amplitude Estimation [BHMT02b], Theorem 12). *Given natural number M and access to an $(n+1)$-qubit unitary U satisfying*

$$U\,|0^n\rangle\,|0\rangle = \sqrt{a}\,|\phi_1\rangle\,|1\rangle + \sqrt{1-a}\,|\phi_0\rangle\,|0\rangle\,,$$

where $|\phi_1\rangle$ and $|\phi_0\rangle$ are arbitrary n-qubit states and $0 < a < 1$, there exists a quantum algorithm that uses M applications of U and $U^\dagger$, and outputs an estimate $\tilde{a}$ that with probability $\geq 2/3$ satisfies

$$|a - \tilde{a}| \leq \frac{6\pi\sqrt{a(1-a)}}{M} + \frac{9\pi^2}{M^2} \leq \frac{15\pi^2}{M}.$$

We will have to search for a marked element in a collection but the oracle that identifies the targets may be probabilistic and return a wrong result with small probability.

Theorem 4 ([HMdW03]). *Given n algorithms, quantum or classical, each computing some bit-value with bounded error probability, there is a quantum algorithm that uses $O(\sqrt{n})$ queries and with constant probability: returns the index of a "1", if there are at least one "1" among the n values; returns $\perp$ if there are no "1".*

This algorithm can easily be used to find the index of the *first algorithm* that returns 1, see e.g. [KKM$^+$21]

Lemma 1. *There exists a quantum algorithm $\mathcal{A}$ with the following property. Let N be an integer and $f : [0, N-1] \to \{0,1\}$ a (classical or quantum with bounded error) function. Let n_0 be the first index such that $f(n_0) = 1$, or let $n_0 = \perp$ if no such index exists. Then $\mathcal{A}^f(N)$ returns $i \in [0, N-1]$ such that $f(i) = 1$, or $\perp$. With constant probability, $\mathcal{A}^f(N) = n_0$. The algorithm runs in expected time $T = O(\sqrt{n_0})$ (or $O(\sqrt{N})$ if $n_0 = \perp$), uses a polynomial number of qubits and makes an expected number T of calls to f. Furthermore, if the algorithm returns $i \in [0, N-1]$, then it only queries f on values in $[0, \min(N-1, 2i)]$.*

Access to Memory "Baseline" quantum circuits are simply built using a universal quantum gate set. A requirement for many quantum algorithms to process data efficiently is to be able to access classical data in quantum superposition. Such algorithms use quantum random-access memory, often denoted as qRAM, and require the circuit model to be augmented with the so-called "qRAM gate". These qRAM gates are assumed to have a time complexity polylogarithmic in the amount of classical data stored, so that each call is not time consuming. This model is inspired by the classical RAM model where we usually assume memory access in time $O(1)$.[2]

Given an input register $0 \leq i \leq r-1$, which represents the index of a memory cell, and many quantum registers $|x_0, \dots x_{r-1}\rangle$, which represent stored data, the qRAM gate fetches the data from register x_i, possibly in superposition:

$$|i\rangle |x_0, \dots x_{r-1}\rangle |y\rangle \mapsto |i\rangle |x_0, \dots x_{r-1}\rangle |y \oplus x_i\rangle \quad .$$

Following the terminology of [Kup13], three are types of qRAMs:

- If the input i is classical, then this is the plain quantum circuit model. We can implement it using a universal quantum gate set.
- If the x_j are classical, we have *classical memory with quantum random access* (QRACM). The qRAM gate becomes

$$|i\rangle |y\rangle \mapsto |i\rangle |y \oplus x_i\rangle \quad .$$

- In general, we have *quantum memory with quantum random access* (QRAQM). This is the most powerful quantum memory model where the data are also in superposition.

In our algorithm for the dual attack, we will be using QRACM. It is possible to implement a QRACM using a universal quantum gate set, albeit at a considerable cost. Given a classical data set $\{x_0, \cdots, x_{r-1}\}$, one can construct, in time $\tilde{O}(r)$, a circuit using $\tilde{O}(r)$ qubits that implements a QRACM for this data set. The obtained circuit then allows query in the form $|i\rangle |y\rangle \mapsto |i\rangle |y \oplus x_i\rangle$ and has circuit depth $O(\text{polylog}(r))$ [GLM08,KP20,MGM20,HLGJ20]. Note that even low depth implementation of QRACM has at least $\Omega(r)$ gates, hence has time complexity at least $\Omega(r)$ by our definition. Therefore, the assumption that the qRAM gates have time complexity polylog(r) is very strong and corresponds to parallel evaluation of the circuit.

[2] The validity of this assumption in the context of lattice reduction is hotly debated, but this assumption is commonly made.

2.5 The Classical Algorithm of [GJ21,MAT22]

In this section, we give an overview of the algorithm in [MAT22]. Our quantum algorithm will be a modified version that rely essentially on the same analysis for the correctness but a new analysis for the quantum complexity.

We are given a sample from $\mathrm{LWE}_{n,m,\chi_s,\chi_e}$, where χ_s and χ_e have small variance σ_s^2 and σ_e^2 respectively. We partition $\mathbf{s}$ into three components:

$$\mathbf{s} = \begin{pmatrix} \mathbf{s}_{\text{enum}} \\ \mathbf{s}_{\text{fft}} \\ \mathbf{s}_{\text{lat}} \end{pmatrix}$$

where $\mathbf{s}_{\text{enum}}$ has k_{enum} coordinates, $\mathbf{s}_{\text{fft}}$ has k_{fft} coordinates, and $\mathbf{s}_{\text{lat}}$ has $k_{\text{lat}} = n - k_{\text{enum}} - k_{\text{fft}}$ coordinates. We spit A into three components accordingly as well:

$$\mathbf{A} = [\mathbf{A}_{\text{enum}}|\mathbf{A}_{\text{fft}}|\mathbf{A}_{\text{lat}}]$$

so that $\mathbf{A} \cdot \mathbf{s} = \mathbf{A}_{\text{enum}} \cdot \mathbf{s}_{\text{enum}} + \mathbf{A}_{\text{fft}} \cdot \mathbf{s}_{\text{fft}} + \mathbf{A}_{\text{lat}} \cdot \mathbf{s}_{\text{lat}}$.

We define the matrix:

$$B = \begin{pmatrix} \alpha \mathbf{I}_m & 0 \\ \mathbf{A}_{\text{lat}}^T & q\mathbf{I}_{k_{\text{lat}}} \end{pmatrix},$$

where α is a constant equal to $\frac{\sigma_e}{\sigma_s}$ and is used for normalization in the case that $\mathbf{s}, \mathbf{e}$ have different distributions. We find D short vectors in the column space of $\mathbf{B}$ using some short vectors sampling procedure (see Algorithm 1).

Given a list L of D vectors $\begin{pmatrix} \alpha \cdot \mathbf{x}_j \\ \mathbf{y}_{j,\text{lat}} \end{pmatrix}$, let $\mathbf{y}_{j,\text{fft}} = \mathbf{x}_j^T \cdot \mathbf{A}_{\text{fft}}$ and $\mathbf{y}_{j,\text{enum}} = \mathbf{x}_j^T \cdot \mathbf{A}_{\text{enum}}$. We can then define the function $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) =$

$$\Re\left(\frac{1}{\psi(\tilde{\mathbf{s}}_{\text{fft}})} \sum_j \exp\left(\left(\left\lfloor \frac{p}{q} \cdot \mathbf{y}_{j,\text{fft}} \right\rceil^T \cdot \tilde{\mathbf{s}}_{\text{fft}} + \frac{p}{q} \cdot \mathbf{y}_{j,\text{enum}}^T \cdot \tilde{\mathbf{s}}_{\text{enum}} - \frac{p}{q} \cdot \mathbf{x}_j^T \cdot \mathbf{b}\right) \cdot \frac{2i\pi}{p}\right)\right)$$

for all $\tilde{\mathbf{s}}_{\text{enum}} \in \mathbb{Z}_q^{k_{\text{enum}}}$ and $\tilde{\mathbf{s}}_{\text{fft}} \in \mathbb{Z}_p^{k_{\text{fft}}}$, where ψ is a complex factor of norm 1 defined in [MAT22, p. 25, proof of Lemma 5.4] and easily computable. The function F_L essentially performs an FFT on values drawn from a certain distribution. Via an analysis that we do not reproduce, one can show that the function F_L above has the following properties (with high probability on the choice of the elements in L, assuming sufficiently many vectors):

- If $\tilde{\mathbf{s}}_{\text{enum}} \neq \mathbf{s}_{\text{enum}}$ then $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) < C$ for all $\tilde{\mathbf{s}}_{\text{fft}} \in \mathbb{Z}_p^{k_{\text{fft}}}$.
- $F_L(\mathbf{s}_{\text{enum}}, \mathbf{s}_{\text{fft}}) > C$
- There might be $\tilde{\mathbf{s}}_{\text{fft}} \neq \mathbf{s}_{\text{fft}}$ such that $F_L(\mathbf{s}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > C$.

The first point corresponds to a wrong guess. In this case, values on which the FFT is performed follow a uniform distribution and the expected value of $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}})$ is 0. The second point corresponds to the correct guess. In this case, values on which the FFT is performed essentially follow a normal distribution with nonzero mean and therefore the expected value of $F_L(\mathbf{s}_{\text{enum}}, \mathbf{s}_{\text{fft}})$ is nonzero. By carefully choosing the value of C, and taking sufficiently many samples in the list, we can ensure that those properties holds with high probability. The third point follows from the fact that [MAT22] performs a modulo switching operation that can introduce some errors and makes the analysis of $F_L(\mathbf{s}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}})$ with $\tilde{\mathbf{s}}_{\text{fft}} \neq \mathbf{s}_{\text{fft}}$ more difficult. Consequently, it is simpler to assume that one can only recover $\mathbf{s}_{\text{enum}}$ with certainty.

We can therefore reformulate the algorithm of [MAT22] as looking for $\tilde{\mathbf{s}}_{\text{enum}}$ such that there exists $\tilde{\mathbf{s}}_{\text{fft}}$ such that $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > C$. We will rely on the following lemma in our analysis below.

Lemma 2 (Adapted from Theorem 5.2 in [MAT22]). *Let $(n, m, q, \chi_s, \chi_e)$ be LWE parameters, let $(\beta_0, \beta_1, k_{enum}, k_{fft}, k_{lat}, p, \mu)$ be parameters for Algorithm 2. Let σ_e be the standard deviate of χ_e, σ_s be the standard deviation of χ_s and $\alpha = \sigma_e/\sigma_s$. Denote by ℓ the expected Euclidean length of the vectors returned by Algorithm 1. Then, Algorithm 2 succeeds with probability at least $1 - \mu$ for*

$$C = \phi_{\text{fp}}(\mu) \cdot \sqrt{D_{\text{arg}} \cdot D} \quad \text{and} \quad D \geq D_{\text{eq}} \cdot D_{\text{round}} \cdot D_{\text{arg}} \cdot D_{\text{fpfn}}(\mu)$$

where

$$\begin{aligned} D_{\text{eq}} &= e^{4\left(\frac{\pi\tau}{q}\right)^2} \text{ for } \tau^2 = \frac{\alpha^{-2} \cdot \|\mathbf{e}\|^2 + \|\mathbf{s}_{lat}\|^2}{m + k_{lat}} \ell^2, \\ D_{\text{round}} &= \left(\prod_{\substack{t=0 \\ s_t \neq 0}}^{k_{fft}-1} \left(\frac{\sin\left(\frac{\pi s_t}{p}\right)}{\frac{\pi s_t}{p}} \right) \right)^{-2} \text{ for } \mathbf{s}_{fft} = (s_0, \ldots, s_{k_{fft}-1}), \\ D_{\text{arg}} &= \frac{1}{2} + e^{-8\left(\frac{\pi\tau}{q}\right)^2}, \\ D_{\text{fpfn}}(\mu) &= \left(\Phi^{-1}\left(1 - \frac{\mu}{2 \cdot N_{\text{enum}}(\mathbf{s}_{\text{enum}}) \cdot p^{k_{\text{fft}}}}\right) + \Phi^{-1}\left(1 - \frac{\mu}{2}\right) \right)^2. \end{aligned}$$

3 Quantum Augmented Dual Attack

We now modify the algorithm of [MAT22] to obtain a quantum speedup. At a high-level, Algorithm 3 works in the same way. First, we run a sampling algorithm to obtain short vectors in the dual. Here, we can take advantage of the existing quantum speedups for sieving [LMv13,Laa15,AGPS20,CL21].

Next, we can obtain a quadratic speedup on the search for $\mathbf{s}_{\text{enum}}$. Indeed, the algorithm simply enumerates them one by one until the correct one is found. By carefully choosing the order and applying a variant of Grover's search algorithm, we can obtain a quadratic speedup (see Section 4). In our case, the quantum search will call an oracle that is probabilistic so care must be taken. We use the improved version of Grover's search in Theorem 4 that can handle bounded-error inputs.

