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, two out of four selected algorithms rely on the conjectured hardness of algebraic variants of Learning With Errors [SSTX09, LPR10] problem.

From the perspective of a cryptanalyst equipped with a quantum computer, lattice problems such as LWE are frustrating. Known quantum speedups to solve these problems [LMv13, Laa15, CL21] 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 lattice reduction algorithms 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" [BV11, AFFP14, KF15, GJS15] which further provides significant performance gains. Overall, 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.

In Section 3 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 an exponential factor in the dimension.

In Section 4 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.

In Section 5 we provide and analyse a quantum augmented dual attack utilising our guessing and mean estimation results.

In Section 6, we then estimate the impact of our algorithm on the cost of solving instances of lattice-based schemes with parameters taken from the literature. We will refer to such parameters as "lattice parameters" going forward. 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).[1] Interestingly, our optimisation routine produces attack parameters that skip the FFT step ("$k_{\text{fft}} = 0$") in favour of a more costly secret guessing stage ("$k_{\text{enum}} > 0$"). Looking ahead, we speculate that this is because we get a super quadratic speedup on the enumeration part and only a quadratic speedup on $p^{k_{\text{fft}}}$. If $2^{H(\chi_s)}$ is sufficiently smaller than p then it is better to enumerate the entire first $n - k_{\text{lat}}$ coordinates; a case we are in often for practical parameters.

In Section 7, 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.

Discussion. On the one hand, we analyse our algorithm in the same "cost model" as prior work such as [LMv13, Laa15, CL21] and in this cost model we obtain mild but noticeable speed-ups over previous work. These are mostly derived from our new algorithm for mean estimation and its composition with other known quantum algorithms from the literature such' as Grovers search for a secret and quantum versions of lattice-reduction attacks.

On the other hand, the cost model adopted here and prior work assume unit cost for quantum operations including accessing classical RAM in superposition (QRACM). As discussed in e.g. [AGPS20], this is a very strong assumption.

Furthermore, even in this model our algorithm falls far short of the quadratic speed-up we would need to "force" a revision of lattice parameters. This is because the resistance of post-quantum algorithms to quantum computers is routinely, e.g. in the NIST PQC Standardization Process, compared to that of the AES family of block ciphers (or other symmetric-key primitives). Here, the state of the art is that AES-λ resists classical attacks of cost $\approx 2^{\lambda}$ and quantum attacks of cost $\approx 2^{\lambda/2}$, the latter being due to Grover's algorithm, see [JNRV20] for more detailed cost estimates. Thus, parameters for post-quantum schemes

[1] A previous version of this work reported wrong estimates due to a bug in our estimation code, which did not match the theorems given in the written report.

are chosen such that they resist known classical attacks of cost $\approx 2^\lambda$ and known quantum attacks of cost $\approx 2^{\lambda/2}$ and any quantum algorithm with complexity $\gg 2^{\lambda/2}$ will not affect the claimed security level. Our algorithm has cost $\gg 2^{\lambda/2}$, whatever the cost model.

2 Preliminaries

Recall that $e^{ix} = \cos(x) + i\,\sin(x)$. For any $z \in \mathbb{C}$, we write $\Re(z)$ for its real part. We write $[x, y]$ for the interval $\{x, x+1, \ldots, y\} \subset \mathbb{Z}$. 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 $\mathbf{v}$.

For any $x \in \mathbb{Z}_q$, denote by $\tilde{x} \in x + q\mathbb{Z}$ the unique integer such that $|\tilde{x}| \leqslant \frac{q-1}{2}$. 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. We define $\|\mathbf{x}\|$ for $\mathbf{x} \in \mathbb{Z}_q^n$ as $\|\tilde{\mathbf{x}}\|$.

2.1 Lattices

A lattice $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^d$. We can represent it as $\{\sum x_i \cdot \mathbf{b}_i | x_i \in \mathbb{Z}\}$ 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.

Algorithm 1: Short Vectors Sampling Procedure [GJ21]

Input: A basis $\mathbf{B} = \begin{bmatrix}\mathbf{b}_0 & \ldots & \mathbf{b}_{d-1}\end{bmatrix}$ for a lattice and $2 \leq \beta_0, \beta_1 \in \mathbb{Z} \leqslant d$ and D.
Output: A list of D vectors from the lattice.

```
L={}.
for i ∈ [0, ⌈D/N_sieve(β_1)⌉ − 1] do
    Randomise the basis B.
    Run BKZ-β_0 to obtain a reduced basis b'_0, ..., b'_{d−1}.
    Run a sieve in dimension β_1 on the sublattice spanned by b'_0, ..., b'_{β_1−1} to
      obtain a list of N_sieve(β_1) vectors and add them to L.
return L
```

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 $N_{\text{sieve}}(\beta_1)$ short vectors in the lattice $\mathcal{L}(\mathbf{B})$ and has a cost exponential

in β_1. The magnitude $N_{\text{sieve}}(\beta_1)$ also grows exponentially with β_1 but slower than the cost of sieving. We will write $T_{\text{BKZ}}(d, \beta_0)$ for the cost of running BKZ-β_0 in dimension d and $T_{\text{sieve}}(\beta_1)$ for the cost of sieving in dimension β_1. 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 has a cost of $2^{0.257\,\beta_1+o(\beta_1)}$. Thus, according to the best known algorithms we have $T_{\text{BKZ}}(d, \beta_0) \in \mathsf{poly}(d) \cdot 2^{\Theta(\beta_0)}$ and $T_{\text{sieve}}(\beta_1) \in 2^{\Theta(\beta_1)}$.

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 $\text{LWE}_{n,m,\chi_s,\chi_e}$ 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, setting $\mathbf{b} \coloneqq \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \bmod q$ and outputting $(\mathbf{A}, \mathbf{b})$.*

We define two problems:

- Decision-LWE. Distinguish the uniform distribution over $\mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$ from $\text{LWE}_{n,m,\chi_s,\chi_e}$.
- Search-LWE. Given a sample from $\text{LWE}_{n,m,\chi_s,\chi_e}$, 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 w.l.o.g $\mathbf{s}, \mathbf{e}$ are short [ACPS09], the attack finds short $\mathbf{x}_j$ s.t. $\mathbf{x}_j^T \cdot \mathbf{A} \equiv \mathbf{0}$. Then, we either obtain $\mathbf{x}_j^T \cdot \mathbf{A} \cdot \mathbf{s} + \langle \mathbf{x}_j, \mathbf{e}\rangle = \langle \mathbf{x}_j, \mathbf{e}\rangle$ or $\langle \mathbf{x}_j, \mathbf{u}\rangle$. The former follows a distribution with small entries, i.e. the distribution of $|e_j|$ for $e_j \coloneqq \langle \mathbf{x}_j, \mathbf{e}\rangle$ is biased towards elements $< q/2$, and the latter follows a uniform distribution mod q.

In [ADPS16], the "normal form" of the dual attack was introduced which finds short $\mathbf{x}_j$ such that $\mathbf{x}_j^T \cdot \mathbf{A} \equiv \mathbf{y}_j \bmod q$ with $\mathbf{y}_j$ short. We then obtain

$$\mathbf{x}_j^T \cdot \mathbf{A} \cdot \mathbf{s} + \langle \mathbf{x}_j, \mathbf{e}\rangle = \langle \mathbf{y}_j, \mathbf{s}\rangle + \langle \mathbf{x}_j, \mathbf{e}\rangle\,,$$

which follows a distribution with small entries when $\mathbf{y}_j, \mathbf{s}, \mathbf{x}_j$ and $\mathbf{e}$ are short.

In [Alb17] a composition of the dual attack with a guessing stage (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.

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

Thus, guessing the correct $\mathbf{s}_1$ and computing $\langle \mathbf{x}_j, \mathbf{b} \rangle - \mathbf{x}^T \cdot \mathbf{A}_1 \cdot \mathbf{s}_1$ produces a value that follows a distribution with small entries. 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, KF15, GJS15] which provides significant performance gains.[2]

2.3 Discrete Gaussian Distribution

Let $\sigma > 0$. For any $\mathbf{x} \in \mathbb{R}^d$, we let $\rho_\sigma(\mathbf{x}) \coloneqq \exp(-\|\mathbf{x}\|^2/2\sigma^2)$. Note that this is 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. We extend the definition of $\rho_\sigma(\cdot)$ to sets of vectors $\mathcal{S}$ by letting $\rho_\sigma(\mathcal{S}) \coloneqq \sum_{\mathbf{x} \in \mathcal{S}} \rho_\sigma(\mathbf{x})$. For any lattice $\mathcal{L} \subset \mathbb{R}^d$, we denote by $D_{\mathcal{L},\sigma}$ the discrete Gaussian distribution over $\mathcal{L}$, defined by $D_{\mathcal{L},\sigma}(\mathbf{x}) = \rho_\sigma(\mathbf{x})/\rho_\sigma(\mathcal{L})$ for all $\mathbf{x} \in \mathcal{L}$. Observing that $D_{\mathbb{Z}^n,\sigma}(\mathbf{x})$ only depends on $\|\mathbf{x}\|$, we abuse notation and for $\ell = \|\mathbf{x}\|$ write $\rho_\sigma(\ell) = \exp(-\ell^2/2\sigma^2)$ and $D_{\mathbb{Z}^n,\sigma}(\ell) = \rho_\sigma(\ell)/\rho_\sigma(\mathbb{Z}^n)$.

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 we will use often implicitly.

We let $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-t^2/2)\ \mathrm{d}t$ be the cumulative distribution function (cdf) of the standard normal distribution and $\Phi^{-1}(x) : [0,1) \to \mathbb{R}$ its inverse.

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*

[2] 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.

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).

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.

For a family of algorithms parameterised by n, the *time complexity* $T(n)$ is the classical time complexity of $\mathcal{A}$ plus the time complexity of the circuits, i.e. 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, i.e. 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 reversible classical computation can be implemented by a quantum computer, although at a non-negligible cost. Here reversible classical computation means that there is a one-to-one mapping between the inputs and the outputs and that a transform exists to return from the output to the input.

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 *principle of deferred measurement* [NC11]. We will use this fact implicitly in the rest of this work and just assume that we can take any classical algorithm and turn it 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 $\{0, 1, \ldots, N-1\}$, 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, we try to minimise the number of queries to the oracle. In the classical case, we need $O(N)$ queries to solve such a problem. Grover, on the other hand, provided 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 here present a generalisation of Grover's algorithm called amplitude amplification [BHMT02].

Theorem 2 (Amplitude Amplification [BHMT02]). *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 [BHMT02], Theorem 12). *Given a 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. The following result generalises Grover search in this setting. We say that a (probabilistic) Boolean function f has bounded error if there exists $b \in \{0,1\}$ such that $f()$ returns b with probability at least $9/10$.

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 is at least one "1" among the n values; returns $\perp$ if there is 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. *Let N be an integer and $f : [0, N-1] \to \{0,1\}$ a function. Let $\mathcal{O}$ be a (classical or quantum with bounded error) algorithm computing f. Let n_0 be the first index such that $f(n_0) = 1$, or let $n_0 = \perp$ if no such index exists. There exists a quantum algorithm $\mathcal{A}^{\mathcal{O}}$ with the following property. $\mathcal{A}^{\mathcal{O}}(N)$ returns $i \in [0, N-1]$ such that $f(i) = 1$, or $\perp$. With constant probability, $\mathcal{A}^{\mathcal{O}}(N) = n_0$. 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 $\mathcal{O}$. Furthermore, if the algorithm returns $i \in [0, N-1]$, then it only queries $\mathcal{O}$ on values in $[0, \min(N-1, 2i)]$.*

Memory Access. "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 $O(1)$.

Given an input integer $0 \leq i \leq r-1$, 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], there are three 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 $\text{polylog}(r)$ is very strong and corresponds to parallel evaluation of the circuit. The feasibility of constructing efficient QRACM is further discussed in e.g. [AGPS20].

2.5 The Classical Algorithm of [GJ21, MAT22]

In this section, we give an overview of the algorithm in [MAT22], reproduced in Algorithm 2. Our quantum algorithm will be a modified version that relies essentially on the same analysis for the correctness but a new analysis for the quantum complexity. We give an overview of the involved parameters in Table 1.

We are given a sample from $\text{LWE}_{n,m,\chi_s,\chi_e}$, 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 split $\mathbf{A}$ 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:

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

where α is a constant equal to $\frac{\sigma_e}{\sigma_s}$ and is used for normalisation in the case that $\mathbf{s}, \mathbf{e}$ have different distributions. We find D short vectors of the form $\begin{pmatrix} \alpha \cdot \mathbf{x}_j \\ \mathbf{y}_{j,\text{lat}} \end{pmatrix}$ in the column space of $\mathbf{B}$ using some short vectors sampling procedure (see Algorithm 1). Then, given a list L of D such vectors let $\mathbf{y}_{j,\text{fft}} \coloneqq \mathbf{x}_j^T \cdot \mathbf{A}_{\text{fft}}$ and $\mathbf{y}_{j,\text{enum}} \coloneqq \mathbf{x}_j^T \cdot \mathbf{A}_{\text{enum}}$. We can then define the function $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) =$

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

Table 1. Dual attack parameters.

parameters	explanation
n, m, χ_s, χ_e	LWE parameters as in Definition 1
β_0, β_1	BKZ block size β_0 and sieving dimension β_1
p	modulus switching target modulus
μ	the target success probability $0 < \mu < 1$
σ_s^2, σ_e^2	variances of χ_s and χ_e respectively
$\mathbf{s}_{\text{enum}}, \mathbf{s}_{\text{fft}}, \mathbf{s}_{\text{lat}}$	components of $\mathbf{s}$ covered by exhaustive search, FFT and lattice reduction
$\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}$	guesses for $\mathbf{s}_{\text{enum}} \bmod q$ and $\mathbf{s}_{\text{fft}} \bmod p$
$N_{\text{enum}}(\mathbf{s}_{\text{enum}})$	number of guesses with $\tilde{\mathbf{s}}_{\text{enum}}$ with probabilities larger than the probability of $\mathbf{s}_{\text{enum}}$ when drawing $\mathbf{s}$ from χ_s.
$k_{\text{enum}}, k_{\text{fft}}, k_{\text{lat}}$	dimension of $\mathbf{s}_{\text{enum}}, \mathbf{s}_{\text{fft}}$ or $\mathbf{s}_{\text{lat}}$ respectively
$\mathbf{A}_{\text{enum}}, \mathbf{A}_{\text{fft}}, \mathbf{A}_{\text{lat}}$	$\mathbf{A} \cdot \mathbf{s} = \mathbf{A}_{\text{enum}} \cdot \mathbf{s}_{\text{enum}} + \mathbf{A}_{\text{fft}} \cdot \mathbf{s}_{\text{fft}} + \mathbf{A}_{\text{lat}} \cdot \mathbf{s}_{\text{lat}}$
α	scaling/normalisation factor $\alpha \coloneqq \sigma_e/\sigma_s$
L, D	list/number of short vectors returned by sieving oracle $D \coloneqq \lvert L \rvert$
$\psi(\mathbf{s}_{\text{fft}})$	$= \exp(\frac{2\pi i c_{q'}}{p} \sum_t s_t)$ where $q' = q/\gcd(p,q)$, $c_{q'} := 0$ when q' is odd and $c_{q'} \coloneqq 1/2q'$ when q' is even
C	FFT cutoff value for scoring function
Φ	defined in Section 2.3
$\phi_{\text{fp}}(\mu)$, $\phi_{\text{fn}}(\mu)$	$= \Phi^{-1}\left(1 - \frac{\mu}{2 \cdot N_{\text{enum}}(\mathbf{s}_{\text{enum}}) \cdot p^{k_{\text{fft}}}}\right)$, $\Phi^{-1}\left(1 - \frac{\mu}{2}\right)$
D_{eq}	exponential factor coming from the clean Fourier coefficient of the error in the dual attack equations for required samples
D_{round}	exponential factor coming from the rounding when modulus switching for required samples
D_{arg}	$\approx 1/2$, improvement factor coming from considering the complex argument of the Fourier coefficient rather than only the magnitude.
D_{fpfn}	a polynomial factor controlling false positives and negatives

for all $\tilde{\mathbf{s}}_{\text{enum}} \in \mathbb{Z}_q^{k_{\text{enum}}}$ and $\tilde{\mathbf{s}}_{\text{fft}} \in \mathbb{Z}_p^{k_{\text{fft}}}$. First, note that $\tilde{\mathbf{s}}_{\text{fft}}$ is mod $p \ll q$ rather than q meaning that only up to $p^{k_{\text{fft}}}$ candidates have to be considered rather than $q^{k_{\text{fft}}}$. Applying modulus switching here is the key innovation of [MAT22], but since its details do not matter for us here, we refer the reader to [MAT22] for details. Second, here $\psi(\tilde{\mathbf{s}}_{\text{fft}})$ is a complex factor of norm 1 defined in [MAT22, p. 25, proof of Lemma 5.4] and easily computable (cf. Table 1). 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 respect to some cutoff parameter C (computed below):

- 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 these properties hold 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 we 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 $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > C$.

The claimed relationships hold with high probability over the choice of the elements in L, assuming sufficiently many vectors. Here "sufficiently many" depends on the input LWE parameters as well as parameters of the dual attack algorithm. In particular, this magnitude depends on (a) D_{eq} which is an exponential factor coming from the LWE error, (b) D_{round} which is an exponential factor coming from the rounding when modulus switching and (c) D_{fpfn} is a polynomial factor controlling false positives and negatives. The following lemma formally states the required magnitudes.

Lemma 2 (Adapted from Theorem 5.2 in [MAT22]). *Let $(n, m, q, \chi_s, \chi_e)$ be LWE parameters and $(\beta_0, \beta_1, k_{\text{enum}}, k_{\text{fft}}, k_{\text{lat}}, p)$ be parameters for Algorithm 2. Let $0 < \mu < 1$ be the targeted failure probability. Let σ_e be the standard deviation 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

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

$$D_{\mathrm{eq}} = \exp\left(4\left(\frac{\pi\tau}{q}\right)^2\right) \text{ for } \tau^2 = \frac{\alpha^{-2}\cdot\|\mathbf{e}\|^2 + \|\mathbf{s}_{\mathrm{lat}}\|^2}{m+k_{\mathrm{lat}}}\ell^2,$$

$$D_{\mathrm{round}} = \left(\prod_{\substack{t=0\\ s_t\neq 0}}^{k_{\mathrm{fft}}-1}\left(\frac{\sin\left(\frac{\pi s_t}{p}\right)}{\frac{\pi s_t}{p}}\right)\right)^{-2} \text{ for } \mathbf{s}_{\mathrm{fft}} = (s_0,\ldots,s_{k_{\mathrm{fft}}-1}),$$

$$D_{\mathrm{arg}} = \frac{1}{2} + \exp\left(-8\left(\frac{\pi\tau}{q}\right)^2\right) \text{ and } D_{\mathrm{fpfn}}(\mu) = \left(\phi_{\mathrm{fp}}(\mu)+\phi_{\mathrm{fn}}(\mu)\right)^2.$$

Remark 1. In [MAT22] two theorems are given establishing costs: Theorem 5.2 (essentially reproduced above), which is with respect to a fixed tuple $(\mathbf{s},\mathbf{e})$ and Theorem 5.9 which is with respect to distributions χ_s,χ_e and which will be restated in Lemma 6.

Algorithm 2: Dual Attack of [MAT22]

Input: LWE parameters (n,m,q,χ_s,χ_e), integers $\beta_0,\beta_1 \leqslant d$, integers $k_{\mathrm{enum}},k_{\mathrm{fft}},k_{\mathrm{lat}}$ such that $k_{\mathrm{enum}}+k_{\mathrm{fft}}+k_{\mathrm{lat}}=n$, 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: (Guess of) the first k_{enum} coordinates of $\mathbf{s}$ or $\perp$.

1 Decompose $\mathbf{A}$ as $\left[\mathbf{A}_{\mathrm{enum}}\ \mathbf{A}_{\mathrm{fft}}\ \mathbf{A}_{\mathrm{lat}}\right]$ of respective dimensions $m\times k_{\mathrm{enum}}$, $m\times k_{\mathrm{fft}}$ and $m\times k_{\mathrm{lat}}$.

2 Compute the matrix $\mathbf{B}=\begin{bmatrix}\alpha\mathbf{I}_m & \mathbf{0}\\ \mathbf{A}_{\mathrm{lat}}^T & q\mathbf{I}_{k_{\mathrm{lat}}}\end{bmatrix}$ where $\alpha=\frac{\sigma_e}{\sigma_s}$

3 Run Algorithm 1 on the basis $\mathbf{B}$ with parameters β_0,β_1,D to get a list L of D short vectors.

4 **for** every value $\tilde{\mathbf{s}}_{\mathrm{enum}}$ in decreasing order of likelihood according to the secret distribution **do**

5 Initialise a table T of dimensions $\underbrace{p\times p\times\cdots p}_{k_{\mathrm{fft}}\text{ times}}$

6 **for** every short vector $(\alpha\mathbf{x}_j,\mathbf{y}_{\mathrm{lat}})$ in L **do**

7 Compute $\mathbf{y}_{j,\mathrm{fft}}=\mathbf{x}_j^T\cdot\mathbf{A}_{\mathrm{fft}}$.

8 Compute $\mathbf{y}_{j,\mathrm{enum}}=\mathbf{x}_j^T\cdot\mathbf{A}_{\mathrm{enum}}$.

9 Add $\exp\left((\mathbf{x}_j^T\cdot\mathbf{b}-\mathbf{y}_{j,\mathrm{enum}}^T\cdot\tilde{\mathbf{s}}_{\mathrm{enum}})\cdot\frac{2i\pi}{q}\right)$ to cell $\lfloor\frac{p}{q}\mathbf{y}_{j,\mathrm{fft}}\rceil$ of T.

10 Perform FFT on T

11 **if** for any $\tilde{\mathbf{s}}_{\mathrm{fft}}$, the real part of $\frac{1}{\psi(\tilde{\mathbf{s}}_{\mathrm{fft}})}T[\tilde{\mathbf{s}}_{\mathrm{fft}}]$ is larger than C **then**

12 **return** $\tilde{\mathbf{s}}_{\mathrm{enum}}$.