Algorithm 2: Dual Attack of [MAT22]

```
   Input: LWE parameters $(n, m, q, \chi_s, \chi_e)$, integers $\beta_0, \beta_1 \leqslant d$, integers
          $k_{\text{enum}}, k_{\text{fft}}, k_{\text{lat}}$ such that $k_{\text{enum}} + k_{\text{fft}} + k_{\text{lat}} = n$, an integer $p \leqslant q$, an
          integer $D$, a real number $C$, and an LWE pair $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$.
   Output: The first $k_{\text{enum}}$ coordinates of $\mathbf{s}$.
1  Decompose $\mathbf{A}$ as $\begin{bmatrix}\mathbf{A}_{\text{enum}} & \mathbf{A}_{\text{fft}} & \mathbf{A}_{\text{lat}}\end{bmatrix}$ of respective dimensions $m \times k_{\text{enum}}$, $m \times k_{\text{fft}}$
     and $m \times k_{\text{lat}}$.
2  Compute the matrix $\mathbf{B} = \begin{bmatrix}\alpha\mathbf{I}_m & \mathbf{0} \\ \mathbf{A}_{\text{lat}}^T & q\mathbf{I}_{k_{\text{lat}}}\end{bmatrix}$ where $\alpha = \frac{\sigma_e}{\sigma_s}$
3  Run Algorithm 1 on the basis $\mathbf{B}$ with parameters $\beta_0, \beta_1, D$ to get a list $L$ of $D$
     short vectors.
4  for every value $\tilde{\mathbf{s}}_{\text{enum}}$ in descending order of probability according to the secret
     distribution do
5      Initialise a table $T$ of dimensions $\underbrace{p \times p \times \cdots p}_{k_{\text{fft}} \text{ times}}$
6      for every short vector $(\alpha\mathbf{x}_j, \mathbf{y}_{\text{lat}})$ in $L$ do
7          Compute $\mathbf{y}_{j,\text{fft}} = \mathbf{x}_j^T \cdot \mathbf{A}_{\text{fft}}$.
8          Compute $\mathbf{y}_{j,\text{enum}} = \mathbf{x}_j^T \cdot \mathbf{A}_{\text{enum}}$.
9          Add $\exp\left((\mathbf{x}_j^T \cdot \mathbf{b} - \mathbf{y}_{j,\text{enum}}^T \cdot \tilde{\mathbf{s}}_{\text{enum}}) \cdot \frac{2i\pi}{q}\right)$ to cell $\lfloor \frac{p}{q}\mathbf{y}_{j,\text{fft}} \rceil$ of $T$.
10     Perform FFT on $T$
11     if for any $\tilde{\mathbf{s}}_{\text{fft}}$, the real part of $\frac{1}{\psi(\tilde{\mathbf{s}}_{\text{fft}})} T[\tilde{\mathbf{s}}_{\text{fft}}]$ is larger than $C$ then
12         return $\tilde{\mathbf{s}}_{\text{enum}}$.
13 return $\perp$
```

We now move to the most interesting part of our quantum speedup. In their algorithm [MAT22], the authors first fill a large array T, perform an FFT and then look at all the entries to check if one is larger than a given threshold C. While it is tempting to use the quantum Fourier transform (QFT), which runs in polynomial time, we do not know how to implement the second step (checking each entry) efficiently. Indeed, the QFT works on the amplitudes and, therefore, simply extracting a coefficient of the result is a nontrivial task (see Section 6). We work around this issue by observing two points:

1. The input array of the FFT is relatively sparse: it has D nonzero entries (out of $p^{k_{\text{fft}}}$).
2. We can obtain a quadratic speedup on the task of evaluating a sum of cosine (Theorem 5).

Since every entry of the output of the FFT is a sum of cosine that we can evaluate efficiently, and since the sum only has D terms, we can evaluate each coefficient in reasonable time. By turning this algorithm into a quantum oracle, we can use Grover's search to obtain a further quadratic speedup on the inner part of the algorithm that looks for an entry above the threshold C.

A crucial detail of this algorithm is our use of a QRACM. Indeed, in order to apply Theorem 5 and obtain a quadratic speedup when evaluating the sums, we need a quantum oracle access to the short vectors stored in L. Since those vectors are obtained by a classical algorithm, we store them in a QRACM to build this oracle.[3]

3.1 A Quantum Algorithm for Mean Estimation

We provide here a quantum algorithm which estimates the mean value of

$$\cos(2\pi(\langle \mathbf{w}_i, \mathbf{b}\rangle)/q)$$

used in the dual attack. The idea is inspired by [ACKS20, Theorem 47] and can be seen as a special case of quantum speedup of Monte Carlo methods [Mon15].

Theorem 5. *Let N be a positive integer, $W = \mathbf{w}_0, \ldots, \mathbf{w}_{N-1}$ be the list of N vectors. Let $f_W(\mathbf{b}) = \frac{1}{N}\sum_{i=0}^{N-1} \cos(2\pi(\langle \mathbf{w}_i, \mathbf{b}\rangle)/q)$. Let $\mathcal{O}_W : |j\rangle |0\rangle \mapsto |j\rangle |\mathbf{w}_j\rangle$. For any $\varepsilon, \delta > 0$, there exists a quantum algorithm $\mathcal{A}$ that given $\mathbf{b} \in \mathbb{Z}_q^n$ and oracle access to $\mathcal{O}_W$, outputs $\mathcal{A}^{\mathcal{O}_W}(\mathbf{b})$ which satisfy $|\mathcal{A}^{\mathcal{O}_W}(\mathbf{b}) - f_W(\mathbf{b})| \leq \varepsilon$ with probability $1-\delta$. The algorithm make $\mathcal{O}(\varepsilon^{-1} \cdot \log\frac{1}{\delta})$ queries to $\mathcal{O}_W$, and requires $\varepsilon^{-1} \cdot \log \frac{1}{\delta} \cdot \mathsf{poly}(\log n)$ elementary quantum gates.*

Proof. We define the positive controlled rotation oracle as, for any $a \in \mathbb{R}$

$$\mathcal{O}_{CR^+} : |a\rangle |0\rangle \to \begin{cases} |a\rangle \left(\sqrt{a}\,|1\rangle + \sqrt{1-a}\,|0\rangle\right), & \text{if } a \geq 0 \\ |a\rangle |0\rangle, & \text{otherwise,} \end{cases}$$

which can be implemented up to negligible error by $\mathsf{poly}(\log n)$ quantum elementary gates. Also, we define the cosine inner product oracle as for any $\mathbf{b}, \mathbf{w} \in \mathbb{Z}^n$

$$\mathcal{O}_{\cos} : |\mathbf{w}\rangle |\mathbf{b}\rangle |0\rangle \to |\mathbf{w}\rangle |\mathbf{b}\rangle |\cos(2\pi\langle \mathbf{w}, \mathbf{b}\rangle/q)\rangle ,$$

which can also be implemented by $\mathsf{poly}(\log n)$ quantum elementary gates. Prepare the state $\frac{1}{\sqrt{N}} \sum\limits_{j=0}^{N-1} |j\rangle |\mathbf{0}\rangle |\mathbf{b}\rangle |0\rangle |0\rangle$, and then we apply $\mathcal{O}_W$ on the first and second registers (storing $\mathbf{w}_j$ there), apply $\mathcal{O}_{\cos}$ on the second, third, fourth registers (storing $\cos(2\pi\langle \mathbf{w}, \mathbf{b}\rangle/q)$ there), and apply $\mathcal{O}_{CR^+}$ on the fourth and fifth registers. Writing $\gamma_j := \cos(2\pi\langle \mathbf{w}_j, \mathbf{b}\rangle/q)$ and letting sums run over $j \in [0, N-1]$, we have

$$\frac{1}{\sqrt{N}} \sum_{\gamma_j \geq 0} |j\rangle |\mathbf{w}_j\rangle |\mathbf{b}\rangle |\gamma_j\rangle \left(\sqrt{\gamma_j}\,|1\rangle + \sqrt{1-\gamma_j}\,|0\rangle\right) + \frac{1}{\sqrt{N}} \sum_{\gamma_j < 0} |j\rangle |\mathbf{w}_j\rangle |\mathbf{b}\rangle |\gamma_j\rangle |0\rangle .$$

By rearranging, we obtain

$$\frac{1}{\sqrt{N}} \sum_{\gamma_j \geq 0} \sqrt{\gamma_j}\, |j\rangle |\mathbf{w}_j\rangle |\mathbf{b}\rangle |\gamma_j\rangle |1\rangle$$

[3] Note that quantum augmented sieving procedures still output classical lists of short vectors.

Algorithm 3: Quantum Augmented Dual Attack

```
**Input:** LWE parameters $(n, m, q, \chi_s, \chi_e)$, integers $\beta_0, \beta_1 \leqslant d$, integers
    $k_{\text{enum}}, k_{\text{fft}}, k_{\text{lat}}$ such that $k_{\text{enum}} + k_{\text{fft}} + k_{\text{lat}} = n$, an integer $p \leqslant q$, an
    integer $D$, a real number $C$, a coefficient $\eta \in [0, 1]$ and an LWE pair
    $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$.
**Output:** The first $k_{\text{enum}}$ coordinates of $s$.
1  Decompose $\mathbf{A}$ as $\begin{bmatrix}\mathbf{A}_{\text{enum}} & \mathbf{A}_{\text{fft}} & \mathbf{A}_{\text{lat}}\end{bmatrix}$ of respective dimensions $m \times k_{\text{enum}}$, $m \times k_{\text{fft}}$
     and $m \times k_{\text{lat}}$.
2  Compute the matrix $\mathbf{B} = \begin{bmatrix} q\mathbf{I}_{k_{\text{lat}}} & \mathbf{A}_{\text{lat}}^T \\ \mathbf{0} & \alpha \mathbf{I}_m \end{bmatrix}$ where $\alpha = \frac{\sigma_e}{\sigma_s}$
3  Run Algorithm 1 on the basis $\mathbf{B}$ with parameters $\beta_0, \beta_1, D$ to get a list $L$ of $D$
     short vectors.
4  Create a QRAM $O_W$
5  for every short vector $(\alpha \cdot \mathbf{x}_j, \mathbf{y}_{j,\text{lat}})$ in $L$ do
6      Add vector $\mathbf{x}_j$ to $\mathcal{O}_W$ at index $j$
7  Use Theorem 5 to create an algorithm $\mathcal{A}$ with $\delta = \frac{1}{10}$, $\varepsilon = \frac{C}{D}\eta$ and “$q$”$=p$
8  create oracle $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}})$:
9      create oracle $\hat{\mathcal{O}}(\tilde{\mathbf{s}}_{\text{fft}})$:
10         Compute $\theta$ such that $\psi(\tilde{\mathbf{s}}_{\text{fft}}) = e^{-\frac{2i\pi}{p}\theta}$ (recall that $|\psi(\tilde{\mathbf{s}}_{\text{fft}})| = 1$)
11         create oracle $\mathcal{O}'_W(j)$:
12             Get $\mathbf{x}_j$ from $\mathcal{O}_W$ at index $j$
13             Compute $\mathbf{y}_{j,\text{fft}} = \mathbf{x}_j^T \cdot \mathbf{A}_{\text{fft}}$
14             Compute $\mathbf{y}_{j,\text{enum}} = \mathbf{x}_j^T \cdot \mathbf{A}_{\text{enum}}$
15             return vector $\left(\frac{p}{q} \cdot \mathbf{y}_{j,\text{enum}}, \left\lfloor \frac{p}{q} \cdot \mathbf{y}_{j,\text{fft}} \right\rceil, \theta - \frac{p}{q} \cdot \mathbf{x}_j^T \cdot \mathbf{b}\right)$
16         if $\mathcal{A}^{\mathcal{O}'_W}((\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1)) > (1 + \eta) \cdot \frac{C}{D}$ then
17             return 1
18         else
19             return 0
20     Use Theorem 4 to find, with probability $\frac{9}{10}$, $i$ such that $\hat{\mathcal{O}}(i) = 1$ or let
         $i = \perp$ if none exists
21     if $i \neq \perp$ then
22         return 1
23     else
24         return 0
25 create oracle $\tilde{\mathcal{O}}(i)$:
26     Compute the $i^{th}$ most probable $\tilde{\mathbf{s}}_{\text{enum}}$ according to the distribution $\chi_s$
27     return $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}})$
28 Find, with probability $\frac{9}{10}$, $\tilde{\mathbf{s}}_{\text{enum}}$ such that $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}}) = 1$ using Lemma 1 with
     oracle $\tilde{\mathcal{O}}$, or let $\tilde{\mathbf{s}}_{\text{enum}} = \perp$ if none is found
29 return $\tilde{\mathbf{s}}_{\text{enum}}$.
```

$$
\begin{aligned}
&+\frac{1}{\sqrt{N}}\Bigg(\sum_{\gamma_j\ge 0}\sqrt{1-\gamma_j}\,|j\rangle\,|\mathbf{w}_j\rangle\,|\mathbf{b}\rangle\,|\gamma_j\rangle+\sum_{\gamma_j<0}|j\rangle\,|\mathbf{w}_j\rangle\,|\mathbf{b}\rangle\,|\gamma_j\rangle\Bigg)\,|0\rangle\\
=&\sqrt{a^+}\,|\phi_1\rangle\,|1\rangle+\sqrt{1-a^+}\,|\phi_0\rangle\,|0\rangle\,,
\end{aligned}
$$

where $a^+ = \sum_{\gamma_j \geq 0} \frac{\gamma_j}{N}$. By applying Theorem 3, we can estimate a^+ with additive error $\varepsilon/2$ by using $\mathcal{O}(\varepsilon^{-1})$ applications of $\mathcal{O}_W$, $\mathcal{O}_W^\dagger$, and $\varepsilon^{-1} \cdot \mathsf{poly}(\log n)$ elementary quantum gates. Following the same strategy, we can also estimate $a^- = \sum_{\gamma_j<0} \frac{\gamma_j}{N}$ with same additive error and by using same amount of queries and quantum elementary gates. Therefore, we can estimate

$$a^+ + a^- \pm \varepsilon = \sum_j \frac{\cos(2\pi\langle \mathbf{w}_j, \mathbf{b}\rangle/q)}{N}.$$

By repeating the procedure $\Theta(\log \frac{1}{\delta})$ times and take the median among them, we finish the proof. □

3.2 Analysis of the Quantum Augmented Dual Attack

We now analyse the quantum augmented dual attack given in Algorithm 3.