13 **return** $\perp$

Remark 2. In Line 4 the algorithm asks to enumerate candidates for $\tilde{\mathbf{s}}_{\mathrm{enum}}$ in decreasing order of likelihood. A technique for achieving this enumeration is discussed in [MAT22, Section 5.4]. Roughly, this is accomplished by enumerating with increasing bounds on log-likelihoods for candidates, see [MAT22] for details.

3 Quantum Guessing

In this section, we give a more rigorous analysis of the classical and quantum guessing complexity for targets following a discrete Gaussian distribution. In more detail, 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 can be shown that the best strategy is to try values of X in 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 denotes Shannon's entropy (i.e. in base 2). On the other hand, the same work shows that it is not, in general, possible to bound $G(X)$ in terms of $H(X)$ only. In Lemma 4, 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 work, we are interested in the *quantum complexity* of guessing. Montanaro showed [Mon11] that the expected number of guesses in this case becomes,

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

and that this is the best possible. However, [Mon11] only deals with the case where the oracle (which answers the question "is X equal to x?") always returns the correct answer. Using Lemma 1, a variant of Grover search that can handle two-sided errors, we extend this algorithm to deal with bounded error oracles.

Lemma 3. *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 non-increasing probability according to X. Let $(\mathcal{O}_x)_{x \in E}$ be a collection of oracles such that for any $x \in E$, $\mathcal{O}_x(x)$ returns 1 with probability at least $9/10$ and for all $y \neq x$, $\mathcal{O}_x(y)$ returns 0 with probability at least $9/10$. 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. See Appendix A.

We now study the guessing complexity of a n-dimensional discrete Gaussian, its modular version and also relate these 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 4. *Let $n \geqslant 4$. Then*

$$1 \leqslant \frac{\rho_\sigma(\mathbb{Z}^n)}{(\sigma\sqrt{2\pi})^n} \leqslant \coth(\pi^2\sigma^2), \qquad G(D_{\mathbb{Z}^n,\sigma}) \leqslant \frac{2^n\rho_\sigma(\mathbb{Z}^n)}{1-e^{-1/2\sigma^2}},$$

and

$$-n\frac{8\pi^2\sigma^2e^{-2\pi^2\sigma^2}}{\log(2)\coth(\pi^2\sigma^2)} \leqslant H(D_{\mathbb{Z}^n,\sigma}) - \frac{\frac{1}{2}+\log(\sigma\sqrt{2\pi})}{\log(2)}n \leqslant n\log_2(\coth(\pi^2\sigma^2)).$$

Furthermore, if $\sigma \geqslant \sqrt[4]{\frac{2}{27\pi^2}} \approx 0.294$ then

$$G^{qc}(D_{\mathbb{Z}^n,\sigma}) \leqslant \frac{7}{6}\cdot\left(\frac{3}{2}\right)^{3n/4}\frac{\sqrt{\rho_\sigma(\mathbb{Z}^n)}}{(1-e^{-1/3\sigma^2})^{3/2}}.$$

Furthermore,

$$G(D_{\mathbb{Z}^n,\sigma}) \leqslant \frac{1}{1-e^{-1/2\sigma^2}}\left(\frac{2a(\sigma)}{\sqrt{e}}\right)^n\cdot 2^{H(D_{\mathbb{Z}^n,\sigma})},$$

$$G^{qc}(D_{\mathbb{Z}^n,\sigma}) \leqslant \frac{7}{6}\cdot\frac{1}{(1-e^{-1/3\sigma^2})^{3/2}}\left(\frac{27a(\sigma)^2}{8e}\right)^{n/4}\cdot 2^{H(D_{\mathbb{Z}^n,\sigma})/2}$$

where $a(\sigma) = e^{8\pi^2\sigma^2e^{-2\pi^2\sigma^2}\tanh(\pi^2\sigma^2)}$ is plotted on Figure 1 and

$$a(\sigma) = 1 + 8\pi^2\sigma^2e^{-2\pi^2\sigma^2} + o(\sigma^2e^{-\pi^2\sigma^2}), \qquad \sigma\to\infty.$$

Finally, for all n and σ,

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

Proof. See Appendix B.

In Lemma 4, we have identified upper bounds which we believe to be tight up to polynomial factors in the dimension. It is not clear that the same method can be used to obtain lower bounds. The bounds we obtained suggest (but do not conclusively show) that the quantum guessing complexity is exponentially smaller than the square root of the classical guessing complexity, for the discrete Gaussian over $\mathbb{Z}^n$. Obtaining lower bounds on these quantities would confirm that this is indeed the case. The situation is the same for the *modular* discrete Gaussian since we also only obtained upper bounds for this distribution.

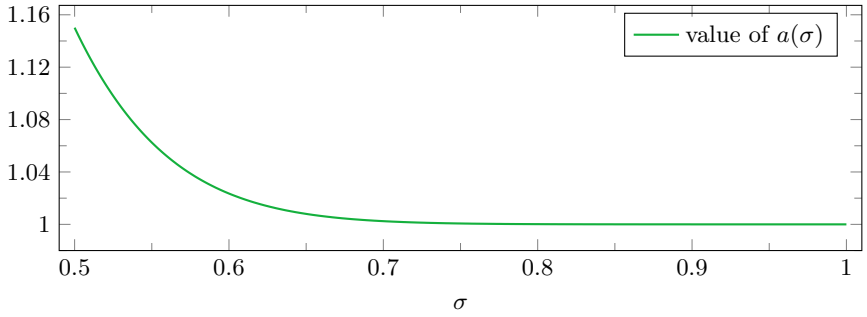

Fig. 1. Value of the extra factor $a(\alpha)$ in Lemma 4.

4 A Quantum Algorithm for Mean Estimation

We provide 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].

Definition 2 (QRACM Oracle). *Let N be a positive integer and W be a list of N vectors $\mathbf{w}_0, \ldots, \mathbf{w}_{N-1} \in \mathbb{Z}^n$. The QRACM oracle for W is*

$$\mathcal{O}_W : |j\rangle |0\rangle \mapsto |j\rangle |\mathbf{w}_j\rangle .$$

Definition 3 (Positive Controlled Rotation Oracle). *The positive controlled rotation oracle for any $a \in \mathbb{R}$ is*

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

which can be implemented up to negligible error by $\mathsf{poly}(\log n)$ quantum elementary gates.

Definition 4 (Cosine Inner Product Oracle). *The cosine inner product oracle for any $\mathbf{b}, \mathbf{w} \in \mathbb{Z}^n$ is*

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

which can be implemented by $\mathsf{poly}(\log n)$ quantum elementary gates.

Theorem 5. *Let N be a positive integer and W be a list of N vectors $\in \mathbb{Z}^n$: $\mathbf{w}_0, \ldots, \mathbf{w}_{N-1}$. Let $f_W(\mathbf{b}) = \frac{1}{N}\sum_{i=0}^{N-1} \cos(2\pi(\langle \mathbf{w}_i, \mathbf{b}\rangle)/q)$, where $\mathbf{b} \in \mathbb{Z}_q^n$. 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 $\mathcal{O}_W$ as defined in Definition 2, outputs $\mathcal{A}^{\mathcal{O}_W}(\mathbf{b})$ which satisfies $|\mathcal{A}^{\mathcal{O}_W}(\mathbf{b}) - f_W(\mathbf{b})| \leq \varepsilon$ with probability $1-\delta$. The algorithm makes $\mathcal{O}(\varepsilon^{-1} \cdot \log \frac{1}{\delta})$ queries to $\mathcal{O}_W$, and requires $\varepsilon^{-1} \cdot \log \frac{1}{\delta} \cdot \mathsf{poly}(\log n)$ elementary quantum gates.*

Proof. Prepare the state $\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}|j\rangle|\mathbf{0}\rangle|\mathbf{b}\rangle|0\rangle|0\rangle$, and then 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 $\gamma_j \coloneqq \cos(2\pi\langle\mathbf{w}_j,\mathbf{b}\rangle/q)$ 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

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

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 taking the median among them, we finish the proof. □

5 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 an (at least) quadratic speedup on the search for $\mathbf{s}_{\text{enum}}$. The original algorithm enumerates them one by one until the correct one is found. By carefully choosing the order in which elements are enumerated, one can show that the expected complexity of this search is related to the guessing complexity (see Section 3). In the quantum setting, we can apply a variant of Grover's search algorithm to obtain a speed-up on this search. The complexity of this search is now related to the quantum guessing complexity which is not necessarily related to the (classical) guessing complexity. Indeed, this quantity

is always smaller than the square root of the classical one and our calculations suggest that it is strictly smaller than the square root for the discrete Gaussian. 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.

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 7). 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]

5.1 Correspondance between the classical and quantum algorithms

Before delving into the analysis of Algorithm 3, we start by explaining the major steps of the algorithm. This is best done by giving the correspondence between our algorithm, and the original algorithm from [MAT22] (Algorithm 2).

- Steps 1-3 are exactly the same for the two algorithms.
- Steps 4-6 create and initialise the QRACM that contains all the short dual vectors. This step is not needed in the original algorithm since the vectors are simply stored in an array.
- Step 7 instantiates the quantum estimation algorithm with certain parameters. This step is only here for readability.
- Steps 8-24 roughly correspond to Steps 5-12 of Algorithm 2. Specifically, our algorithm will perform a quantum search using those steps as an oracle,

[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: (Guess of) the first k_{enum} coordinates of $\mathbf{s}$ or $\perp$.

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 $m \times k_{\text{lat}}$.

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 $\alpha = \frac{\sigma_e}{\sigma_s}$

Run Algorithm 1 on the basis $\mathbf{B}$ with parameters β_0, β_1, D to get a list L of D short vectors.

Create a QRACM $\mathcal{O}_W$

for every short vector $(\alpha \cdot \mathbf{x}_j, \mathbf{y}_{j,\text{lat}})$ in L **do**

  Add vector $\mathbf{x}_j$ to $\mathcal{O}_W$ at index j

Use Theorem 5 to create an algorithm $\mathcal{A}$ with $\delta = \frac{1}{10}$, $\varepsilon = \frac{C}{D}\eta$ and "q"=p

create oracle $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}})$**:**

  create oracle $\hat{\mathcal{O}}(\tilde{\mathbf{s}}_{\text{fft}})$**:**

    Compute θ such that $\psi(\tilde{\mathbf{s}}_{\text{fft}}) = e^{-\frac{2i\pi}{p}\theta}$ (recall that $|\psi(\tilde{\mathbf{s}}_{\text{fft}})| = 1$)

    create oracle $\mathcal{O}'_W(j)$**:**

      Get $\mathbf{x}_j$ from $\mathcal{O}_W$ at index j

      Compute $\mathbf{y}_{j,\text{fft}} = \mathbf{x}_j^T \cdot \mathbf{A}_{\text{fft}}$

      Compute $\mathbf{y}_{j,\text{enum}} = \mathbf{x}_j^T \cdot \mathbf{A}_{\text{enum}}$

      return vector $\left(\frac{p}{q} \cdot \mathbf{y}_{j,\text{enum}}, \left\lfloor \frac{p}{q} \cdot \mathbf{y}_{j,\text{fft}} \right\rceil, \theta - \frac{p}{q} \cdot \mathbf{x}_j^T \cdot \mathbf{b}\right)$

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

      return 1

    else

      return 0

  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

  if $i \neq \perp$ **then**

    return 1

  else

    return 0

create oracle $\tilde{\mathcal{O}}(i)$**:**

  Compute the i^{th} most probable $\tilde{\mathbf{s}}_{\text{enum}}$ according to the distribution $\chi_s^{k_{\text{enum}}}$

  return $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}})$

Find, with probability $\frac{9}{10}$, $\tilde{\mathbf{s}}_{\text{enum}}$ such that $\mathcal{O}(\tilde{\mathbf{s}}_{\text{enum}}) = 1$ using Lemma 3 with oracle $\tilde{\mathcal{O}}$, or let $\tilde{\mathbf{s}}_{\text{enum}} = \perp$ if none is found

return $\tilde{\mathbf{s}}_{\text{enum}}$.

whereas Algorithm 2 uses those steps as a body of a search loop. The result is the same in the sense that our oracle $\mathcal{O}$ essentially produces the same result as the body of the loop in Algorithm 2. The details on how this result is obtained are different however:

- Step 10 is used to compute the phase of the normalisation factor in the FFT sum. This factor is needed at Step 11 of Algorithm 2 but we need to put it in the search oracle because of the way the search algorithm works.
- Steps 11-15 roughly correspond to Steps-6-10. As explained at the beginning of Section 5 and also in Section 7, our quantum algorithm does not perform the FFT in the same way as the original algorithm. Contrary to the classical algorithm where all the output coefficients are computed "at once" by a single FFT computation, our algorithm computes each individual output FFT coefficient on demand and check whether it is above the threshold (this is what $\hat{\mathcal{O}}$ does). To do so, given a specified output coefficient $\tilde{s}_{\text{fft}}$, we define an oracle $\mathcal{O}'_W$ which (essentially) returns the (input) coefficients of the FFT and use Theorem 3 to estimate the output coefficient. We note that the steps 12-14 could have been in the oracle $\mathcal{O}_W$ instead, this would be completely equivalent (and save a small polynomial factor in the overall complexity). Step 16 corresponds to Step 11 of Algorithm 2. The difference between the thresholds comes from the way Theorem 3 works (C vs C/D) and also because of the two-sided errors made by the oracle (factor η). See Theorem 6 for more details.

– Steps 25-28 correspond to Step 5 of Algorithm 2. Specifically, the classical algorithm performs a sequential search by decreasing probability of $\tilde{s}_{\text{enum}}$. Our algorithm uses Lemma 3 to obtain a quantum speedup on this search which requires to create an oracle.

5.2 Analysis of the Quantum Augmented Dual Attack

We now analyse the quantum augmented dual attack given in Algorithm 3. Informally, we first establish that with constant probability the algorithm outputs a guess for $\mathbf{s}_{\text{enum}}$ where the corresponding FFT scoring function's score is above the chosen threshold C. Below, we will instantiate this theorem with appropriate choices of thresholds C (and number of samples D). Note that like [MAT22, Thm. 5.2], the following theorem is with respect to a fixed tuple $(\mathbf{s}, \mathbf{e})$.

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

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

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

Next, we establish that the parameter choices of C, D in Lemma 2 for Algorithm 2 allow us to also instantiate Algorithm 3 such that it succeeds with a comparable probability.

Lemma 5. *Let $(n, m, q, \chi_s, \chi_e)$ be* LWE *parameters, let $(\beta_0, \beta_1, k_{\text{enum}}, k_{\text{fft}}, k_{\text{lat}}, p, C, D, \eta)$ be a tuple of parameters for Algorithm 3, and let $0 < \nu < 1$. Fix $(\mathbf{s}, \mathbf{e}) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^m$. By choosing the parameters C, D according to Lemma 2 with $\mu = \nu/2$, and $\eta \leqslant \frac{\sqrt{2\pi}\mu}{8\phi_{fp}(\mu)}$, Algorithm 3 returns $\mathbf{s}_{\text{enum}}$ with probability at least $1 - \nu$.*

Proof. See Appendix C.

Next, we state the analogous result as [MAT22, Thm. 5.9] for our setting, i.e. express the success probability with respect to χ_s, χ_e rather than a fixed tuple $(\mathbf{s}, \mathbf{e})$. The only difference compared to [MAT22, Thm. 5.9] is that we replace the Shannon entropy $H(X)$ by the guessing complexity $G(X)$, as discussed in Section 3.

Lemma 6. *Let $(n, m, q, \chi_s, \chi_e)$ be* LWE *parameters, $(\beta_0, \beta_1, k_{\text{enum}}, k_{\text{fft}}, k_{\text{lat}}, p, D, C, \eta)$ be parameters for Algorithm 3. Let $0 < \nu < 1$, $\mu = \nu/2$, $\eta \leqslant \frac{\sqrt{2\pi}\mu}{8\phi_{fp}(\mu)}$. Denote by ℓ the expected Euclidean length of the vectors returned by Algorithm 1. Let $G(X)$ as in Lemma 4. Let $D \geq D_{\text{eq}} \cdot D_{round} \cdot D_{\text{arg}} \cdot D_{\text{fpfn}}(\mu)$ and $C = \tilde{\phi}_{\text{fp}}(\mu)\sqrt{D_{\text{arg}} \cdot D}$ with*

$$D_{\text{eq}} = \exp\left(4\left(\frac{\pi\sigma_s\ell}{q}\right)^2\right), \qquad D_{\text{round}} = \prod_{\bar{s}\neq 0}\left(\frac{\sin\left(\frac{\pi\bar{s}}{p}\right)}{\frac{\pi\bar{s}}{p}}\right)^{-2k_{\text{fft}}\chi_s(\bar{s})}$$

$$D_{\text{arg}} = \frac{1}{2}\exp\left(2\left(\chi_e(0) + e^{-\frac{8\pi^2\alpha^{-2}\ell^2}{q^{2\,(m+k_{\text{lat}})}}}\right)^m \cdot \left(\chi_s(0) + e^{-\frac{8\pi^2\ell^2}{q^{2(m+k_{\text{lat}})}}}\right)^{k_{\text{lat}}}\right) \approx \frac{1}{2}$$

$$D_{\text{fpfn}}(\mu) = \left(\phi_{\text{fp}}(\mu) + \phi_{\text{fn}}(\mu)\right)^2 \cdot \mu$$

$$\phi_{\text{fp}}(\mu) = \Phi^{-1}\left(1 - \frac{\mu}{2 \cdot G\left(\chi_s\right) \cdot p^{k_{\text{fft}}}}\right) \quad \text{and} \quad \tilde{\phi}_{\text{fn}}(\mu) = \Phi^{-1}\left(1 - \frac{\mu}{2}\right).$$

Then, with probability at least $1 - \nu$ the algorithm returns $\mathbf{s}_{\text{enum}}$.

The proof is exactly the same as in [MAT22, Theorem 5.9] 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 3 for more details. We can now state our main theorem which gives the complexity of our quantum augmented dual attack.

Theorem 7. *Let* $(n, m, q, \chi_s, \chi_s)$ *be* LWE *parameters,* $(\beta_0, \beta_1, k_{\text{enum}}, k_{\text{fft}}, k_{\text{lat}}, p)$ *be a partial tuple of parameters for Algorithm 3, and let* $0 < \nu < 1$*. Let* $T_{\text{BKZ}}(d, \beta_0)$ *denote the cost of running BKZ-*β_0 *in dimension* d*, let* $T_{\text{sieve}}(\beta_1)$ *be the cost of sieving in dimension* β_1 *and let* $G^{qc}(X)$ *be the quantum guessing complexity. Choosing the parameters* C, D *according to Lemma 6, Algorithm 3 outputs* $\mathbf{s}_{\text{enum}}$ *with probability at least* $1 - \nu$ *in expected time*

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

up to polynomial factors.

Proof. The correctness follows directly from Lemma 6. We now analyse the complexity. In this proof, we will neglect all polynomials factors in n and m.

First, the cost of Step 3 is that of Algorithm 1, which is

$$\left\lceil \frac{D}{N_{\text{sieve}}(\beta_1)} \right\rceil \cdot (T_{\text{BKZ}}(d, \beta_0) + T_{\text{sieve}}(\beta_1)) \tag{1}$$

and $N_{\text{sieve}}(\beta_1) = (\sqrt{4/3})^{\beta_1 + o(\beta_1)}$ (this was justified in [GJ21]). Next, the cost of creating (Step 4) and filling (Steps 5-the QRACM is D (up to polynomial factors), by the assumption of the QRACM model (see Section 2.4), which is negligible compared to (1). The cost of Step 7 is zero (all the cost of the algorithm is when $\mathcal{A}$ is run), this step is just for the readability of the algorithm.

We now analyse the complexity of each call to the oracle $\mathcal{O}$. On input $\tilde{s}_{\text{enum}}$, the algorithm starts by creating the oracle $\hat{\mathcal{O}}$. This creation cost is negligible, all the cost is incurred when the oracle is run. We need to analyse the cost of each call to $\hat{\mathcal{O}}$. The cost of computing θ (Step 10) is negligible (polynomial). The creation of oracle $\mathcal{O}'_W$ is also negligible and each call to $\mathcal{O}'_W$ takes time polynomial since in our model, getting an entry from the QRACM $\mathcal{O}_W$ is $O(1)$. Therefore, the cost of each call to $\hat{\mathcal{O}}$ is, up to polynomial factors, the cost of running $\mathcal{A}^{\mathcal{O}'_W}$. By Theorem 5, this algorithm makes $O(\varepsilon^{-1} \log \frac{1}{\delta})$ queries to $\mathcal{O}'_W$ (which runs in polynomial time) and takes time $O(\varepsilon^{-1} \log(\frac{1}{\delta}) \mathsf{poly}(\log n))$. Recall that by Step 7, we have $\delta = \frac{1}{10}$ and $\varepsilon = \frac{C}{D} n$. Furthermore, recall that we chose the parameter C according to Lemma 2, that is

$$C = \phi_{\text{fp}}(\mu) \cdot \sqrt{D_{\text{arg}} \cdot D}, \qquad D_{\text{arg}} = \frac{1}{2} + \exp\left(-8\left(\frac{\pi\tau}{q}\right)^2\right)$$

It is immediate that $D_{\text{arg}} \geqslant 1/2$ by definition, and $\phi_{\text{fp}}(\mu)$ is a constant factor (it only depends $\mu = \frac{\nu}{2}$ which is fixed). Therefore, each call to $\hat{\mathcal{O}}$ takes time

$$O\left(\frac{D}{C} \mathsf{poly}(\log n)\right) = O\left(\sqrt{D} \mathsf{poly}(\log n)\right). \tag{2}$$

We can now finish the analysis of the cost of running $\mathcal{O}$. The cost of the search for an FFT coefficient above the threshold (Step 20) is given by Theorem 4: it

makes $O(\sqrt{M})$ calls to $\hat{\mathcal{O}}$ where M is the size of the search space, which is $\mathbb{Z}_p^{k_{\text{fft}}}$. Hence, the total cost of each call to $\mathcal{O}$ is

$$O(\sqrt{p^{k_{\text{fft}}}}) = O\left(p^{k_{\text{fft}}/2}\sqrt{D}\mathsf{poly}(\log n)\right) \tag{3}$$