Theorem 6. *Let $(n, m, q, \chi_s, \chi_e)$ be LWE parameters, let*

$$(\beta_0, \beta_1, k_{enum}, k_{fft}, k_{lat}, p, D, C, A, b, \eta)$$

be the input of Algorithm 3. Let L be the list of vectors obtained at Line 3 of Algorithm 3. For any $x > 0$, let $S_x^L = \{\, \tilde{\mathbf{s}}_{enum} : \exists \tilde{\mathbf{s}}_{fft}, F_L(\tilde{\mathbf{s}}_{enum}, \tilde{\mathbf{s}}_{fft}) > x \,\}$. With probability at least $9/10$, the algorithm returns a value in $S_C^L \cup \{\, \perp \,\}$. Furthermore, if $S_{(1+2\eta)C}^L \neq \varnothing$ then the algorithm returns a value in S_C^L with probability at least $9/10$.

Proof. Below, we will establish the following claims:

(1) For all $\tilde{\mathbf{s}}_{\text{enum}}$, the oracle $\hat{\mathcal{O}}$ inside $\mathcal{O}(\tilde{s}_{\text{enum}})$ is such that, for all $\tilde{\mathbf{s}}_{\text{fft}}$, with probability at least $1-\delta$, if $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > (1+2\eta)\cdot C$ then $\hat{\mathcal{O}}(\tilde{\mathbf{s}}_{\text{fft}}) = 1$ and if $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{s}_{\text{fft}}) \leqslant C$ then $\hat{\mathcal{O}}(\tilde{\mathbf{s}}_{\text{fft}}) = 0$.
(2) For all $\tilde{\mathbf{s}}_{\text{enum}}$, with probability at least $9/10$, if there exists $\tilde{\mathbf{s}}_{\text{fft}}$ such that $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > (1+2\eta)\cdot C$ then $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}}) = 1$.
(3) For all $\tilde{\mathbf{s}}_{\text{enum}}$, with probability at least $9/10$, if $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) \leqslant C$ for all $\tilde{\mathbf{s}}_{\text{enum}}$ then $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}}) = 0$.
(4) With probability at least $9/10$, if the algorithm returns $\tilde{\mathbf{s}}_{\text{enum}} \neq \perp$ then there exists $\tilde{\mathbf{s}}_{\text{fft}}$ such that $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > C$.

We start by establishing the result as following from the claims and then establish these claims below. Let x be the output of the algorithm. By claim (4), if $x \neq \perp$ then, with probability at least $9/10$, there exist $\tilde{\mathbf{s}}_{\text{fft}}$ such that $F(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > C$. Therefore, $x \in S_C$. Hence, this proves that $x \in S_C \cup \{\, \perp \,\}$ with probability at least 9/10. Now assume that $S_{(1+2\eta)C} \neq \varnothing$ and let $\tilde{\mathbf{s}}_{\text{enum}} \in S_{(1+2\eta)C}$. Then by claim (2), with probability at least 9/10, $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}}) = 1$ so the algorithm will not return $\perp$, i.e. $x \neq \perp$.

Proof of claim (1). Fix $\tilde{\mathbf{s}}_{\text{enum}}$ and check that $\hat{\mathcal{O}}(\tilde{\mathbf{s}}_{\text{fft}})$ returns 1 if and only if

$$\mathcal{A}^{\mathcal{O}'_W}((\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1)) > (1+\eta) \cdot \tfrac{C}{D}.$$

Now $\mathcal{O}_W$ is defined in such a way that

$$\mathcal{O}'_W(j) = \left(\tfrac{p}{q} \cdot \mathbf{y}_{i,\text{enum}}, \left\lceil \tfrac{p}{q} \cdot \mathbf{y}_{i,\text{fft}} \right\rfloor, \theta - \tfrac{p}{q} \cdot \mathbf{x}_j^T \cdot \mathbf{b}\right)$$

where $\mathbf{y}_{j,\text{enum}}$ and $\mathbf{y}_{j,\text{fft}}$ are defined as expected. Therefore, by Theorem 5, with probability at least $1-\delta$,

$$\left|\mathcal{A}^{\mathcal{O}'_W}((\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1)) - f_W((\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1))\right| \leqslant \varepsilon.$$

But one checks that

$$\begin{aligned}\langle \mathcal{O}_W(j), (\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1)\rangle &= \left\langle \left(\frac{p}{q} \cdot \mathbf{y}_{i,\text{enum}}, \left\lceil \frac{p}{q} \cdot \mathbf{y}_{i,\text{fft}} \right\rfloor, \theta - \tfrac{p}{q} \cdot \mathbf{x}_j^T \cdot \mathbf{b}\right), (\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1)\right\rangle \\ &= \frac{p}{q} \cdot \mathbf{y}_{i,\text{enum}}^T \cdot \tilde{\mathbf{s}}_{\text{enum}} + \left\lceil \frac{p}{q} \cdot \mathbf{y}_{i,\text{fft}} \right\rfloor^T \cdot \tilde{\mathbf{s}}_{\text{fft}} + \theta - \frac{p}{q} \cdot \mathbf{x}_j^T \cdot \mathbf{b}.\end{aligned}$$

Therefore, $f_W((\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1))$

$$\begin{aligned}&= \frac{1}{D} \sum_j \cos\left(\frac{2\pi}{p} \langle \mathcal{O}_W(j), (\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1)\rangle\right) \\ &= \frac{1}{D} \sum_j \cos\left(\frac{2\pi}{p}\left(\frac{p}{q} \cdot \mathbf{y}_{i,\text{enum}}^T \cdot \tilde{\mathbf{s}}_{\text{enum}} + \left\lceil \frac{p}{q} \cdot \mathbf{y}_{i,\text{fft}} \right\rfloor^T \cdot \tilde{\mathbf{s}}_{\text{fft}} + \theta - \frac{p}{q} \cdot \mathbf{x}_j^T \cdot \mathbf{b}\right) + \frac{2\pi}{p} \cdot \theta\right) \\ &= \frac{1}{D} \Re\left(\sum_j \exp\left(\frac{2i\pi}{p}\left(\frac{p}{q} \cdot \mathbf{y}_{i,\text{enum}}^T \cdot \tilde{\mathbf{s}}_{\text{enum}} + \left\lceil \frac{p}{q} \cdot \mathbf{y}_{i,\text{fft}} \right\rfloor^T \tilde{\mathbf{s}}_{\text{fft}} - \frac{p}{q} \cdot \mathbf{x}_j^T \cdot \mathbf{b}\right) + \frac{2i\pi}{p} \cdot \theta\right)\right) \\ &= \frac{1}{D} \Re\left(e^{\frac{2i\pi}{p}\theta} \sum_j \exp\left(\frac{2i\pi}{p}\left(\frac{p}{q} \cdot \mathbf{y}_{i,\text{enum}}^T \cdot \tilde{\mathbf{s}}_{\text{enum}} + \left\lceil \frac{p}{q} \cdot \mathbf{y}_{i,\text{fft}} \right\rfloor^T \cdot \tilde{\mathbf{s}}_{\text{fft}} - \frac{p}{q} \cdot \mathbf{x}_j^T \cdot \mathbf{b}\right)\right)\right) \\ &= \frac{1}{D} F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}})\end{aligned}$$

since θ was computed so that $\psi(\tilde{\mathbf{s}}_{\text{fft}}) = e^{-\frac{2i\pi}{p}\theta}$. It follows that, with probability at least $1-\delta$,

$$\left|\mathcal{A}^{\mathcal{O}'_W}((\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1)) - \frac{1}{D} F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}})\right| \leqslant \varepsilon = \frac{C}{D} \cdot \eta.$$

Assume that this inequality holds.

– If $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > (1+2\eta) \cdot C$ then $\mathcal{A}^{\mathcal{O}'_W}((\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1)) > (1+2\eta) \cdot \frac{C}{D} - \frac{C}{D} \cdot \eta = (1+\eta) \cdot \frac{C}{D}$ so $\hat{\mathcal{O}}(\tilde{\mathbf{s}}_{\text{fft}}) = 1$.

– If $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) \leqslant C$ then $\mathcal{A}^{O'_W}((\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1)) \leqslant \frac{C}{D} + \frac{C}{D}\eta = (1+\eta)\frac{C}{D}$ so $\hat{\mathcal{O}}(\tilde{\mathbf{s}}_{\text{fft}}) = 0$.

Proof of claim (2). Fix $\tilde{\mathbf{s}}_{\text{enum}}$. If there exists $\tilde{\mathbf{s}}_{\text{fft}}$ such that $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > (1+2\eta) \cdot C$ then by claim (1), with probability at least $1-\delta$, $\hat{\mathcal{O}}(\tilde{\mathbf{s}}_{\text{fft}}) = 1$. It follows by Theorem 4 that the search will, with probability at least 9/10, return $i \neq \perp$ and therefore $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}})$ will return 1.
Proof of claim (3). For $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}})$ to return 0, it is sufficient to have $\hat{\mathcal{O}}(\tilde{\mathbf{s}}_{\text{fft}}) = 0$ for all $\tilde{\mathbf{s}}_{\text{fft}}$. By claim (1), $\hat{\mathcal{O}}(\tilde{\mathbf{s}}_{\text{fft}}) = 0$ with probability at least $1-\delta$ when $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) \leqslant C$. Hence, by Theorem 4, the search algorithm will return $\perp$ with probability 9/10 and $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}}) = 0$. Note here that there is no need for a union bound because of Theorem 4.
Proof of claim (4). For the algorithm to return $\tilde{\mathbf{s}}_{\text{enum}}$, with probability 9/10, we must have $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}}) = 1$. By claim (3), with probability at least 9/10, this can only happen if $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > C$ for some $\tilde{\mathbf{s}}_{\text{fft}}$. Therefore the probability that the algorithm returns $\tilde{\mathbf{s}}_{\text{enum}}$ such that $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) \leqslant C$ for all $\tilde{\mathbf{s}}_{\text{fft}}$ is bounded by 1/10, by Lemma 1 This finishes the proof. ☐

Lemma 3. *Let* $(n, m, q, \chi_s, \chi_e)$ *be* LWE *parameters,* $(\beta_0, \beta_1, k_{enum}, k_{fft}, k_{lat}, p)$ *be a partial tuple of parameters for Algorithm 3, and let* $0 < \nu < 1$*. Fix* $(\mathbf{s}, \mathbf{e}) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^m$*. Choosing the parameters* C, D *according to Lemma 2 with* $\mu = \nu/2$*, and* $\eta \leqslant \frac{\sqrt{2\pi}\mu}{8\phi_{fp}}$*, with probability at least* $1-\nu$ *the algorithm returns* $\mathbf{s}_{enum}$*.*

Proof. Recall that for any list L and any $x > 0$, we let

$$S_x^L = \{\, \tilde{\mathbf{s}}_{\text{enum}} : \exists \tilde{\mathbf{s}}_{\text{fft}}, F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > x \,\}\,.$$

In the proof of [MAT22, Theorem 5.2], it is shown that for any threshold[4] X,

$$\Pr_L[F_L(\mathbf{s}_{\text{enum}}, \mathbf{s}_{\text{fft}}) > X] \geq \Phi\left(\phi_{\text{fp}} + \phi_{\text{fn}} - \frac{X}{\sqrt{D_{\text{arg}} \cdot D}}\right).$$

and that for any $\tilde{\mathbf{s}}_{\text{enum}} \neq \mathbf{s}_{\text{enum}}$, any $\tilde{\mathbf{s}}_{\text{fft}}$ and any threshold Y,

$$\Pr_L[F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > Y] \leqslant 1 - \Phi\left(\frac{Y}{\sqrt{D_{\text{arg}} \cdot D}}\right).$$

We are going to apply those inequalities to $X = (1+2\eta) \cdot C$ and $Y = C$. The second inequality, by the choice of C, gives that

$$\Pr_L[F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > C] \leqslant 1 - \Phi\left(\frac{C}{\sqrt{D_{\text{arg}} \cdot D}}\right) = \frac{\mu}{2N_{\text{enum}}(\mathbf{s}_{\text{enum}}) \cdot p^{k_{\text{fft}}}}.$$

The number of guesses of $\tilde{\mathbf{s}}_{\text{enum}}$ before reaching $\mathbf{s}_{\text{enum}}$ is $N_{\text{enum}}(\mathbf{s}_{\text{enum}})$ in the classical case. However note that Lemma 1 may call the oracle on more entries

[4] The proof assume a particular value of C but the first three lines of the derivation in [MAT22, Theorem 5.2] holds for any value of C, which we call X here.

than $N_{\text{enum}}(\mathbf{s}_{\text{enum}})$. Specifically, Lemma 1 guarantees that the oracle will only call the oracle on the first $2N_{\text{enum}}(\mathbf{s}_{\text{enum}})$ entries (with constant probability). Therefore, by a union bound,

$$\Pr_L[F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) \leqslant C \text{ for the first } 2N_{\text{enum}}(\mathbf{s}_{\text{enum}}) \text{ values of } \tilde{\mathbf{s}}_{\text{enum}}] \geq 1-\mu. \quad (1)$$

On the other hand,

$$\begin{aligned}\Pr_L[F_L(\mathbf{s}_{\text{enum}}, \mathbf{s}_{\text{fft}}) > (1+2\eta)C] &\geq \varPhi\left(\phi_{\text{fp}} + \phi_{\text{fn}} - (1+2\eta)\frac{C}{\sqrt{D_{\text{arg}} \cdot D}}\right) \\ &= \varPhi\left(\phi_{\text{fp}} + \phi_{\text{fn}} - (1+2\eta)\phi_{\text{fp}}\right) \\ &= \varPhi\left(\phi_{\text{fn}} - 2\eta\phi_{\text{fp}}\right).\end{aligned}$$

It is easy to check by taking the derivative that $\varPhi$ satisfies the following inequality for all $y \geq 0$:

$$\varPhi(y) \geq 1 - \frac{e^{-y^2/2}}{\sqrt{\pi}}.$$

Furthermore, $\varPhi$ is an increasing function so $\varPhi^{-1}$ is also increasing. Hence,

$$\varPhi^{-1}(1-x) \leqslant \sqrt{-2\ln(\pi x)}$$

for all $x \leqslant \frac{1}{\sqrt{\pi}}$. Now recall that

$$\phi_{\text{fp}} = \varPhi^{-1}\left(1 - \frac{\mu}{2N_{\text{enum}}(\mathbf{s}_{\text{enum}})p^{k_{\text{fft}}}}\right), \qquad \phi_{\text{fn}} = \varPhi^{-1}\left(1 - \frac{\mu}{2}\right).$$

Therefore,

$$\phi_{\text{fp}} \leqslant \sqrt{-2\ln\frac{\pi\mu}{2N_{\text{enum}}(\mathbf{s}_{\text{enum}})p^{k_{\text{fft}}}}}.$$

From this we get that ϕ_{fp} is a polynomial factor in all the relevant parameters. Now observe that by the integral definition of $\varPhi$,

$$\begin{aligned}\varPhi\left(\phi_{\text{fn}} - 2\eta\phi_{\text{fp}}\right) &= \varPhi\left(\phi_{\text{fn}}\right) - \frac{1}{\sqrt{2\pi}}\int_{\phi_{\text{fn}}-2\eta\phi_{\text{fp}}}^{\phi_{\text{fn}}} e^{-t^2/2}\mathrm{d}t \\ &\geq 1 - \frac{\mu}{2} - \frac{2\eta}{\sqrt{2\pi}}\phi_{\text{fp}} \\ &\geq 1 - \frac{3\mu}{4}\end{aligned}$$

when

$$\eta \leqslant \frac{\sqrt{2\pi}\mu}{8\phi_{\text{fp}}}.$$

Therefore, by Theorem 6, with probability at least $1-\frac{3\mu}{4}$, we have that $S^L_{(1+2\eta)C} \neq \varnothing$ so the algorithm returns an element from S^L_C. Furthermore, by Equation (1), with probability at least $1-\mu$, this element must be $\mathbf{s}_{\text{enum}}$ because all the other elements satisfy $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) \leqslant C$. Therefore, by a union bound, the probability that the algorithm returns $\mathbf{s}_{\text{enum}}$ is at least $1 - \frac{3\mu}{4} - \mu \geq 1 - 2\mu \geq 1 - \nu$. This concludes the proof. $\square$

Lemma 4. *Let* $(n, m, q, \chi_s, \chi_e)$ *be* LWE *parameters,* $(\beta_0, \beta_1, k_{enum}, k_{fft}, k_{lat}, p)$ *be a partial tuple of parameters for Algorithm 3, and let* $0 < \nu < 1$. *Let* $\eta \leqslant \frac{\sqrt{2\pi\mu}}{8\phi_{fp}}$, *and* C, D *as in Lemma 2 with* $\mu = \nu/2$ *and replace* $2^{k_{enum}\cdot H(\chi_s)}$ *in the formula by* $G(\chi_s^{k_{enum}})$. *Then, with probability at least* $1-\nu$ *the algorithm returns* $\mathbf{s}_{enum}$.

Proof. The proof is exactly the same except that we replace the inequality

$$\mathbb{E}[N_{\text{enum}}(\mathbf{s}_{\text{enum}})] \leqslant 2^{k_{\text{enum}} H(\chi_s)}$$

by

$$\mathbb{E}[N_{\text{enum}}(\mathbf{s}_{\text{enum}})] = G(\chi_s^{k_{\text{enum}}}).$$

The reason for this replacement is that the first inequality does not appear to be justified in [MAT22] and does not hold in general. See Section 4 for more details.

Theorem 7. *Let* $(n, m, q, \chi_s, \chi_s)$ *be* LWE *parameters,* $(\beta_0, \beta_1, k_{enum}, k_{fft}, k_{lat}, p)$ *be a partial tuple of parameters for Algorithm 3, and let* $0 < \nu < 1$. *Choosing the parameters* C, D *according to Lemma 2, Algorithm 3 outputs* $\mathbf{s}_{enum}$ *with probability at least* $1-\nu$ *in time*

$$O\left(\left\lceil \frac{D}{(\sqrt{4/3})^{\beta_1 + o(\beta_1)}} \right\rceil \cdot (T_{BKZ}(d, \beta_0) + T_{sieve}(\beta_1)) + G^{qc}(\chi_s^{k_{enum}}) \cdot p^{k_{fft}/2} \cdot \sqrt{D}\right)$$

Proof. The time complexity is clear as we need $O(G^{qc}(\chi_s^{k_{\text{enum}}}))$ time to guess the correct value of $\mathbf{s}_{\text{enum}}$ by Lemma 5, $O(p^{k_{\text{fft}}/2})$ times to call the oracle $\hat{\mathcal{O}}$. Each call to $\hat{\mathcal{O}}$ makes $O(\varepsilon^{-1}\log\frac{1}{\delta})$ queries to $\mathcal{O}'_W$ (defined inside $\hat{\mathcal{O}}(\tilde{\mathbf{s}}_{\text{fft}})$) which take a polynomial time. Therefore each call to $\hat{\mathcal{O}}$ takes $\sqrt{D}$ times. Hence, the total complexity is (up to constant asymptotic factors)

$$G^{qc}(\chi_s^{k_{\text{enum}}}) \cdot p^{k_{\text{fft}}/2} \left(\frac{C}{D}\eta\right)^{-1} \log(10) = G^{qc}(\chi_s^{k_{\text{enum}}}) \cdot p^{k_{\text{fft}}/2} \left(\phi_{\text{fp}}\sqrt{D_{\text{arg}}}\eta\right)^{-1} \log(10)$$

where $D_{\text{arg}} \approx 1/2$, and ϕ_{fp} and η are polynomial factors. Hence, the complexity, up to polynomials factors is

$$G^{qc}(\chi_s^{k_{\text{enum}}}) \cdot p^{k_{\text{fft}}/2} \cdot \sqrt{D}.$$

This completes the proof. $\square$

4 Quantum Guessing

Let X be a random variable on a finite or countable set. We consider the problem of guessing the value taken by X by asking questions of the form "Is X equal to x?" until the answer is yes. This problem arises when we must find the secret $\mathbf{s}_{\text{enum}}$ in the dual attack by asking the question "is the secret equal to $\tilde{\mathbf{s}}_{\text{enum}}$?". Let N be the number of guesses used in the guessing strategy that minimises $\mathbb{E}[N]$. It is clear that the best strategy is to try values of X is decreasing order of probability. Without loss of generality, we can identify the possible values of X with $\mathbb{N}$ in such a way that $p_0 \geq p_1 \geq p_2 \geq \cdots$ where $p_i = \Pr[X = i]$. The expected number of guesses of the optimal strategy is therefore

$$G(X) = \sum_i i \cdot p_i.$$

It is well-known that a *lower bound* on $G(X)$ is given by the entropy of X. More precisely, Massey showed in [Mas94] that

$$G(X) \geq \tfrac{1}{4} \cdot 2^{H(X)} + 1$$

provided that $H(X) \geq 2$ bits, where H denote Shannon's entropy (i.e. in base 2). On the other hand, the same paper shows that it is not, in general, possible to bound $G(X)$ in terms of $H(X)$ only. In Lemma 6, we heuristically show that $G(X) \approx (\frac{2}{\sqrt{e}})^n \cdot 2^{H(X)}$ when X is distributed according to a n-dimensional discrete Gaussian. In this paper, we are interested in the *quantum complexity* of guessing. It can be shown (Lemma 5) that the expected number of guess in this case becomes, up to a constant factor,

$$G^{qc}(X) = \sum_i \sqrt{i} \cdot p_i.$$

Lemma 5. *Let X be a random variable taking values in some (effectively describable) set E. Assume that there is an efficiently computable bijective function $\sigma : \mathbb{N} \to E$ such that for all $i \leqslant j$, $\Pr_X[X = \sigma(i)] \geqslant \Pr_X[X = \sigma(j)]$,* i.e. *$\sigma$ orders E by decreasing probability according to X. Given $x \in E$, define the oracle $\mathcal{O}_x$ by*

$$\mathcal{O}_x(x) = 1 \qquad \text{and} \qquad \mathcal{O}_x(y) = 0, \quad \forall y \in E \setminus \{\, x \,\}\,.$$

Then there is a quantum algorithm $\mathcal{A}$, with quantum oracle access to σ and $\mathcal{O}_x$ such that for all $x \in E$, $\mathcal{A}^{\sigma, \mathcal{O}_x}() = x$ with constant probability, and

$$\mathbb{E}_X[T(X)] = O(G^{qc}(X)), \qquad \mathbb{E}_X[Q(X)] = O(G^{qc}(X)),$$

where $T(x)$ is the running time complexity of $\mathcal{A}^{\sigma, \mathcal{O}_x}()$, and $Q(x)$ its query complexity.

Proof. Consider the following algorithm, assuming oracle access to σ and $\mathcal{O}_x$ (for some unknown x). It first builds the oracle $\mathcal{O}'(i) = \mathcal{O}_x(\sigma(i))$. It then sets

$n = 1$ and repeats the following until success. Call the algorithm from Lemma 1 to find the first i such that $\mathcal{O}'(i) = 1$. If none is found, double n and repeat. Otherwise return i.

It is not hard to see that this algorithm returns x with constant probability. Indeed, σ is surjective so there exists i such that $\sigma(i) = x$ and then $\mathcal{O}'(i) = 1$. Since, σ is injective, $\sigma(j) \neq x$ for $j \neq i$ so $\mathcal{O}'(j) = 0$. Hence, when the algorithm has doubled n sufficiently many times, we have $i < n$ and the algorithm finds i with constant probability.

We now analyse its complexity. Let i be such that $\sigma(i) = x$. Let p be such that $2^p < i \leqslant 2^{p+1}$. During the first p iterations of the algorithm, we have $n \leqslant 2^p$ so $\mathcal{O}'(j) = 0$ for all $j \in [1, 2^p]$, hence the algorithm from Lemma 1 returns $\perp$ so the algorithm continues. Each of those p calls has time/query complexity $O(\sqrt{n})$, so overall this phase has complexity

$$O\left(\sum_{k=1}^{p} \sqrt{2^k}\right) = O(\sqrt{2^p}).$$

On the $(p+1)^{th}$ iteration, the algorithm will return i with constant probability (say 9/10) and has time/query complexity $O(\sqrt{2^{p+1}})$. If the algorithm fails (which happens with probability 1/10), the algorithm will run again on $[1, 2^{p+2}]$. It then will, with probability 9/10, return i and has time/query complexity $O(\sqrt{2^{p+2}})$. More generally, at the $(p+1+k)^{th}$ iteration, which only happen with probability $(1/10)^k$, the algorithm will return i with probability at least 9/10 and has time/query complexity $O(\sqrt{2^{p+k+1}})$. Hence, this part of the algorithm has expected time/query complexity

$$O\left(\sum_{k=0}^{\infty} 10^{-k} \frac{9}{10} \sqrt{2^{p+k+1}}\right) = O\left(\sqrt{2^{p+1}} \sum_{k=0}^{\infty} 10^{-k} \sqrt{2^k}\right) = O\left(\sqrt{2^{p+1}}\right) = O\left(\sqrt{i}\right).$$

Now recall that this analysis holds when the algorithm is called with an oracle $\mathcal{O}_x$ for a given x. We now let x be chosen by X. Then every x is chosen with probability $\Pr_X[X = x]$ and, when this is the case, the returned index is $i = \sigma^{-1}(x)$. Hence, the expected time/query complexity of the algorithm when given $\mathcal{O}_X$ is

$$O\left(\sum_{x \in E} \Pr_X[X = x] \sqrt{\sigma^{-1}(x)}\right) = O\left(\sum_{i=0}^{\infty} \Pr_X[X = \sigma(i)] \sqrt{i}\right) = O(G^{qc}(X)).$$

since we assumed that σ orders the elements by decreasing probability. $\square$

Now we study the guessing complexity of a n-dimensional discrete Gaussian and its modular version. We also related those quantities to the entropy. The reason why we also study the (non-modular) Gaussian is that it is not clear how to order the elements by decreasing probability in the modular case, whereas it is easy in the non-modular one. Therefore we study the discrete Gaussian first and then show that its guessing complexity is an upper bound on the guessing complexity of the modular Gaussian.

Lemma 6. *Let $n \in \mathbb{N}$ large, $\sigma \geqslant 0.5$ and $X = D_{\mathbb{Z}^n,\sigma}$. Then*

$$G(X) \approx \frac{n! \cdot (\sigma\sqrt{2\pi})^n}{2\,\Gamma(\frac{n}{2}+1)^2}, \quad H(X) \approx n\log_2(\sigma\sqrt{2e\pi}), \quad G(X) \approx \frac{n! \cdot 2^{H(X)} \cdot e^{-n/2}}{2\,\Gamma(\frac{n}{2}+1)^2}$$

and

$$G^{qc}(X) \approx \frac{n \cdot (2\pi\sigma^2)^{n/4} \cdot \Gamma(\frac{3n}{4})}{2\,\Gamma(\frac{n}{2}+1)^{3/2}}.$$

Furthermore,

$$G(X) \sim_{n\to\infty} \frac{1}{\sqrt{2\pi n}} \cdot \left(\frac{2}{\sqrt{e}}\right)^n \cdot 2^{H(X)}, \quad G^{qc}(X) \sim_{n\to\infty} \frac{\sqrt{6}}{3(\pi n)^{\frac{1}{4}}} \cdot \left(\frac{27}{8e}\right)^{\frac{n}{4}} \cdot 2^{H(X)/2}.$$

Finally, for all n and σ,

$$G(D_{\mathbb{Z}_q^n,\sigma}) \leqslant 2\,G(X), \qquad G^{qc}(D_{\mathbb{Z}_q^n,\sigma}) \leqslant \frac{3}{2}\,G^{qc}(X).$$

Proof. See Appendix A.