It remains to analyse the final cost: creating the oracle $\tilde{\mathcal{O}}$ takes polynomial time. The cost of each call to the oracle is (3) plus a polynomial cost. Indeed, finding the i^{th} most probably vector according to $\chi_s^{k_{\text{enum}}}$ can easily be done in polynomial time for all reasonable choices of χ_s such as the discrete Gaussian. Finally, the final search (Step 28), given by Lemma 3, takes an expected time $G^{qc}(\chi_s^{k_{\text{enum}}})$ and makes an expected $G^{qc}(\chi_s^{k_{\text{enum}}})$ calls to $\mathcal{O}$. Therefore the overall cost of this step is

$$O(G^{qc}(\chi_s^{k_{\text{enum}}}) \cdot (3)) = O\left(\sqrt{D}p^{k_{\text{fft}}/2}G^{qc}(\chi_s^{k_{\text{enum}}})\mathsf{poly}(\log n)\right). \tag{4}$$

To conclude the proof, just note that the total cost of the algorith, up to polynomial factors is (1) + (4). □

6 Application

In this section we give *rough* estimates for the impact of our algorithm on the cost of solving lattice parameters from the literature. In particular, we consider NIST PQC Round 3 candidate Saber [DKR+20] and NIST PQC to-be-standardised candidate Kyber [SAB+20] and some TFHE parameters [CGGI20].[4] 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 elementary operations such as linear algebra mod q 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 quantum 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. Similarly, in the

[4] These schemes do not use discrete Gaussians as secret distributions. However, we simply ignore this difference here.

same spirit as the "Core-SVP" model, we cost algorithms by simply dropping the $O()$ around cost estimates, which means our estimates may over- or underestimate the true cost; however, all evidence so far points towards underestimating the costs.
– 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 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.

Table 2. Dual attack cost estimates. All costs are logarithms to base two.

Scheme	CC	CN	C0	GE19	QN	Q0	This work (QN)	This work (Q0)
Kyber 512	139.2	134.4	115.4	139.5	124.4	102.7	119.3	99.7
Kyber 768	196.1	190.6	173.7	191.9	175.3	154.6	168.3	150.0
Kyber 1024	262.4	256.1	241.8	252.0	234.5	215.0	225.6	208.4
LightSaber	145.3	140.8	121.8	145.3	129.7	107.7	120.8	100.6
Saber	204.7	198.9	182.3	199.0	182.6	162.4	176.6	157.9
FireSaber	267.9	261.7	247.4	257.0	239.4	220.3	231.3	214.0
TFHE630	118.2	113.3	93.0	120.2	105.2	83.0	100.8	80.7
TFHE1024	122.0	117.2	95.4	123.9	108.5	84.8	105.6	83.2

We give the source code and the results of the comparison in Appendix E and Table 2.[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], those tagged "asymptotic" cf. [MAB+22].[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

[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 4.

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 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.[6]

QN Quantum costs in the quantum query model for Algorithm 2 using the quantum version of [BDGL16] as the sieving oracle; all other steps are classical. 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; all other steps are classical. 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".

Remark 3. It is worth noting that our optimisation routine almost always returns attack parameters where $k_{\text{fft}} = 0$ and $k_{\text{enum}} > 0$, i.e. the FFT part of the algorithm is disabled, leaving the mean estimation and secret guessing part.

On the one hand, comparing the column labelled "QN" and the last column shows that our algorithm offers an improvement of between 3 and 9 "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".

7 Open Problems

The crux of our quantum improvement is Section 5. Here we formalise the problem that this algorithm solves a promise variant. We introduce some minor no-

[6] Note that the quantum costs in this cost model may be higher than classical costs because the crossover between classical and quantum computing under these cost models is in higher dimensions.

tation first. Fix 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 $\mathbf{u}_j$ are elements of G and w_j complex numbers. For every $\mathbf{x} \in G$, we let

$$f_L(\mathbf{x}) = \sum_{j=0}^{k-1} w_j e^{-\frac{2i\pi}{q}\mathbf{u}_j^T \mathbf{x}}.$$

In other words, f_L is the (unormalized) discrete Fourier transform of the sequence $T : G \to \mathbb{C}$ defined by

$$T(\mathbf{u}) = \sum_{j:\mathbf{u}_j = \mathbf{u}} w_j. \tag{5}$$

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:** $L = \{(\mathbf{u}_0, w_0), \ldots, (\mathbf{u}_{k-1}, w_{k-1})\}$,
- **output:** decide whether $\exists \mathbf{x} \in G$ such that $\Re(f_L(\mathbf{x})) > \delta$.

PROMISE-INPUT-SPARSE-FFT-THRESHOLD:

- **input:** $G = \mathbb{Z}_q^n$ a finite group,
- **input:** $\delta^+ > \delta^- > 0$ two thresholds,
- **input:** $L = \{(\mathbf{u}_0, w_0), \ldots, (\mathbf{u}_{k-1}, w_{k-1})\}$,
- **promise:** $\Re(f_L(\mathbf{x})) \notin [\delta^-, \delta^+]$ for all $\mathbf{x} \in G$,
- **output:** decide whether $\exists \mathbf{u} \in G$ such that $\Re(f_L(\mathbf{x})) > \delta^+$.

Remark 4. To map this formulation back to our task, let $\mathbf{u}_j = \lfloor \frac{p}{q} \cdot \mathbf{y}_{j,\text{fft}} \rceil$ and $w_j = \exp\left((\mathbf{x}_j^T \cdot \mathbf{b} - \mathbf{y}_{j,\text{enum}}^T \cdot \tilde{\mathbf{s}}_{\text{enum}}) \cdot \frac{2i\pi}{q}\right)$. Line 9 of Algorithm 2 is then adding w_j to the cell $\mathbf{u}_j$ of T, hence building T as defined in (5). We then seek to decide if there is some $\mathbf{x} = \tilde{\mathbf{s}}_{\text{fft}} \in G = \mathbb{Z}_p^{k_{\text{fft}}}$ s.t. $\Re(f_L(\mathbf{x})) > \delta = C$, i.e. the entry in the FFT'd table T.

Classical case

In the classical case, to the best of our knowledge, the best algorithm is to first fill an multidimensional array T with the k coefficients and perform a complete FFT on the $|G|$ coefficients, which therefore takes time $O(|G| \log |G|)$ plus $O(k)$ to fill the array. 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 small compared to the number of input coefficients, 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 for all ranges of parameters. This is surprising in light of the fact that QFT can be done in polynomial time: we will explain below why this fact alone is not sufficient.

With access to a QRACM

Our quantum (with QRACM) algorithm from Section 5 solves PROMISE-INPUT-SPARSE-FFT-THRESHOLD as follows. For every $\mathbf{x} \in G$, it computes an approximation of $\Re(f_L(\mathbf{x}))$ with error at most $\frac{1}{2}(\delta^+ - \delta^-)$ and then compares 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 $\mathbf{x} \in G$ can be done using Grover's algorithm, and the approximation is done by amplitude estimation (Theorem 5). The running time of our algorithm is $k\sqrt{|G|}/(\delta^+ - \delta^-)$ assuming oracle access to the $\mathbf{u}_j$ and w_j, and it outputs a correct index with constant probability. A QRACM holding the $\mathbf{u}_j$ and w_j can be built in time $O(k)$. In the dual attack algorithm, the interesting set of parameters for this algorithm is roughly

$$\delta^+ - \delta^- = \Theta(\sqrt{k}), \qquad \delta^- = \Theta(\sqrt{k}), \qquad |w_j| = 1. \tag{6}$$

Therefore our algorithm has running time roughly $O(\sqrt{k|G|})$ (assuming oracle access to the $\mathbf{u}_j$ and w_j) which is always better than $O(|G| \log |G| + k)$ and potentially much better if k is either much smaller or much bigger than $|G|$.

Without access to a QRACM

It is possible to obtain a speed up over the classical algorithm when k is much smaller than $|G|$ without using any QRACM. First, one can create an oracle $\mathcal{O}_L$ that given $\mathbf{x}$, compute $f_L(\mathbf{x})$ and returns 1 if it is greater than δ. This oracle contains the list L hardcoded into the circuit and compute the sum naively. Hence, it has depth $O(k)$ and takes time $O(k)$ to evaluate. We then use Grover's algorithm to search for a $\mathbf{u}$ such that $\mathcal{O}(\mathbf{x}) = 1$, that is $f_L(\mathbf{u}) > \delta$. Grover's search and the oracle can be implemented with a plain quantum circuit and a polynomial number of qubits, without the need for a QRACM. The overall complexity is then, up to some small polynomial factors for the arithmetic operations that we have neglected,

$$O(k\sqrt{|G|}).$$

This algorithm is better than the classical $O(|G| \log |G|)$ whenever $k = o(\sqrt{|G|})$. In the context of dual attacks, this unfortunately brings no improvement for our algorithm (without access to a QRACM) since the optimal choice of the parameters always seem to satisfy $k \gtrsim \sqrt{|G|}$. For this reason, we did not report the results in Table 2.

Why the QFT does not help

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 need to create the superposition

$$|\psi\rangle = \frac{1}{\sqrt{Z}} \sum_{j=0}^{k-1} w_j \, |\mathbf{u}_j\rangle \tag{7}$$

where Z is the normalisation factor. In general,

$$Z = \sum_{\mathbf{u} \in G} \left| \sum_{j : \mathbf{u}_j = \mathbf{u}} w_j \right|^2$$

which simplifies to

$$Z = \sum_{j=0}^{k-1} |w_j|^2 \qquad \text{if the } \mathbf{u}_j \text{ are distinct.} \tag{8}$$

We then perform a QFT on $|\psi\rangle$ to obtain

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

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

1. it is not clear how to create the superposition in Equation (28) efficiently without a QRACM,
2. it is not clear that we can efficiently detect whether there is an amplitude in front of some $|\mathbf{x}\rangle$ which is above the threshold.

The first point is not a problem in our setting since we consider QRACM as part of our model of computation. Indeed, by storing the w_j in a QRACM and applying some elementary operations, one can produce such a state with high probability (see e.g. [MKF19, Theorem 1]). Regarding the second point, a possible strategy would be to measure $|\widehat{\psi}\rangle$ and obtain one $\mathbf{x}$. By repeating this algorithm a very large number of times, we can approximate the probability of the most likely $\mathbf{x}$ and therefore detect whether there is some sufficiently large $f_L(\mathbf{x})$. In order to approximate this quantity within ε and with probability at least $1 - \nu$, we need N samples, where (see Appendix D)

$$N = O(\ln(1/\nu)/\varepsilon^2), \qquad \varepsilon = \frac{\eta^+ - \eta^-}{2}, \qquad \eta^{\pm} = \frac{(\delta^{\pm})^2}{Z \cdot |G|}. \tag{9}$$

In the case of the dual attack, the parameters (6) do not completely characterize the state and we need to take into account the relation between k and $|G|$.