5 Application

In this section we measure the impact of our algorithm on the cost of solving lattice parameters from the literature. In particular, we consider NIST PQC Round 3 candidates Kyber and Saber [SAB$^+$20,DKR$^+$20] and some TFHE parameters [CGGI20]. Such a measurement is complicated by two major obstacles.

- The cost given in Theorem 7 is the sum of two costs: lattice reduction and a quantum search. Roughly speaking, lowering the first summand increases the second and vice versa. In other words, the final cost is obtained by balancing the two summands. For the first summand cost estimates in various cost models are available. In particular, estimates in quantum circuit models are available [AGPS20]. Thus, to give precise cost estimates we require quantum circuit costs for the oracles in Algorithm 3. It is clear that such costs would be substantial when compared with e.g. [AGPS20]. In the latter, the costed circuit is essentially an XOR followed by an adder. Here, we have to implement matrix vector products mod q which will cost significantly more. We consider designing and costing quantum circuits for these elementary operations beyond the scope of this paper. For this reason, we cost our algorithm in the quantum query model only, both for the oracle inside lattice reduction and our search. In this model, all oracle queries are assigned unit cost. As just outlined, this is unrealistic but gives a "best case" estimate from the perspective of an attacker.
- A second major obstacle is that our algorithm critically relies on QRACM, a possibly unrealistic resource as already pointed out in e.g. [AGPS20]. Thus, even armed with a quantum circuit for our oracle, we would have to assume

Scheme	CC	CN	C0	GE19	QN	Q0	This work (QN)	This work (Q0)
Kyber 512	139.1	134.3	115.2	139.4	124.1	102.4	118.6	94.2
Kyber 768	195.8	191.1	173.5	191.7	174.9	154.5	167.8	140.7
Kyber 1024	262.1	255.9	241.4	251.7	234.3	214.8	224.9	195.0
LightSaber	138.2	133.0	113.4	138.2	122.6	101.0	112.1	93.3
Saber	201.6	195.6	178.9	196.1	179.6	159.3	173.3	146.1
FireSaber	264.3	257.9	243.5	253.5	235.6	216.7	213.8	197.6
TFHE630	117.3	112.8	92.8	119.5	104.9	83.0	95.0	76.3
TFHE1024	121.9	117.1	95.4	123.5	108.4	84.8	101.2	80.0

Table 1. Dual attack cost estimates. All costs are logarithms to base two.

a QRACM oracle (for which we, following previous work [AGPS20], assign unit cost for querying). This would not permit us to draw conclusions about realistic costs of solving instances of lattice problems.

We give the source code for and results of the comparison in Appendix B and Table 1. In our table, for each set of parameters, we give the following cost estimates.

CC Classical cost estimates in a classical circuit model [AGPS20,SAB$^+$20,MAT22] for Algorithm 2 using [BDGL16] as the sieving oracle. We derive these estimates by implementing the cost estimates from [MAT22].[5] This is the most detailed cost estimate available in the literature. However, we caution that these estimates, too, ignore the cost of memory access and thus may significantly underestimate the true cost. That is, while RAM access is expected to be considerably cheaper than QRACM it is still not "free", cf. [MAB$^+$22]. This cost model is called "list_decoding-classical" in [AGPS20]. We naturally do not cost our algorithm in this cost model.

CN Classical cost estimates in a query model for Algorithm 2 using [BDGL16] as the sieving oracle. We include this cost model for completeness and for interpreting our quantum query cost model estimates. This cost model is called "list_decoding-naive_classical" in [AGPS20]. We naturally do not cost our algorithm in this cost model.

C0 Classical cost estimates in the "Core-SVP" cost model [ADPS16] for Algorithm 2 using [BDGL16] as the sieving oracle. This model assumes a single SVP call suffices to reduce a lattice. It furthermore assumes that all lower-order terms in the exponent are zero. This is to enable comparison with "Q0" below.

GE19 Quantum costs in a circuit model based on [GE19] for Algorithm 2 using [BDGL16]. This is the most detailed quantum cost model available in the literature but we recall that here we still assume unit cost QRACM. This

[5] This explains the minor differences in numerical results compared to [MAT22]. In particular, we have an additional exponential factor for the guessing complexity, cf. Lemma 6.

cost model is called “list_decoding-ge19” in [AGPS20]. We do not cost our algorithm in this cost model due to the lack of a quantum circuit design for our oracles.

QN Quantum costs in the quantum query model for Algorithm 2 using the quantum version of [BDGL16] as the sieving oracle. This cost model is called “list_decoding-naive_quantum” in [AGPS20].

Q0 Quantum cost estimates in the “Core-SVP” cost model [ADPS16] for Algorithm 2 using [CL21] as the sieving oracle. This is the asymptotically fastest quantum sieving algorithm but no estimates exist in the literature for lower-order terms; hence, we only consider it in the Core-SVP model.

This work (QN) The cost of Algorithm 3 in the quantum query model assuming the quantum version [Laa15,AGPS20] of [BDGL16]. Thus, the most natural comparison is to the column labelled “CN”.

This work (Q0) The cost of Algorithm 3 in the Core-SVP model assuming [CL21]. Thus, the most natural comparison is to the column labelled “C0”.

On the one hand, comparing the column labelled “QN” and the last column shows that our algorithm offers a significant improvement of between 10 and 20 “bits” in complexity in the query model. On the other hand, even in this – arguably unrealistic – model our improvements do not lower the cost of solving below a square-root of the targeted security level. That is, to force a revision of lattice parameters, a quantum algorithm would have to obtain a quadratic speed-up over the classical cost given as “CN”.

6 Open Problem

The crux of our quantum improvement is Section 3. Here we formalise the problem that this algorithm solves and a promise variant. We introduce some minor notation first. Given a finite group $G = \mathbb{Z}_q^n$ and a list

$$L = \{(\mathbf{u}_0, w_0), \ldots, (\mathbf{u}_{k-1}, w_{k-1})\}$$

where the $\mathbf{u}_i$ are distinct, we let $f_L : G \to \mathbb{C}$ be defined by $f_L(\mathbf{u}_i) = w_i$ and $f_L(\mathbf{u}) = 0$ for all $\mathbf{u} \in G \setminus \{\, \mathbf{u}_0, \ldots, \mathbf{u}_{k-1} \,\}$. Recall that $\widehat{f_L}$ denotes the Fourier transform of f_L. We now introduce two problems, which we call “input sparse FFT” to avoid confusion with “sparse FFT” where the sparseness refers to the number of nonzero Fourier coefficient, not the number of nonzero inputs coefficients.

INPUT-SPARSE-FFT-THRESHOLD:

- **input:** $G = \mathbb{Z}_q^n$ a finite group,
- **input:** $\delta > 0$ a threshold,
- **input:** $L = \{(\mathbf{u}_0, w_0), \ldots, (\mathbf{u}_{k-1}, w_{k-1})\}$ where the $\mathbf{u}_i$ are distinct,
- **output:** decide whether $\exists \mathbf{u} \in G$ such that $\Re(\widehat{f_L}(\mathbf{u})) > \delta$, or $\perp$ if none exists.

PROMISE-INPUT-SPARSE-FFT-THRESHOLD:

- **input:** $G = \mathbb{Z}_q^n$ a finite group,
- **input:** $\delta^+ > \delta^- > 0$ two thresholds,
- **input:** $f : G \to \mathbb{C}$ an efficiently computable function,
- **input:** $L = \{(\mathbf{u}_0, w_0), \ldots, (\mathbf{u}_{k-1}, w_{k-1})\}$ where the $\mathbf{u}_i$ are distinct,
- **promise:** $\Re(\widehat{f_L}(\mathbf{u})) \notin [\delta^-, \delta^+]$ for all $\mathbf{u} \in G$,
- **output:** decide whether $\exists \mathbf{u} \in G$ such that $\Re(\widehat{f_L}(\mathbf{u})) > \delta^+$, or $\perp$ if none exists.

Remark 1. To map this formulation back to our task consider Line 9 of Algorithm 2. The $\mathbf{u}_i$ correspond to $\lfloor \frac{p}{q} \cdot \mathbf{y}_{j,\text{fft}} \rceil$ and $w_i := f_L(\mathbf{u}_i)$ is the sum over all $\exp\left((\mathbf{x}_j^T \cdot \mathbf{b} - \mathbf{y}_{j,\text{enum}}^T \cdot \tilde{\mathbf{s}}_{\text{enum}}) \cdot \frac{2i\pi}{q}\right)$ that are stored in the cell $\lfloor \frac{p}{q} \mathbf{y}_{j,\text{fft}} \rceil$ of T, i.e. $w_i = T\left[\lfloor \frac{p}{q} \mathbf{y}_{j,\text{fft}} \rceil\right] = f_L(\mathbf{u}_i)$. We then seek to decide if there is some $\mathbf{u} = \tilde{\mathbf{s}}_{\text{fft}} \in G = \mathbb{Z}_p^{k_{\text{fft}}}$ s.t. $\Re(\widehat{f_L}(\mathbf{u})) > \delta = C$, i.e. the entry in the FFT'd table T.

Our quantum (with QRACM) algorithm from Section 3 solve PROMISE-INPUT-SPARSE-FFT-THRESHOLD as follows. For every $\mathbf{u} \in G$, it compute an approximation of $\Re(\widehat{f_L}(\mathbf{u}))$ with error at most $\frac{1}{2}(\delta^+ - \delta^-)$ and then compare it to δ^+. By the promise, this suffices to solve the problem. We then leverage two facts to obtain a quantum speedup: the search over $\mathbf{u} \in G$ can be done using Grover's algorithm, and the approximation is done by amplitude estimation (Theorem 5). The running time of our algorithm is $\sqrt{|G|}/(\delta^+ - \delta^-)$, and it outputs a correct index with constant probability. In the dual algorithm, it turns out that the interesting set of parameters for this algorithm is $\delta^+ - \delta^- = O(k^{-1/2})$, therefore our algorithm has running time roughly $O(\sqrt{k|G|})$ which is always better than $O(|G|)$ and potentially much better if k is much smaller than $|G|$.

In the classical case, to the best of our knowledge, the best algorithm is to perform a complete FFT on the $|G|$ coefficients, which therefore takes time $O(|G| \log |G|)$. While there are algorithms for "sparse" FFT (see e.g. [HIKP12]), it is not clear that their approximation guarantees would be sufficient. Indeed, the sparseness in such algorithm refers to the number of output coefficients $\widehat{f_L}(\mathbf{u})$ which is assumed small. Since we expect all the output coefficients of our FFT to be small and the threshold δ to be exponentially close to 0, it is unlikely that such an approximation would be sufficient.

In the quantum case, the situation depends on the availability of quantum memories (QRACM). Our algorithm relies on the use of a QRACM in a crucial way. In fact, without a QRACM, we are not aware of any algorithm better than the classical one. This is surprising in light of the fact that QFT can be done in polynomial time: we now explain why this fact alone is not sufficient.

Let $L = \{(\mathbf{u}_0, w_0), \ldots, (\mathbf{u}_{k-1}, w_{k-1})\}$ be a list. In order to apply the QFT, we would need to create the superposition

$$|\psi\rangle = \frac{1}{Z} \sum_i w_i \, |\mathbf{u}_i\rangle \tag{2}$$

where Z is a normalisation factor. We could then perform a QFT on $|\psi\rangle$ to obtain

$$|\widehat{\psi}\rangle = \frac{1}{Z} \sum_{\mathbf{u} \in G} \widehat{f_L}(\mathbf{u}) \, |\mathbf{u}\rangle \, .$$

We now would like to decide if there is some $\mathbf{u} \in G$ such that $\widehat{f_L}(\mathbf{u}) > \delta$. Unfortunately, there are two problems with this approach:

– it is not clear how to create the superposition in Equation (2) efficiently,
– it is not clear that we can efficiently detect whether there is an amplitude in front of some $|\mathbf{u}\rangle$ which is above the threshold.

The first problem is the most serious one: while we can create the superposition Equation (2) in time $O(k)$, any algorithm for the second step would probably need to repeat this step many times (see below). This would make the algorithm essentially worse than the classical one. With the use of a QRACM and some elementary operations, we could create a superposition of the form

$$\sum_i w_i \, |\mathbf{u}_i\rangle \, |w_i\rangle$$

in time $O(1)$ after the QRACM is created (which takes time $O(k)$ once). However note that we cannot apply the QFT on this state: we first need to "uncompute" $|w_i\rangle$ from the state to obtain Equation (2), which is not possible in general.

The second problem may be less serious: if we could create Equation (2), a possible strategy would be to measure $|\widehat{\psi}\rangle$ and obtain one $\mathbf{u}$. By repeating this algorithm a very large number of times, we can approximation the probability of the most likely $\mathbf{u}$ and therefore recover whether there is some sufficiently large $\widehat{f_L}(\mathbf{u})$. Indeed, this strategy recovers $\mathbf{u}$. In order to approximate this quantity within ε, we would need $1/\varepsilon^2$ samples. Since $\varepsilon = 1/\sqrt{k}$ in the dual attack, such an algorithm would take time at least $\Omega(k)$. It is not clear if there is a better strategy that merely decides on the presence of some $\mathbf{u}$ without recovering it.

In conclusion, the complexity of solving `INPUT-SPARSE-FFT-THRESHOLD` and `PROMISE-INPUT-SPARSE-FFT-THRESHOLD` is unclear in the quantum setting. We have shown how to solve the promise problem in $O(\sqrt{|G|/(\delta^+ - \delta^-)})$ with QRACM, and $\Omega(k)$ is a clear lower bound on the complexity since the algorithm needs to read the input in any case. Of particular relevance in the context of dual attacks are the following two questions regarding `PROMISE-INPUT-SPARSE-FFT-THRESHOLD`:

– When $(\delta^+ - \delta^-)^{-1} = \Theta(\sqrt{k})$, is the quantum complexity $O(\sqrt{k|G|})$ optimal with QRACM?

– When $(\delta^+ - \delta^-)^{-1} = \Theta(\sqrt{k})$, can we achieve any quantum complexity better than $O(|G| \log |G|)$ without QRACM?

Acknowledgements

We thank André Chailloux, Yanlin Chen, Thomas Debris-Alazard, Yassine Hamoudi, Amaury Pouly, Jean-Pierre Tillich for helpful discussions. The research of MA was supported by EPSRC grants EP/S020330/1 and EP/S02087X/1, and by the European Union Horizon 2020 Research and Innovation Program Grant 780701. The research of YS was supported by EPSRC grant EP/S02087X/1 and EP/W02778X/1.

References

ACKS20. Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, and Yixin Shen. Improved classical and quantum algorithms for the shortest vector problem via bounded distance decoding, 2020.

ACPS09. Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, Heidelberg, August 2009.

ADPS16. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

AFFP14. Martin R. Albrecht, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Lazy modulus switching for the BKW algorithm on LWE. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 429–445. Springer, Heidelberg, March 2014.

AGPS20. Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 583–613. Springer, Heidelberg, December 2020.

Alb17. Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 103–129. Springer, Heidelberg, April / May 2017.

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

BDGL16. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24. ACM-SIAM, January 2016.

Ben89. Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, 1989.

BHMT02a. Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

BHMT02b. Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Information*, page 53–74, 2002.

BPR12. Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737. Springer, Heidelberg, April 2012.

BV11. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

CGGI20. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.

CL21. André Chailloux and Johanna Loyer. Lattice sieving via quantum random walks. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 63–91. Springer, 2021.

DKR$^+$20. Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

EJK20. Thomas Espitau, Antoine Joux, and Natalia Kharchenko. On a dual/hybrid approach to small secret LWE - A dual/enumeration technique for learning with errors and application to security estimates of FHE schemes. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 440–462. Springer, Heidelberg, December 2020.

GE19. Craig Gidney and Martin Ekerå. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits, 2019. arXiv:1905.09749.

GJ21. Qian Guo and Thomas Johansson. Faster dual lattice attacks for solving LWE with applications to CRYSTALS. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 33–62. Springer, 2021.

GLM08. Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, Apr 2008.

Gro96. Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

HIKP12. Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse fourier transform. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 563–578, New York, NY, USA, 2012. Association for Computing Machinery.

HLGJ20. C. Hann, G. Lee, S. Girvin, and Liang Jiang. The resilience of quantum random access memory to generic noise. *arXiv: Quantum Physics*, 2020.

HMdW03. Peter Høyer, Michele Mosca, and Ronald de Wolf. Quantum search on bounded-error inputs. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming*, pages 291–299, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

JNRV20. Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and LowMC. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 280–310. Springer, Heidelberg, May 2020.

KKM$^+$21. Ruslan Kapralov, Kamil Khadiev, Joshua Mokut, Yixin Shen, and Maxim Yagafarov. Fast classical and quantum algorithms for online k-server problem on trees. In Claudio Sacerdoti Coen and Ivano Salvo, editors, *Proceedings of the 22nd Italian Conference on Theoretical Computer Science, Bologna, Italy, September 13-15, 2021*, volume 3072 of *CEUR Workshop Proceedings*, pages 287–301. CEUR-WS.org, 2021.

KP20. Iordanis Kerenidis and Anupam Prakash. Quantum gradient descent for linear systems and least squares. *Phys. Rev. A*, 101:022316, Feb 2020.

Kup13. Greg Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In Simone Severini and Fernando G. S. L. Brandão, editors, *8th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2013, May 21-23, 2013, Guelph, Canada*, volume 22 of *LIPIcs*, pages 20–34. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

Laa15. Thijs Laarhoven. *Search problems in cryptography: From fingerprinting to lattice sieving.* PhD thesis, Eindhoven University of Technology, 2015.

LMv13. Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Solving the shortest vector problem in lattices faster using quantum search. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, pages 83–101. Springer, Heidelberg, June 2013.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.

LS90. Robert Y. Levin and Alan T. Sherman. A note on bennett's time-space tradeoff for reversible computation. *SIAM J. Comput.*, 19(4):673–677, 1990.

MAB$^+$22. Matzov, Daniel Apon, Daniel J. Bernstein, Carl Mitchell, Léo Ducas, Martin Albrecht, and Chris Peikert. Improved Dual Lattice Attack. `https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Fm4cDfsx65s`, 2022.

Mas94. J.L. Massey. Guessing and entropy. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, pages 204–, 1994.

MAT22. MATZOV. Report on the Security of LWE: Improved Dual Lattice Attack, April 2022.

MGM20. O. D. Matteo, V. Gheorghiu, and M. Mosca. Fault-tolerant resource estimation of quantum random-access memories. *IEEE Transactions on Quantum Engineering*, 1:1–13, 2020.

Mon15. Ashley Montanaro. Quantum speedup of monte carlo methods. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2181):20150301, sep 2015.

MR09. Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer, Heidelberg, Berlin, Heidelberg, New York, 2009.

NC11. Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, USA, 10th edition, 2011.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.

SAB$^+$20. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

SSTX09. Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 617–635. Springer, Heidelberg, December 2009.

Appendix

A Proof of Lemma 6

First observe that $D_{\mathbb{Z}^n,\sigma}(\mathbf{x})$ only depends on $\|\mathbf{x}\|$ and decreases with $\|\mathbf{x}\|$. Formally, $D_{\mathbb{Z}^n,\sigma}(\mathbf{x}) = \rho_\sigma(\|\mathbf{x}\|)/\rho_\sigma(\mathbb{Z}^n)$. Therefore, we can rewrite the guessing complexity as

$$G(D_{\mathbb{Z}^n,\sigma}) = \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \sum_{i=N(\ell-1)+1}^{N(\ell)} i$$

where $N(\ell) = \{\, \mathbf{x} \in \mathbb{Z}^n : \|\mathbf{x}\| \leqslant \sqrt{\ell}\,\}$. We let $N(-1) = 0$ by convention. It follows that, for $n \geqslant 4$ (we need every number to be a sum of n squares so that $N(\ell) > N(\ell-1)$):

$$\begin{aligned} G(X) &= \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \cdot \frac{(N(\ell)-N(\ell-1))\cdot(N(\ell-1)+N(\ell)+1)}{2} \qquad (3)\\ &\leqslant \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \cdot (N(\ell)-N(\ell-1))\cdot N(\ell) \\ &= \frac{1}{\rho_\sigma(\mathbb{Z}^n)} \sum_{\ell=0}^{\infty} \rho_\sigma(\sqrt{\ell})\cdot(N(\ell)-N(\ell-1))\cdot N(\ell). \end{aligned}$$

By a standard estimate[6], we have that (this only works for $n \geqslant 3$):

$$N(\ell) = \mathrm{vol}(B_n)\cdot \ell^{n/2} + O(\ell^{n/2-1})$$

where $B_n := B_n(1)$ and $B_n(r)$ denotes the unit ball of $\mathbb{R}^n$ for the Euclidean norm. Therefore, for $\ell \geqslant 1$,

$$\begin{aligned} N(\ell)-N(\ell-1) &= \mathrm{vol}(B_n)\cdot(\ell^{n/2}-(\ell-1)^{n/2}) + O(\ell^{n/2-1}) \\ &= \mathrm{vol}(B_n)\cdot\frac{n}{2}\cdot\ell^{n/2-1} + O(\ell^{n/2-1}) \\ &= (\mathrm{vol}(B_n)+O(1))\cdot\frac{n}{2}\cdot\ell^{n/2-1}. \end{aligned}$$

We note that the approximation above only depends on the dimension and not on σ. We now obtain that

$$\begin{aligned} G(D_{\mathbb{Z}^n,\sigma}) &\lesssim \frac{1}{\rho_\sigma(\mathbb{Z}^n)} + \frac{\mathrm{vol}(B_n)^2}{\rho_\sigma(\mathbb{Z}^n)} \cdot \sum_{\ell=1}^{\infty} \rho_\sigma(\sqrt{\ell})\cdot\frac{n}{2}\cdot\ell^{n-1} \\ &= \frac{1}{\rho_\sigma(\mathbb{Z}^n)} + \frac{n\cdot\mathrm{vol}(B_n)^2}{2\rho_\sigma(\mathbb{Z}^n)} \cdot \sum_{\ell=1}^{\infty} \rho_\sigma(\sqrt{\ell})\cdot\ell^{n-1}. \end{aligned}$$

[6] Place a unit cube centred on each integer point: the resulting body contains $B_n(\sqrt{\ell}-\sqrt{n}/2)$ and is contained in $B_n(\sqrt{\ell}+\sqrt{n}/2)$.

We can approximate the sum by an integral and get that

$$\begin{aligned}\sum_{\ell=1}^{\infty} \rho_\sigma(\sqrt{\ell}) \cdot \ell^{n-1} &\approx \int_1^\infty \rho_\sigma(\sqrt{t}) \cdot t^{n-1}\,\mathrm{d}t \\ &\leqslant \int_0^\infty e^{-\frac{t}{2\sigma^2}} \cdot t^{n-1}\,\mathrm{d}t \\ &= (n-1)! \cdot (2\sigma^2)^n.\end{aligned} \tag{4}$$

We also bound $\rho_\sigma(\mathbb{Z}^N)$ by using the Poisson summation formula as follows:

$$\rho_\sigma(\mathbb{Z}^n) = (\rho_\sigma(\mathbb{Z}))^n = \left(\sigma\sqrt{2\pi}\rho_{1/\sqrt{2}\pi\sigma}(\mathbb{Z})\right)^n > (\sigma\sqrt{2\pi})^n$$

which is a good approximation for large σ. Using the standard asymptotic (in n) estimate for $\mathrm{vol}(B_n)$, we have that

$$G(D_{\mathbb{Z}^n,\sigma}) \lesssim \frac{n!\,(2\sigma^2)^n}{2\cdot(\sigma\cdot\sqrt{2\pi})^n} \cdot \left(\frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)}\right)^2 = \frac{n!\,(2\pi\sigma^2)^{n/2}}{2\,\Gamma(\frac{n}{2}+1)^2}.$$

We now argue that this formula should be relatively accurate for large n and σ at least 1 (with a precise that increases as σ gets larger). Recall that the approximation of $N(\ell)$ only depend on the dimension n and not σ. Therefore, the only approximation step that depends on σ is (4) where we approximate the sum by an integral. Let $f(t) = \rho_\sigma(\sqrt{t}) \cdot t^{n-1}$. One easily checks that f is increasing on $[0, N]$ and decreasing on $[N, \infty]$ where $N = 2(n-1)\sigma^2$. Assume for simplicity that σ is an integer. Then,

$$\begin{aligned}\sum_{\ell=1}^{\infty} f(\ell) &= \sum_{\ell=1}^{N-1} f(\ell) f(N) + \sum_{N+1}^{\infty} f(\ell) \\ &\leqslant \sum_{\ell=1}^{N-1} \int_\ell^{\ell+1} f(t)\mathrm{d}t + f(N) + \sum_{N+1}^{\infty} \int_{\ell-1}^{\ell} f(t)\mathrm{d}t \\ &= f(N) + \int_1^\infty f(t)\mathrm{d}t \\ &\leqslant f(N) + \int_0^\infty f(t)\mathrm{d}t \\ &= f(N) + (n-1)! \cdot (2\sigma^2)^n.\end{aligned}$$

Similarly,

$$\begin{aligned}\sum_{\ell=1}^{\infty} f(\ell) &= \sum_{\ell=1}^{N} f(\ell) - f(N) + \sum_{N}^{\infty} f(\ell) \\ &\geqslant \sum_{\ell=1}^{N} \int_{\ell-1}^{\ell} f(t)\mathrm{d}t - f(N) + \sum_{N}^{\infty} \int_{\ell}^{\ell+1} f(t)\mathrm{d}t\end{aligned}$$

$$
\begin{aligned}
&= -f(N) + \int_0^\infty f(t)\mathrm{d}t \\
&= -f(N) + (n-1)! \cdot (2\sigma^2)^n .
\end{aligned}
$$

Hence, we see that the error introduced by the sum/integral approximation in (4) is $\pm f(N)$. This term is then divided by $\rho_\sigma(\mathbb{Z}^n)$ which is at least $(\sigma\sqrt{2\pi})^n$. Finally, this term should be compared to the approximation to obtain a relative error. Hence, the approximation's relative error is at most

$$
\begin{aligned}
\frac{\rho_\sigma(\sqrt{2(n-1)}\sigma) \cdot (2(n-1)\sigma^2)^{n-1}}{(\sigma\sqrt{2\pi})^n \cdot (n-1)!(2\sigma^2)^n} &= \frac{\rho_\sigma(\sqrt{2(n-1)}\sigma) \cdot (n-1)^{n-1}}{(\sigma\sqrt{2\pi})^n \cdot (n-1)!(2\sigma^2)} \\
&= \frac{e^{-(n-1)} \cdot (n-1)^{n-1}}{(\sigma\sqrt{2\pi})^n \cdot (n-1)!(2\sigma^2)} \\
&= \frac{1}{2\sigma(\sigma\sqrt{2\pi})^n (n-1)!} \left(\frac{n-1}{e}\right)^{n-1} \\
&\sim_{n\to\infty} \frac{1}{2\sigma(\sigma\sqrt{2\pi})^n} \frac{1}{\sqrt{2\pi(n-1)}} .
\end{aligned}
$$

Looking at this formula, we see that it is very close to 0 (for large n) as long as $\sigma\sqrt{2\pi} > 1$. Since $\sqrt{2\pi} \approx 2.5$, this means that any value of σ above 0.5 ensures a very good approximation.