Assuming that $k \ll |G|$, then the $\mathbf{u}_j$ are distinct with high probability and therefore, by (8), $Z = k$ since $|w_j| = 1$ because all inputs coefficients. Therefore,

$$\varepsilon = \frac{\eta^+ - \eta^-}{2} = \frac{(\delta^+)^2 - (\delta^-)^2}{2|G|Z} = \frac{(\delta^+ - \delta^-)(\delta^+ + \delta^-)}{2|G|Z} = \Theta\left(\frac{1}{|G|}\right).$$

Hence, to succeed with probability $1 - \nu$, we need

$$N = \Theta(|G|^2 \ln(1/\nu)) \tag{10}$$

samples. It is not clear if there is a better strategy that merely decides on the presence of some $\mathbf{x}$ without recovering it. Note that (10) is even worse than the classical complexity.

Summary

We have introduced a problem, and its promise version, related to FFT on a sparse input. We have explained how our quantum algorithm for the dual attacks solves the promise version more efficiently than the classical algorithm but requires access to a QRACM. We have also given an alternative quantum algorithm that does not require access to a QRACM but which is always worse than our algorithm and not always better than the classical algorithm (depending on the value of k). Specifically, we have the following algorithms to solve the promise problem:

- classical: $O(|G| \log |G|)$ plus $O(k)$ for array filling,
- quantum (no QRACM): $O(k\sqrt{|G|})$,
- quantum (QRACM): $O(k\sqrt{|G|/(\delta^+ - \delta^-)})$ plus $O(k)$ for QRACM filling.

Which one is better depends on the choice of the parameters. Clearly, $\Omega(k)$ is a lower bound on the complexity since the algorithm needs to read the input in any case. We have also explained why the natural algorithm based on the QFT and measurements is seemingly always worse than all of the above.

In the context of dual attacks, the quantum algorithm with QRACM above has complexity $O(\sqrt{k|G|})$ for parameters (6). Therefore the following two questions regarding `PROMISE-INPUT-SPARSE-FFT-THRESHOLD`:

- Is the quantum complexity $O(\sqrt{k|G|})$ optimal with a QRACM ?
- Can we achieve quantum complexity better than $O(|G| \log |G| + k)$ when $k = \Omega(\sqrt{|G|})$ and without a QRACM?

Another open problem is to prove a matching lower bound for the (classical and quantum) guessing complexity of a discrete Gaussian. See Lemma 4 and the remark below for more details.

Acknowledgements

We thank André Chailloux, Yanlin Chen, Thomas Debris-Alazard, Yassine Hamoudi, André Schrottenloher, Amaury Pouly, Jean-Pierre Tillich for helpful discussions. We thank Erik Mårtensson and Alessandro Budroni for alerting us to a bug in the estimation code contained and used in a previous version of this work. 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. arXiv:2002.07955.

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.

Ban93. W. Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296(4):625–636, 1993.

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.

BHMT02. Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Information*, page 53–74, 2002.

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`.

DKW56. A. Dvoretzky, J. Kiefer, and J. Wolfowitz. Asymptotic Minimax Character of the Sample Distribution Function and of the Classical Multinomial Estimator. *The Annals of Mathematical Statistics*, 27(3):642 – 669, 1956.

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.

GJS15. Qian Guo, Thomas Johansson, and Paul Stankovski. Coded-BKW: Solving LWE using lattice codes. In Gennaro and Robshaw [GR15], pages 23–42.

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

GR15. Rosario Gennaro and Matthew J. B. Robshaw, editors. *CRYPTO 2015, Part I*, volume 9215 of *LNCS*. Springer, Heidelberg, August 2015.

Gro96. Lov K. Grover. A fast quantum mechanical algorithm for database search. In *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.

KF15. Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Gennaro and Robshaw [GR15], pages 43–62.

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.

Mas90. P. Massart. The Tight Constant in the Dvoretzky-Kiefer-Wolfowitz Inequality. *The Annals of Probability*, 18(3):1269 – 1283, 1990.

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. Available at `https://doi.org/10.5281/zenodo.6412487`, 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.

MKF19. Kosuke Mitarai, Masahiro Kitagawa, and Keisuke Fujii. Quantum analog-digital conversion. *Phys. Rev. A*, 99:012301, Jan 2019.

Mon11. Ashley Montanaro. Quantum search with advice. In Wim van Dam, Vivien M. Kendon, and Simone Severini, editors, *Theory of Quantum Computation, Communication, and Cryptography*, pages 77–93, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

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 3

We intuitively want to call the algorithm from Lemma 1 directly but there is a small issue. Indeed, with small probability, the algorithm might not find the marked element and run in time $\sqrt{|E|}$ which will make average execution time bigger than what we want. Instead, we search in a subset of the array that we double at each failure. This ensures that the "worst-case" cost remains under control.

Algorithm 4: Quantum guessing algorithm with bounded error oracle

Input: size N of E, order σ and bounded-error oracle $\mathcal{O}$
Output: (guess for) i such that $\mathcal{O}(i) = 1$, or $\perp$

create oracle $\mathcal{O}'(i)$**:**
  return $\mathcal{O}(\sigma(i))$
$n \leftarrow 1$
while $n < 2N$ **do**
  Use Lemma 1 to find $i \in [1, \min(n, N)]$ such that $\mathcal{O}'(i) = 1$, or $i = \perp$
  if $i \neq \perp$ **then**
    Call $1 + 2\log(n)$ times $\mathcal{O}'(i)$ and let $b \in \{0, 1\}$ be the majority answer
    if $b = 1$ **then**
      return i
  $n \leftarrow 2n$
return $\perp$

Consider Algorithm 4, which we call with σ and $\mathcal{O}_x$ for some fixed $x \in E$. Note that σ is surjective so there exists i such that $\sigma(i) = x$ and then $\mathcal{O}'(i) = 1$ with probability at least 9/10. Since σ is injective, $\sigma(j) \neq x$ for $j \neq i$ so $\mathcal{O}'(j) = 0$ with probability at least 9/10. Therefore, $\mathcal{O}'$ computes, with bounded error at most 1/10, the function $f_x(x) = 1$ and $f_x(y) = 0$ for all $y \neq x$. Let i_x be such that $\sigma(i_x) = x$, and p_x such that $2^{p_x - 1} < i_x \leqslant 2^{p_x}$.

Let $k < p_x$ and let us analyse the k^{th} iteration of the loop. During the iteration, $n = 2^{k-1}$ so $n < i_x$ and therefore, $f_x(j) = 0$ for all $j \in [1, n]$. It follows that Lemma 1 will return $\perp$ with probability at least 9/10 and then the loop will continue to its next iteration. With probability at most 1/10, it will return a (wrong) index $i \neq \perp$. But $f_x(i) = 0$ so each call to $\mathcal{O}'(i)$ will return 0 with probability at least 9/10. Therefore with probability at most 10^{-k}, a majority of the $1 + 2k$ calls will return 0 and the loop with continue. In summary, the loop will return a wrong index only with probability at most 10^{-k-1}. In all cases, the cost of this iteration is $O(\sqrt{2^k})$ queries/time for the search, plus an extra $1 + 2k$ calls to the oracle.

It follows that with probability at most $\sum_{k=1}^{p_x-1} 10^{-k-1} \leqslant 1/9$, the algorithm will stop during on the first p_x iterations and return a wrong answer. Hence, with probability at least $8/9$, the algorithm reaches the p_x^{th} iterations.

Now assume that the algorithm has reached the p_x^{th} iteration. Let $p_x \leqslant k \leqslant 1+\log_2(N)$ and let us analyse the k^{th} iteration of the loop. During the iteration, $n = 2^{k-1}$ so $i_x \in [1, \min(n, N)]$ and $f_x(j) = 0$ for all $j \neq i_x$. It follows that Lemma 1 will return i_x with probability at least $9/10$. Since $f_x(i_x) = 1$, each call to $\mathcal{O}'(i_x)$ will return 1 with probability at least $9/10$. Therefore with probability at least $1 - 10^{-k} \geqslant 9/10$, a majority of the $1 + 2k$ calls will return 1 and the algorithm will return i_x. In summary, with probability at least $(9/10)^2 \geqslant 4/5$, the algorithm stops and return i_x during the k^{th} iteration. Conversely, with probability at most $1/5$, the algorithm will either return a wrong answer or continue to the next iteration.

This analysis shows that the algorithm is correct since already with probability at least $8/9$ the algorithm will reach p_x^{th} iteration and then with probability at least $4/5$ it will return i_x during that iteration, so it returns i_x with probability at least $32/45$.

We now analyse the complexity: the cost of the k^{th} iteration is the cost of the search and (maybe) the $1 + 2k$ calls to the oracle, which overall is

$$O(\sqrt{2^{k-1}} + 1 + 2k) = O(\sqrt{2^k})$$

queries/time. To simplify the analysis, we can always assume that the first $p_x - 1$ iterations are done in all cases (this is clearly an upper bound in the case where the algorithm stops too early). We can also assume that the while loop does not stop at $n < 2N$ but instead goes on forever (again this is clearly an upper bound on the real cost). For each $k \geqslant p_x$, the k^{th} iteration only occurs with probability at most $5^{-(k-p_x+1)}$ by the analysis above. Therefore, the expected running time/number of queries of the algorithm is bounded by

$$\begin{aligned} O\left(\sum_{k=1}^{p_x-1} \sqrt{2^k} + \sum_{k=p_x}^{\infty} 5^{-(k-p_x+1)}\sqrt{2^k}\right) &= O\left(\sqrt{2^{p_x}} + \sqrt{2^{p_x}} \sum_{k=0}^{\infty} (\sqrt{2}/5)^k\right) \\ &= O\left(\sqrt{2^{p_x}}\right) \\ &= O\left(\sqrt{i_x}\right). \end{aligned}$$

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 drawn from X. Then every x is chosen with probability $\Pr_X[X = x]$ and, when this is the case, the returned index is $i_x = \sigma^{-1}(x)$. 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. □

B Proof of Lemma 4

For reasons that will be clear in the proof of the bounds, we introduce for all $s > 0$ and $n \in \mathbb{N}$,

$$g_{n,s}(\sigma) := \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \cdot N(\ell)^s$$

where $N(\ell) := N_n(\ell) = \left|\{\, \mathbf{x} \in \mathbb{Z}^n : \|\mathbf{x}\|^2 \leqslant \ell \,\}\right|$. Note that $N(-1) = 0$. We also bound $\rho_\sigma(\mathbb{Z}^n)$ by using the Poisson summation formula as follows:

$$\rho_\sigma(\mathbb{Z}^n) = (\rho_\sigma(\mathbb{Z}))^n = \left(\sigma\sqrt{2\pi}\rho_{1/2\pi\sigma}(\mathbb{Z})\right)^n > (\sigma\sqrt{2\pi})^n \tag{11}$$

and

$$\rho_{1/2\pi\sigma}(\mathbb{Z}) = 1 + 2\sum_{n=1}^{\infty} e^{-2\pi^2\sigma^2 n^2} \leqslant 1 + 2\sum_{n=1}^{\infty} e^{-2\pi^2\sigma^2 n} = \coth(\pi^2\sigma^2). \tag{12}$$

Also recall [Ban93] that

$$\rho_s(\mathcal{L}) \leqslant s^n \cdot \rho(\mathcal{L}) \tag{13}$$

for any $s \geqslant 1$ and lattice $\mathcal{L} \subset \mathbb{R}^n$.

B.1 Generic bounds on $g_{n,k}(\sigma)$ for k integer

We start by obtaining a general relationship between the $g_{n,k}$ and $g_{n,1}$ where k is an integer. A simple counting argument shows that $N(\ell)^k \leqslant N_{kn}(k\ell)$. Indeed, the map $\phi : (\mathbf{x}_1, \ldots, \mathbf{x}_k) \in (\mathbb{Z}^n)^k \to (\mathbf{x}_1, \cdots, \mathbf{x}_k) \in \mathbb{Z}^{kn}$ is injective and if $\|\mathbf{x}_1\|^2, \ldots, \|\mathbf{x}_k\|^2 \leqslant \ell$ then $\|\phi(\mathbf{x}_1, \ldots, \mathbf{x}_k)\|^2 = \|\mathbf{x}_1\|^2 + \cdots + \|\mathbf{x}_k\|^2 \leqslant k\ell$. Therefore,

$$g_{n,k}(\sigma) \leqslant \sum_{\ell=0}^{\infty} N_{kn}(k\ell) \frac{e^{-\frac{\ell}{2\sigma^2}}}{\rho_\sigma(\mathbb{Z}^n)} = \frac{1}{\rho_\sigma(\mathbb{Z}^n)} H^0_{kn}(e^{-1/2\sigma^2})$$

where

$$H^i_n(x) = \sum_{\ell=0}^{\infty} N_n(k\ell + i) x^\ell.$$

Furthermore,

$$H^0_n(x^k) + xH^1_n(x^k) + \cdots + x^{k-1}H^{k-1}_n(x^k) = \sum_{\ell=0}^{\infty} N_n(\ell) \cdot x^\ell.$$

Since $N_n(\ell)$ is increasing with ℓ, we immediately have that $H^0_n(x) \leqslant H^i_n(x)$ for all i so

$$H^0_n(x) \leqslant \frac{1}{1 + x^{1/k} + \cdots + x^{(k-1)/k}} \sum_{\ell=0}^{\infty} N_n(\ell) \cdot \sqrt[k]{x}^{\ell}.$$

In particular,

$$\begin{aligned} g_{n,k}(\sigma) &\leqslant \frac{1}{\rho_\sigma(\mathbb{Z}^n)} \frac{1}{\sum_{i=0}^{k-1} e^{-i/2k\sigma^2}} \sum_{\ell=0}^{\infty} N_{kn}(\ell) \cdot e^{-\ell/2k\sigma^2} \\ &= \frac{\rho_{\sqrt{k}\sigma}(\mathbb{Z}^{kn})}{\rho_\sigma(\mathbb{Z}^n)} \frac{1}{\sum_{i=0}^{k-1} e^{-i/2k\sigma^2}} \sum_{\ell=0}^{\infty} \frac{N_{kn}(\ell)}{\rho_{\sqrt{k}\sigma}(\mathbb{Z}^{kn})} \cdot e^{-\ell/2(\sqrt{k}\sigma)^2} \\ &= \frac{\rho_{\sqrt{k}\sigma}(\mathbb{Z}^{kn})}{\rho_\sigma(\mathbb{Z}^n)} \frac{1}{\sum_{i=0}^{k-1} e^{-i/2k\sigma^2}} \cdot g_{kn,1}(\sqrt{k}\sigma). \end{aligned}$$

It is well-known (check by developing $(1-x)^{-1}$ into its series) that

$$\sum_{\ell=0}^{\infty} N_n(\ell) \cdot x^\ell = \frac{1}{1-x} \sum_{\ell=0}^{\infty} S_n^L(\ell) \cdot x^\ell, \qquad S_n^L(\ell) = \left| \{ \mathbf{x} \in \mathbb{Z}^n : \|\mathbf{x}\|^2 = \ell \} \right|.$$

On the other hand, for $x = e^{-1/2\sigma^2}$, $\sum_{\ell=0}^{\infty} S_n^L(\ell) \cdot x^\ell = \rho_\sigma(\mathbb{Z}^n)$. Therefore,

$$g_{n,1}(\sigma) = \frac{1}{1 - e^{-1/2\sigma^2}} \tag{14}$$

Therefore

$$\begin{aligned} g_{n,k}(\sigma) &\leqslant \frac{\sqrt{k}^{kn} \rho_\sigma(\mathbb{Z})^{(k-1)n}}{\sum_{i=0}^{k-1} e^{-i/2k\sigma^2}} \cdot g_{kn,1}(\sqrt{k}\sigma) && \text{by (13)} \\ &= \frac{\sqrt{k}^{kn} \rho_\sigma(\mathbb{Z})^{(k-1)n}}{\sum_{i=0}^{k-1} e^{-i/2k\sigma^2}} \cdot \frac{1}{1 - e^{-1/2(\sqrt{k}\sigma)^2}} && \text{by (14)} \\ &= \frac{\sqrt{k}^{kn} \rho_\sigma(\mathbb{Z})^{(k-1)n}}{1 - e^{-1/2\sigma^2}}. && (15) \end{aligned}$$

Note that this equation also holds for $k = 1$ by (14). In fact, it even holds for $k = 0$, with the convention that $0^0 = 1$. since

$$g_{n,0}(\sigma) = \frac{1}{\rho_\sigma(\mathbb{Z}^n)} \sum_{\ell=0}^{\infty} e^{-\ell/2\sigma^2} = \frac{1}{\rho_\sigma(\mathbb{Z}^n)} \frac{1}{1 - e^{-1/2\sigma^2}}. \tag{16}$$

B.2 Bound on $G(D_{\mathbb{Z}^n,\sigma})$

Observing that $D_{\mathbb{Z}^n,\sigma}$ decreases with $\|\mathbf{x}\|$, we rewrite the guessing complexity as

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

It follows that, for $n \geqslant 4$ (we need every number to be a sum of n squares so that $N(\ell) > N(\ell-1)$):

$$G(D_{\mathbb{Z}^n,\sigma}) = \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \cdot \frac{(N(\ell) - N(\ell-1))(N(\ell-1) + N(\ell) + 1)}{2} \tag{17}$$

$$
\begin{aligned}
&\leqslant \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \cdot (N(\ell) - N(\ell-1)) \cdot N(\ell) \\
&\leqslant g_{n,2}(\sigma) \\
&\leqslant \frac{2^n \rho_\sigma(\mathbb{Z})^n}{1 - e^{-1/2\sigma^2}} && \text{by (15).} \qquad (18)
\end{aligned}
$$

B.3 Bound on $G^{qc}(D_{\mathbb{Z}^n,\sigma})$

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

$$
G^{qc}(D_{\mathbb{Z}^n,\sigma}) = \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \sum_{i=N(\ell-1)+1}^{N(\ell)} \sqrt{i}. \qquad (19)
$$

Now check by induction on b that

$$
\sum_{i=0}^{b} \sqrt{i} \leqslant \tfrac{2}{3} n^{3/2} + \tfrac{1}{2}\sqrt{n}.
$$

It follows that

$$
\begin{aligned}
G^{qc}(D_{\mathbb{Z}^n,\sigma}) &\leqslant \frac{2}{3} \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} N(\ell)^{3/2} + \frac{1}{2} \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} \sqrt{N(\ell)} \\
&= \tfrac{2}{3} g_{n,3/2}(\sigma) + \tfrac{1}{2} g_{n,1/2}(\sigma).
\end{aligned}
$$

Now check, that for $\alpha = 2/3$,

$$
\begin{aligned}
g_{n,3/2}(\sigma) &= \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} N(\ell)^{3/2} \\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)} \sum_{\ell=0}^{\infty} \sqrt{\rho_\sigma(\sqrt{\ell})^\alpha N(\ell)}^3 \\
&\leqslant \frac{1}{\rho_\sigma(\mathbb{Z}^n)} \sqrt{\sum_{\ell=0}^{\infty} \rho_\sigma(\sqrt{\ell})^\alpha N(\ell)}^3 && \text{by Cauchy–Schwarz} \\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)} \sqrt{\sum_{\ell=0}^{\infty} \rho_{\sigma/\sqrt{\alpha}}(\sqrt{\ell}) N(\ell)}^3 \\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)} \sqrt{\rho_{\sigma/\sqrt{\alpha}}(\mathbb{Z}^n) g_{n,1}(\tfrac{\sigma}{\sqrt{\alpha}})}^3 \\
&= \frac{\rho_{\sigma/\sqrt{\alpha}}(\mathbb{Z}^n)^{3/2}}{\rho_\sigma(\mathbb{Z}^n)} \frac{1}{(1 - e^{-1/3\sigma^2})^{3/2}} && \text{by (14)} \\
&\leqslant \frac{\rho_\sigma(\mathbb{Z}^n)^{3/2}}{\sqrt{\alpha}^{3n/2} \rho_\sigma(\mathbb{Z}^n)} \frac{1}{(1 - e^{-1/3\sigma^2})^{3/2}} && \text{by (13) since } 1/\sqrt{\alpha} > 1
\end{aligned}
$$

$$= \left(\frac{3}{2}\right)^{3n/4} \frac{\sqrt{\rho_\sigma(\mathbb{Z}^n)}}{(1-e^{-1/3\sigma^2})^{3/2}} \tag{20}$$

Similarly,

$$\begin{aligned}
g_{n,1/2}(\sigma) &= \sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} N(\ell)^{1/2} \\
&= \sum_{\ell=0}^{\infty} \sqrt{\frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} N(\ell)} \cdot \sqrt{\frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)}} \\
&\leqslant \sqrt{\sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)} N(\ell)} \sqrt{\sum_{\ell=0}^{\infty} \frac{\rho_\sigma(\sqrt{\ell})}{\rho_\sigma(\mathbb{Z}^n)}} \\
&= \sqrt{g_{n,1}(\sigma) g_{n,0}(\sigma)} \\
&= \frac{1}{1-e^{-1/2\sigma^2}} \frac{1}{\sqrt{\rho_\sigma(\mathbb{Z}^n)}} \qquad \text{by (14) and (16).}
\end{aligned}$$

It follows by (20) that

$$\begin{aligned}
G^{qc}(D_{\mathbb{Z}^n,\sigma}) &\leqslant \frac{2}{3} \cdot \left(\frac{3}{2}\right)^{3n/4} \frac{\sqrt{\rho_\sigma(\mathbb{Z}^n)}}{(1-e^{-1/3\sigma^2})^{3/2}} + \frac{1}{2} \cdot \frac{1}{1-e^{-1/2\sigma^2}} \frac{1}{\sqrt{\rho_\sigma(\mathbb{Z}^n)}} \\
&= \frac{2}{3} \cdot \left(\frac{3}{2}\right)^{3n/4} \frac{\sqrt{\rho_\sigma(\mathbb{Z}^n)}}{(1-e^{-1/3\sigma^2})^{3/2}} \\
&\quad \times \left(1 + \frac{3}{4} \cdot \left(\frac{2}{3}\right)^{3n/4} \frac{(1-e^{-1/3\sigma^2})^{3/2}}{1-e^{-1/2\sigma^2}} \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\right) \\
&\leqslant \frac{2}{3} \cdot \left(\frac{3}{2}\right)^{3n/4} \frac{\sqrt{\rho_\sigma(\mathbb{Z}^n)}}{(1-e^{-1/3\sigma^2})^{3/2}} \left(1 + \frac{3}{4} \cdot \left(\frac{2}{3}\right)^{3n/4} \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\right) \\
&\leqslant \frac{2}{3} \cdot \left(\frac{3}{2}\right)^{3n/4} \frac{\sqrt{\rho_\sigma(\mathbb{Z}^n)}}{(1-e^{-1/3\sigma^2})^{3/2}} \left(1 + \frac{3}{4} \cdot \left(\frac{2^{1/4}}{3^{3/4}\sigma\sqrt{\pi}}\right)^n\right) \qquad \text{by (11)} \\
&\leqslant \frac{7}{6} \cdot \left(\frac{3}{2}\right)^{3n/4} \frac{\sqrt{\rho_\sigma(\mathbb{Z}^n)}}{(1-e^{-1/3\sigma^2})^{3/2}} \qquad \text{when } \sigma \geqslant \sqrt[4]{\frac{2}{27\pi^2}}. \quad (21)
\end{aligned}$$