For the quantum guessing complexity, we use the same approach to get that

$$
G^{qc}(D_{\mathbb{Z}^n,\sigma}) = \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \sum_{i=N(\ell-1)+1}^{N(\ell)} \sqrt{i}. \tag{5}
$$

We now note that

$$
\sum_{i=a+1}^{n} \sqrt{i} \approx \int_{a+1}^{b} \sqrt{t}\mathrm{d}t = \frac{2}{3}\left(b^{3/2} - (a+1)^{3/2}\right).
$$

This approximation can be made formal easily since the square root function is increasing, but for brevity we omit this step. Therefore,

$$
\begin{aligned}
G^{qc}(D_{\mathbb{Z}^n,\sigma}) &\approx \frac{1}{\rho_\sigma(\mathbb{Z}^n)} + \frac{2}{3}\sum_{\ell=1}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \cdot \left(N(\ell)^{3/2} - (N(\ell-1)+1)^{3/2}\right) \\
&\approx \frac{1}{\rho_\sigma(\mathbb{Z}^n)} + \frac{2}{3}\sum_{\ell=1}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \mathrm{vol}(B_n)^{3/2} \cdot \left(\ell^{3n/4} - (\ell-1)^{3n/4}\right) \\
&\approx \frac{1}{\rho_\sigma(\mathbb{Z}^n)} + \frac{2}{3}\sum_{\ell=1}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \cdot \mathrm{vol}(B_n)^{3/2} \cdot \frac{3n}{4} \cdot \ell^{3n/4-1} \\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)} + \frac{n \cdot \mathrm{vol}(B_n)^{3/2}}{2\rho_\sigma(\mathbb{Z}^n)} \cdot \sum_{\ell=1}^{\infty} \rho_\sigma(\sqrt{\ell}) \cdot \ell^{3n/4-1}.
\end{aligned}
$$

We can approximate the sum by an integral again and get that

$$\begin{aligned}\sum_{\ell=1}^{\infty}\rho_\sigma(\sqrt{\ell})\cdot\ell^{3n/4-1} &\approx \int_1^\infty \rho_\sigma(\sqrt{t})\cdot t^{3n/4-1}\,\mathrm{d}t\\ &\leqslant \int_0^\infty e^{-\frac{t}{2\sigma^2}}t^{3n/4-1}\,\mathrm{d}t\\ &= \left(2\sigma^2\right)^{3n/4}\cdot\Gamma\left(\tfrac{3n}{4}\right).\end{aligned}$$

One can analyse the sum/integral approximation as above and conclude that it is sound for σ above 0.5. Therefore,

$$\begin{aligned}G^{qc}(X) &\lesssim \frac{n\cdot\mathrm{vol}(B_n)^{3/2}}{2\rho_\sigma(\mathbb{Z}^n)}\cdot\left(2\sigma^2\right)^{3n/4}\cdot\Gamma\left(\tfrac{3n}{4}\right)\\ &\lesssim \frac{n}{2\left(\sigma\sqrt{2\pi}\right)^n}\cdot\left(\frac{\pi^{n/2}}{\Gamma\left(\frac{n}{2}+1\right)}\right)^{3/2}\cdot\left(2\sigma^2\right)^{3n/4}\cdot\Gamma\left(\tfrac{3n}{4}\right)\\ &= \frac{n\cdot\left(2\pi\sigma^2\right)^{n/4}\cdot\Gamma\left(\frac{3n}{4}\right)}{2\,\Gamma\left(\frac{n}{2}+1\right)^{3/2}}.\end{aligned}$$

Finally, we estimate the entropy of this distribution as follows:

$$\begin{aligned}H(D_{\mathbb{Z}^n,\sigma}) &= -\sum_{\mathbf{x}\in\mathbb{Z}^n} D_{\mathbb{Z}^n,\sigma}(\mathbf{x})\cdot\log_2(D_{\mathbb{Z}^n,\sigma}(\mathbf{x}))\\ &= \frac{1}{2\sigma^2\log(2)\cdot\rho_\sigma(\mathbb{Z})}\cdot\sum_{\mathbf{x}\in\mathbb{Z}^n}\rho_\sigma(\mathbf{x})\cdot\|\mathbf{x}\|^2+\frac{\log_2\rho_\sigma(\mathbb{Z}^n)}{\rho_\sigma(\mathbb{Z}^n)}\sum_{\mathbf{x}\in\mathbb{Z}^n}\rho_\sigma(\mathbf{x})\\ &= \frac{1}{2\sigma^2\log(2)\cdot\rho_\sigma(\mathbb{Z}^n)}\sum_{\mathbf{x}\in\mathbb{Z}^n}\rho_\sigma(\mathbf{x})\cdot\|\mathbf{x}\|^2+\log_2\rho_\sigma(\mathbb{Z}^n)\end{aligned}$$

By approximating the sum with the integral, we have that

$$\begin{aligned}H(D_{\mathbb{Z}^n,\sigma}) &\approx \frac{1}{2\sigma^2\cdot\log(2)\cdot\rho_\sigma(\mathbb{Z}^n)}\int_{\mathbb{R}^n}\rho_\sigma(\mathbf{x})\cdot\|\mathbf{x}\|^2\,\mathrm{d}\mathbf{x}+\log_2(\rho_\sigma(\mathbb{Z}^n))\\ &= \frac{n\cdot\sigma^2\cdot\left(\sigma\sqrt{2\pi}\right)^n}{2\cdot\sigma^2\log(2)\rho_\sigma(\mathbb{Z}^n)}+\log_2\rho_\sigma(\mathbb{Z}^n)\\ &\approx \frac{n\cdot\sigma^2\cdot\left(\sigma\sqrt{2\pi}\right)^n}{2\,\sigma^2\cdot\log(2)\cdot\left(\sigma\sqrt{2\pi}\right)^n}+\log_2(\sigma\sqrt{2\pi})^n\\ &= \frac{n}{2\log(2)}+n\log_2(\sigma\sqrt{2\pi})\\ &= n\log_2(\sigma\sqrt{2e\pi}).\end{aligned}$$

Hence, we have that, for large n,

$$G(D_{\mathbb{Z}^n,\sigma})\approx\frac{n!\left(\sigma\sqrt{2\pi}\right)^n}{2\,\Gamma\left(\frac{n}{2}+1\right)^2}$$

$$
\begin{aligned}
&\sim_{n\to\infty} \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{2\left(\sqrt{2\pi\frac{n}{2}}\left(\frac{n}{2e}\right)^{n/2}\right)^2}\cdot\left(\sigma\sqrt{2\pi}\right)^n\\
&\sim \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{2\pi n\left(\frac{n}{2e}\right)^n}\cdot\left(\sigma\sqrt{2\pi}\right)^n\\
&\sim \frac{2^n}{\sqrt{2\pi n}}\cdot\left(\sigma\sqrt{2\pi}\right)^n\\
&\sim \frac{1}{\sqrt{2\pi n}}\cdot\left(\frac{2}{\sqrt{e}}\right)^n\cdot\left(\sigma\sqrt{2e\pi}\right)^n\\
&\sim \frac{1}{\sqrt{2\pi n}}\cdot\left(\frac{2}{\sqrt{e}}\right)^n\cdot 2^{H(X)}
\end{aligned}
$$

and that

$$
\begin{aligned}
G^{qc}(D_{\mathbb{Z}^n,\sigma}) &\approx \frac{n\cdot\left(2\pi\sigma^2\right)^{n/4}\cdot\varGamma(\frac{3n}{4})}{2\varGamma(\frac{n}{2}+1)^{3/2}}\\
&= \frac{n\cdot\varGamma(\frac{3n}{4})}{2\varGamma(\frac{n}{2}+1)^{3/2}\cdot e^{n/4}}\cdot 2^{H(X)/2}\\
&\sim_{n\to\infty} \frac{n\cdot\sqrt{2\pi\frac{3n}{4}}\left(\frac{\frac{3n}{4}-1}{e}\right)^{\frac{3n}{4}-1}}{2\left(\sqrt{2\pi\frac{n}{2}}\cdot\left(\frac{n}{2e}\right)^{n/2}\right)^{\frac{3}{2}}\cdot e^{n/4}}\cdot 2^{H(X)/2}\\
&\sim \frac{n\sqrt{2\pi\frac{3n}{4}}\left(\frac{3n}{4e}\right)^{\frac{3n}{4}-1}}{2e\left(\sqrt{2\pi\frac{n}{2}}\left(\frac{n}{2e}\right)^{n/2}\right)^{\frac{3}{2}}\cdot e^{n/4}} 2^{H(X)/2}\\
&\sim \frac{\sqrt{6}}{3(\pi n)^{\frac{1}{4}}}\left(\frac{27}{8e}\right)^{\frac{n}{4}}\cdot 2^{H(X)/2}
\end{aligned}
$$

We now consider the case of the modular discrete Gaussian. We do not know how to order the elements of $\mathbb{Z}_q^n$ by decreasing probability so we will instead consider one possible order and bound the complexity of this order. This will prove an upper bound on $G(Y)$.

For any $x \in \mathbb{Z}_q$, denote by $\tilde{x} \in x + q\mathbb{Z}$ the unique integer such that $|x| \leqslant \frac{q-1}{2}$. We extend this notion to vectors in $\mathbf{x} \in \mathbb{Z}_q^n$ componentwise. In other words, $\tilde{\mathbf{x}}$ is the lift from $\mathbb{Z}_q$ to $\mathbb{Z}$ centered on 0.

Let $\tau : \mathbb{Z}_q^n \to \mathbb{N}$ be an ordering of $\mathbb{Z}_q^n$ such that for all $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_q^n$, if $\|\tilde{\mathbf{x}}\| < \|\tilde{\mathbf{y}}\|$ then $\tau(\mathbf{x}) < \tau(\mathbf{y})$. In other words, we order points of $\mathbb{Z}_q^n$ according to the norm of their "lift" in $\{\frac{q-1}{2}, \ldots, \frac{q-1}{2}\}$. Intuitively, when σ is much smaller than q, this will be the optimal order but we were not able to show this result. We now have that

$$\begin{aligned}
G(D_{\mathbb{Z}_q^n,\sigma}) &\leqslant \sum_{x\in\mathbb{Z}_q^n} D_{\mathbb{Z}_q^n,\sigma}(x)\cdot\tau(x)\\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\cdot\sum_{\mathbf{x}\in\mathbb{Z}_q^n}\tau(\mathbf{x})\cdot\sum_{\mathbf{y}\in\mathbb{Z}^n}\rho_\sigma(\mathbf{x}+q\cdot\mathbf{y})\\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\sum_{\mathbf{x}\in\mathbb{Z}_q^n}\sum_{\mathbf{y}\in\mathbb{Z}^n}\rho_\sigma(\mathbf{x}+q\cdot\mathbf{y})\tau(\widetilde{\mathbf{x}+q\cdot\mathbf{y}}) \quad \text{since } \widetilde{\mathbf{x}+q\cdot\mathbf{y}}=\tilde{\mathbf{x}}\\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\cdot\sum_{\mathbf{x}\in\mathbb{Z}^n}\rho_\sigma(x)\cdot\tau(\widetilde{\mathbf{x}}).
\end{aligned}$$

Now fix $\mathbf{x}\in\mathbb{Z}^n$. We now observe that by definition of the order τ, $\tau(\tilde{\mathbf{x}})<\tau(\tilde{\mathbf{y}})$ for any $\mathbf{y}\in\mathbb{Z}_q^n$ such that $\|\tilde{\mathbf{y}}\|>\|\tilde{\mathbf{x}}\|$. In particular, choose $\mathbf{y}$ such that $\|\tilde{\mathbf{y}}\|=\|\tilde{\mathbf{x}}\|$ and $\tau(\mathbf{y})$ is the largest possible among all such $\mathbf{y}$. Then

$$\begin{aligned}
\tau(\tilde{\mathbf{y}}) &= |\{\,\mathbf{z}\in\mathbb{Z}_q^n : \|\tilde{\mathbf{z}}\|\leqslant\|\tilde{\mathbf{x}}\|\,\}|\\
&\leqslant |\{\,\mathbf{z}\in\mathbb{Z}^n : \|\mathbf{z}\|\leqslant\|\tilde{\mathbf{x}}\|\,\}|\\
&= N(\|\tilde{\mathbf{x}}\|^2)
\end{aligned}$$

where recall that we defined $N(\ell)=\{\,\mathbf{x}\in\mathbb{Z}^n : \|\mathbf{x}\|\leqslant\sqrt{\ell}\,\}$ and $N(-1)=0$ at the beginning of the proof. Therefore,

$$\begin{aligned}
G(D_{\mathbb{Z}_q^n,\sigma}) &\leqslant \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\sum_{\mathbf{x}\in\mathbb{Z}^n}\rho_\sigma(\mathbf{x})N(\|\tilde{\mathbf{x}}\|^2)\\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\sum_{\ell=0}^{\infty}\sum_{\mathbf{x}\in\mathbb{Z}^n:\|\mathbf{x}\|^2=\ell}\rho_\sigma(\mathbf{x})N(\|\tilde{\mathbf{x}}\|^2)\\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\sum_{\ell=0}^{\infty}\rho_\sigma(\sqrt{\ell})\cdot N(\ell)\sum_{\mathbf{x}\in\mathbb{Z}^n:\|\mathbf{x}\|^2=\ell}1\\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\sum_{\ell=0}^{\infty}\rho_\sigma(\sqrt{\ell})\cdot N(\ell)\cdot(N(\ell)-N(\ell-1))\\
&\leqslant 2G(D_{\mathbb{Z}^n,\sigma}) \qquad \text{by (3).}
\end{aligned}$$

We now consider the case of the quantum guessing complexity. Virtually the same argument yields that

$$\begin{aligned}
G^{qc}(D_{\mathbb{Z}_q^n,\sigma}) &\leqslant \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\sum_{\mathbf{x}\in\mathbb{Z}^n}\rho_\sigma(\mathbf{x})\sqrt{N(\|\tilde{\mathbf{x}}\|^2)}\\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\sum_{\ell=0}^{\infty}\rho_\sigma(\sqrt{\ell})\cdot\sqrt{N(\ell)}\sum_{\mathbf{x}\in\mathbb{Z}^n:\|\mathbf{x}\|^2=\ell}1\\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\sum_{\ell=1}^{\infty}\rho_\sigma(\sqrt{\ell})\cdot\sqrt{N(\ell)}\cdot(N(\ell)-N(\ell-1)).
\end{aligned}$$

Now recall by (5) that

$$G^{qc}(D_{\mathbb{Z}^n,\sigma}) = \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \sum_{i=N(\ell-1)+1}^{N(\ell)} \sqrt{i}.$$

Furthermore, since $\sqrt{\cdot}$ is an increasing function, it is not hard to see that for all $a, b \in \mathbb{N}$,

$$\sum_{i=a+1}^{b} \sqrt{i} \geqslant \int_a^b \sqrt{x}\,\mathrm{d}x = \frac{2}{3}(b^{3/2} - a^{3/2}).$$

Therefore, for any $\ell \in \mathbb{N}$,

$$\sum_{i=N(\ell-1)+1}^{N(\ell)} \sqrt{i} \geqslant \frac{2}{3}\left(N(\ell)^{3/2} - N(\ell-1)^{3/2}\right).$$

But check that $N(\ell-1) \leqslant N(\ell)$ so that

$$\sqrt{N(\ell)}(N(\ell) - N(\ell-1)) \leqslant N(\ell)^{3/2} - N(\ell-1)^{3/2} \leqslant \frac{3}{2} \sum_{i=N(\ell-1)+1}^{N(\ell)} \sqrt{i}.$$

It then follows easily that

$$G^{qc}(D_{\mathbb{Z}_q^n,\sigma}) \leqslant \frac{3}{2}\frac{1}{\rho_\sigma(\mathbb{Z}^n)} \sum_{\ell=1}^{\infty} \rho_\sigma(\sqrt{\ell}) \sum_{i=N(\ell-1)+1}^{N(\ell)} \sqrt{i} = \frac{3}{2} G^{qc}(D_{\mathbb{Z}^n,\sigma}).$$

This finishes the proof. □

B Source code

Our code relies on the modified LWE Estimator from [APS15] available at `https://github.com/malb/lattice-estimator/`. We also attached our code as an attachment to this PDF. Not all PDF viewers support this feature. If the reader's PDF reader does not then e.g. `pdfdetach` can be used to extract the source code without having to copy and paste it by hand.

```
# -*- coding: utf-8 -*-
"""
Run like this::

    sage: attach("estimates.py")
    sage: %time results = runall()
    sage: save(results, "../data/estimates.sobj")
    sage: print(results_table(results))

"""
from sage.all import sqrt, log, exp, e, pi, RR, ZZ

from estimator.estimator.cost import Cost
from estimator.estimator.lwe_parameters import LWEParameters
from estimator.estimator.reduction import delta as deltaf
from estimator.estimator.reduction import RC, ReductionCost
from estimator.estimator.conf import red_cost_model as red_cost_model_default
from estimator.estimator.util import local_minimum, early_abort_range
from estimator.estimator.io import Logging
from estimator.estimator.schemes import (
    Kyber512,
    Kyber768,
    Kyber1024,
    LightSaber,
    Saber,
    FireSaber,
)
from estimator.estimator.schemes import TFHE630, TFHE1024


class ChaLoy21(ReductionCost):

    __name__ = "ChaLoy21"
    short_vectors = ReductionCost._short_vectors_sieve

    def __call__(self, beta, d, B=None):
        """
        :param beta: Block size ≥ 2.
        :param d: Lattice dimension.
        :param B: Bit-size of entries.
        """

        return ZZ(2) ** RR(0.2570 * beta)


class MATZOV:
    """ """

    C_prog = 1.0 / (1 - 2.0 ** (-0.292))  # p.37
    C_mul = 32**2  # p.37
    C_add = 5 * 32  # guessing based on C_mul

    @classmethod
    def T_fftf(cls, k, p):
        """
        The time complexity of the FFT in dimension ‘k‘ with modulus ‘p‘.

        :param k: Dimension
        :param p: Modulus ≥ 2

        """
        return cls.C_mul * k * p ** (k + 1)  # Theorem 7.6, p.38

    @classmethod
    def T_tablef(cls, D):
        """
        Time complexity of updating the table in each iteration.

        :param D: Number of nonzero entries

        """
        return 4 * cls.C_add * D  # Theorem 7.6, p.39

    @classmethod
    def Nf(cls, params, m, beta_bkz, beta_sieve, k_enum, k_fft, p):
        """
        Required number of samples to distinguish with advantage.

        :param params: LWE parameters
        :param m:
        :param beta_bkz: Block size used for BKZ reduction
        :param beta_sieve: Block size used for sampling
        :param k_enum: Guessing dimension
        :param k_fft: FFT dimension
        :param p: FFT modulus

        """
        mu = 0.5
        k_lat = params.n - k_fft - k_enum  # p.15

        # p.39
        lsigma_s = (
            params.Xe.stddev ** (m / (m + k_lat))
            * (params.Xs.stddev * params.q) ** (k_lat / (m + k_lat))
            * sqrt(4 / 3.0)
            * sqrt(beta_sieve / 2 / pi / e)
            * deltaf(beta_bkz) ** (m + k_lat - beta_sieve)
        )

        # p.29, we’re ignoring O()
        N = (
            exp(4 * (lsigma_s * pi / params.q) ** 2)
            * exp(k_fft / 3.0 * (params.Xs.stddev * pi / p) ** 2)
            * (k_enum * cls.Hf(params.Xs) + k_fft * log(p) + log(1 / mu))
        )

        return RR(N)

    @staticmethod
    def Hf(Xs):
        return RR(1 / 2 + log(sqrt(2 * pi) * Xs.stddev)) / log(2.0)

    @classmethod
    def cost(
        cls,
        beta,
        params,
        m=None,
        p=2,
        k_enum=0,
        k_fft=0,
        beta_sieve=None,
        red_cost_model=red_cost_model_default,
    ):
        """
        Theorem 7.6

        """

        if m is None:
            m = params.n

        k_lat = params.n - k_fft - k_enum  # p.15

        # We assume here that β_sieve ≈ β
        N = cls.Nf(
            params,
            m,
            beta,
            beta_sieve if beta_sieve else beta,
            k_enum,
            k_fft,
            p,
        )
        rho, T_sample, _, beta_sieve = red_cost_model.short_vectors(
            beta, N=N, d=k_lat + m, sieve_dim=beta_sieve
        )

        H = cls.Hf(params.Xs)
        T_guess = (
            ((2 / sqrt(e)) ** k_enum)
            * (2 ** (k_enum * H))
            * (cls.T_fftf(k_fft, p) + cls.T_tablef(N))
        )
        cost = Cost(rop=T_sample + T_guess, problem=params)
        cost["red"] = T_sample
        cost["guess"] = T_guess
        cost["beta"] = beta
        cost["p"] = p
        cost["zeta"] = k_enum
        cost["t"] = k_fft
        cost["beta_"] = beta_sieve
        cost["N"] = N
        cost["m"] = m

        cost.register_impermanent(
            {"β'": False, "ζ": False, "t": False}, rop=True, p=False, N=False
        )
        return cost

    def __call__(
        self,
        params: LWEParameters,
        red_cost_model=red_cost_model_default,
        log_level=1,
    ):
        """
        Optimizes cost of dual attack as presented in [Matzov22]_.

        :param params: LWE parameters
        :param red_cost_model: How to cost lattice reduction

        The returned cost dictionary has the following entries:

        - ``rop``: Total number of word operations (≈ CPU cycles).
        - ``red``: Number of word operations in lattice reduction and
                   short vector sampling.
        - ``guess``: Number of word operations in guessing and FFT.
        - ``β``: BKZ block size.
        - ``ζ``: Number of guessed coordinates.
        - ``t``: Number of coordinates in FFT part mod `p`.
        - ``d``: Lattice dimension.

        """
        params = params.normalize()

        for p in early_abort_range(2, params.q):
            for k_enum in early_abort_range(0, params.n, 5):
                for k_fft in early_abort_range(0, params.n - k_enum[0], 5):
                    with local_minimum(
                        40, params.n, log_level=log_level + 4
                    ) as it:
                        for beta in it:
                            cost = self.cost(
                                beta,
                                params,
                                p=p[0],
                                k_enum=k_enum[0],
                                k_fft=k_fft[0],
                                red_cost_model=red_cost_model,
                            )
                            it.update(cost)
                        Logging.log(
                            "dual",
                            log_level + 3,
                            f"t: {k_fft[0]}, {repr(it.y)}",
                        )
                    k_fft[1].update(it.y)
                Logging.log(
                    "dual", log_level + 2, f"ζ: {k_enum[0]}, {repr(k_fft[1].y)}"
                )
                k_enum[1].update(k_fft[1].y)
            Logging.log("dual", log_level + 1, f"p:{p[0]}, {repr(k_enum[1].y)}")
            p[1].update(k_enum[1].y)
        Logging.log("dual", log_level, f"{repr(p[1].y)}")
        return p[1].y


class QMATZOV(MATZOV):
    @classmethod
    def cost(
        cls,
        beta,
        params,
        m=None,
        p=2,
        k_enum=0,
        k_fft=0,
        beta_sieve=None,
        red_cost_model=red_cost_model_default,
    ):
        """
        Theorem 7.6

        """

        if m is None:
            m = params.n

        k_lat = params.n - k_fft - k_enum  # p.15

        # We assume here that β_sieve ≈ β
        N = cls.Nf(
            params,
            m,
            beta,
            beta_sieve if beta_sieve else beta,
            k_enum,
            k_fft,
            p,
        )
        rho, T_sample, _, beta_sieve = red_cost_model.short_vectors(
            beta, N=N, d=k_lat + m, sieve_dim=beta_sieve
        )

        H = cls.Hf(params.Xs)
        T_guess = ((27 / 8 / e) ** (k_enum / 4)) * sqrt(
            2 ** (k_enum * H) * p ** (k_fft / 2.0) * cls.T_tablef(N)
        ) + N
        cost = Cost(rop=T_sample + T_guess, problem=params)
        cost["red"] = T_sample
        cost["guess"] = T_guess
        cost["beta"] = beta
        cost["p"] = p
        cost["zeta"] = k_enum
        cost["t"] = k_fft
        cost["beta_"] = beta_sieve
        cost["N"] = N
        cost["m"] = m

        cost.register_impermanent(
            {"β'": False, "ζ": False, "t": False}, rop=True, p=False, N=False
        )
        return cost
```

```
def runall(
    schemes=(
        Kyber512,
        Kyber768,
        Kyber1024,
        LightSaber,
        Saber,
        FireSaber,
        TFHE630,
        TFHE1024,
    ),
    # schemes=(Kyber512, Kyber768, Kyber1024, LightSaber, Saber, FireSaber),
    nns=(
        "list_decoding-naive_classical",
        "list_decoding-classical",
        "list_decoding-naive_quantum",
        "list_decoding-ge19",
    ),
):

    results = {}

    for scheme in schemes:
        results[scheme] = {}
        print(f"{repr(scheme)}")
        for nn in nns:
            cost = MATZOV()(scheme, red_cost_model=RC.MATZOV.__class__(nn=nn))
            results[scheme][(nn, "classical")] = cost
            print(f" nn: {nn},  cost: {repr(cost)}")
            cost = QMATZOV()(scheme, red_cost_model=RC.MATZOV.__class__(nn=nn))
            print(f" nn: {nn}, qcost: {repr(cost)}")
            results[scheme][(nn, "quantum")] = cost

        cost = MATZOV()(scheme, red_cost_model=RC.ADPS16)
        print(f" C0, cost: {repr(cost)}")
        results[scheme][("C0", "classical")] = cost

        cost = MATZOV()(scheme, red_cost_model=ChaLoy21())
        print(f" Q0, cost: {repr(cost)}")
        results[scheme][("Q0", "classical")] = cost

        cost = QMATZOV()(scheme, red_cost_model=ChaLoy21())
        print(f"Q0, qcost: {repr(cost)}")
        results[scheme][("Q0", "quantum")] = cost

    return results


def results_table(results, fmt=None):
    import tabulate

    rows = []

    def pp(cost):
        return round(log(cost["rop"], 2), 1)

    for scheme, costs in results.items():
        row = [
            scheme.tag,
            pp(costs[("list_decoding-classical", "classical")]),
            pp(costs[("list_decoding-naive_classical", "classical")]),
            pp(costs[("C0", "classical")]),
            pp(costs[("list_decoding-ge19", "classical")]),
            pp(costs[("list_decoding-naive_quantum", "classical")]),
            pp(costs[("Q0", "classical")]),
            pp(costs[("list_decoding-naive_quantum", "quantum")]),
            pp(costs[("Q0", "quantum")]),
        ]
        rows.append(row)
    if fmt is None:
        return rows
    else:
        return tabulate.tabulate(
            results_table(results),
            headers=[
                "Scheme",
                "CC",
                "CN",
                "C0",
                "GE19",
                "QN",
                "Q0",
                "This work (QN)",
                "This work (Q0)",
            ],
            tablefmt="latex_booktabs",
            floatfmt=".1f",
        )
```