B.4 Bound on $H(D_{\mathbb{Z}^n,\sigma})$

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

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

$$= \frac{1}{2\sigma^2 \log(2) \rho_\sigma(\mathbb{Z}^n)} \sum_{x \in \mathbb{Z}^n} \rho_\sigma(x) \cdot \|x\|^2 + \log_2 \rho_\sigma(\mathbb{Z}^n).$$

Now note that

$$\begin{aligned}
\sum_{x \in \mathbb{Z}^n} \rho_\sigma(x) \cdot \|x\|^2 &= \sum_{k=1}^{n} \sum_{x \in \mathbb{Z}^n} x_k^2 \prod_{j=1}^{k} \rho_\sigma(x_j) \\
&= \sum_{k=1}^{n} \left(\sum_{x \in \mathbb{Z}} \rho_\sigma(x) \right)^{n-1} \sum_{x \in \mathbb{Z}} x^2 \rho_\sigma(x) \\
&= n \, \rho_\sigma(\mathbb{Z})^{n-1} \sum_{x \in \mathbb{Z}} f_\sigma(x)
\end{aligned}$$

where $f_\sigma(x) = x^2 \rho_\sigma(x)$. Hence, since $\rho_s(\mathbb{Z}^n) = \rho_s(\mathbb{Z})^n$,

$$H(D_{\mathbb{Z}^n,\sigma}) = \frac{n}{2\sigma^2 \log(2) \rho_\sigma(\mathbb{Z})} \sum_{x \in \mathbb{Z}} f_\sigma(x) + n \log_2 \rho_\sigma(\mathbb{Z}). \tag{22}$$

It is not hard to check that the Fourier transform of f_σ is

$$\widehat{f_\sigma}(x) = e^{-2\pi^2\sigma^2x^2} \sigma^3 \sqrt{2\pi} \left(1 - 4\pi^2\sigma^2x^2\right).$$

Therefore, by the Poisson summation formula

$$\begin{aligned}
\sum_{x \in \mathbb{Z}} f_\sigma(x) &= \sigma^3 \sqrt{2\pi} \left(\sum_{x \in \mathbb{Z}} e^{-2\pi^2\sigma^2x^2} - 4\pi^2\sigma^2 \sum_{x \in \mathbb{Z}} e^{-2\pi^2\sigma^2x^2} x^2 \right) \\
&= \sigma^3 \sqrt{2\pi} \left(\rho_{1/2\pi\sigma}(\mathbb{Z}) - 4\pi^2\sigma^2 \sum_{x \in \mathbb{Z}} e^{-2\pi^2\sigma^2x^2} x^2 \right) \\
&= \sigma^3 \sqrt{2\pi} \left(\frac{1}{\sigma\sqrt{2\pi}} \rho_\sigma(\mathbb{Z}) - 4\pi^2\sigma^2 \sum_{x \in \mathbb{Z}} e^{-2\pi^2\sigma^2x^2} x^2 \right) \qquad \text{by (11)} \\
&= \sigma^2 \rho_\sigma(\mathbb{Z}) - 4\sqrt{2\pi}\pi^2\sigma^5 \sum_{x \in \mathbb{Z}} e^{-2\pi^2\sigma^2x^2} x^2 \qquad (23)
\end{aligned}$$

Hence, we have that

$$\begin{aligned}
H(D_{\mathbb{Z}^n,\sigma}) &\leqslant \frac{n}{2\log(2)} + n \log_2 \rho_\sigma(\mathbb{Z}) && \text{by (22) and (23)} \\
&\leqslant \frac{n}{2\log(2)} + n \log_2(\sigma\sqrt{2\pi} \coth(\pi^2\sigma^2)) && \text{by (11) and (12).} \qquad (24)
\end{aligned}$$

Now check that for any $\alpha > 0$,

$$\sum_{x \in \mathbb{Z}} e^{-\alpha x^2} x^2 \leqslant 2 \sum_{n=1}^{\infty} e^{-\alpha n} n^2 = \frac{2e^\alpha(1+e^\alpha)}{(e^\alpha - 1)^3} \leqslant 4e^{-\alpha}.$$

Hence, for $\alpha = 2\pi^2\sigma^2$,

$$4\sqrt{2\pi}\pi^2\sigma^5 \sum_{x\in\mathbb{Z}} e^{-2\pi^2\sigma^2 x^2} x^2 \leqslant 16\sqrt{2\pi}\pi^2\sigma^5 e^{-2\pi^2\sigma^2}.$$

From this, (22) and (23) we conclude that,

$$\begin{aligned}
H(D_{\mathbb{Z}^n,\sigma}) &\geqslant \frac{n}{2\log(2)} + n\log_2 \rho_\sigma(\mathbb{Z}) - \frac{n\cdot 16\sqrt{2\pi}\pi^2\sigma^5 e^{-2\pi^2\sigma^2}}{2\sigma^2\log(2)\rho_\sigma(\mathbb{Z})} \\
&= \frac{n}{2\log(2)} + n\log_2 \rho_\sigma(\mathbb{Z}) - n\frac{8\sqrt{2\pi}\pi^2\sigma^3 e^{-2\pi^2\sigma^2}}{\log(2)\rho_\sigma(\mathbb{Z})} \\
&\geqslant \frac{n}{2\log(2)} + n\log_2 \rho_\sigma(\mathbb{Z}) - n\frac{8\sqrt{2\pi}\pi^2\sigma^3 e^{-2\pi^2\sigma^2}}{\log(2)\sigma\sqrt{2\pi}\coth(\pi^2\sigma^2)} \quad \text{by (12)} \\
&= \frac{n}{2\log(2)} + n\log_2 \rho_\sigma(\mathbb{Z}) - n\frac{8\pi^2\sigma^2 e^{-2\pi^2\sigma^2}}{\log(2)\coth(\pi^2\sigma^2)}.
\end{aligned}$$

Therefore,

$$2^{H(D_{\mathbb{Z}^n,\sigma})} \geqslant e^{n/2}\rho_\sigma(\mathbb{Z})^n e^{-n\frac{8\pi^2\sigma^2 e^{-2\pi^2\sigma^2}}{\coth(\pi^2\sigma^2)}}. \tag{25}$$

B.5 Relation between $H(D_{\mathbb{Z}^n,\sigma})$, $G(D_{\mathbb{Z}^n,\sigma})$ and $G^{qc}(D_{\mathbb{Z}^n,\sigma})$

Recall that by (18),

$$G(D_{\mathbb{Z}^n,\sigma}) \leqslant \frac{2^n\rho_\sigma(\mathbb{Z})^n}{1-e^{-1/2\sigma^2}}.$$

Therefore, by (25),

$$\frac{2^{H(D_{\mathbb{Z}^n,\sigma})}}{G(D_{\mathbb{Z}^n,\sigma})} \geqslant (1-e^{-1/2\sigma^2})\left(\frac{\sqrt{e}}{2a(\sigma)}\right)^n, \qquad a(\sigma) = e^{8\pi^2\sigma^2 e^{-2\pi^2\sigma^2}\tanh(\pi^2\sigma^2)}.$$

All asymptotics in what follows are as $\sigma\to\infty$. We introduce $\alpha = 2\pi^2\sigma^2$ to simplify calculations. Recall that $\tanh(\pi^2\sigma^2) = 1-2e^{-\alpha}+o(e^{-\alpha})$. Hence,

$$8\pi^2\sigma^2 e^{-2\pi^2\sigma^2}\tanh(\pi^2\sigma^2) = 4\alpha e^{-\alpha}(1-2e^{-\alpha}+o(e^{-\alpha})).$$

Note that this quantity goes to 0 as $\alpha\to\infty$. It follows that

$$\begin{aligned}
a(\sigma) &= e^{4\alpha e^{-\alpha}(1-2e^{-\alpha}+o(e^{-\alpha}))} \\
&= 1 + 4\alpha e^{-\alpha}(1-2e^{-\alpha}+o(e^{-\alpha})) + o(\alpha e^{-\alpha}) \\
&= 1 + 4\alpha e^{-\alpha} + o(\alpha e^{-\alpha}).
\end{aligned}$$

A similar analysis holds for $G^{qc}(D_{\mathbb{Z}^n,\sigma})$. Recall that for $\sigma \geqslant \sqrt[4]{\frac{2}{27\pi^2}}$,

$$G^{qc}(D_{\mathbb{Z}^n,\sigma}) \leqslant \frac{7}{6}\cdot\left(\frac{3}{2}\right)^{3n/4}\frac{\sqrt{\rho_\sigma(\mathbb{Z}^n)}}{(1-e^{-1/3\sigma^2})^{3/2}}.$$

Therefore, for $\sigma \geqslant \sqrt[4]{\frac{2}{27\pi^2}}$, by (25),

$$\frac{2^{H(D_{\mathbb{Z}^n,\sigma})/2}}{G^{qc}(D_{\mathbb{Z}^n,\sigma})} \geqslant \frac{6}{7}(1-e^{-1/3\sigma^2})^{3/2}\left(\frac{8e}{27a(\sigma)^2}\right)^{n/4}$$

where $a(\sigma)$ was defined above.

B.6 Bounds on $G(D_{\mathbb{Z}_q^n,\sigma})$ and $G^{qc}(D_{\mathbb{Z}_q^n,\sigma})$

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 $G(D_{\mathbb{Z}_q^n,\sigma})$. Our proof strategy is to relate the guessing complexity of $D_{\mathbb{Z}_q^n,\sigma}$ to that of $D_{\mathbb{Z}^n,\sigma}$ and use the results proven in the previous subsections.

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, however, do not require an optimal order to obtain an upper bound. We now have that

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

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 we 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,

$$G(D_{\mathbb{Z}_q^n,\sigma}) \leqslant \frac{1}{\rho_\sigma(\mathbb{Z}^n)}\sum_{\mathbf{x}\in\mathbb{Z}^n}\rho_\sigma(\mathbf{x})N(\|\tilde{\mathbf{x}}\|^2)$$

$$
\begin{aligned}
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)} \sum_{\ell=0}^{\infty} \sum_{\mathbf{x}\in\mathbb{Z}^n : \|\mathbf{x}\|^2=\ell} \rho_\sigma(\mathbf{x}) N(\|\tilde{\mathbf{x}}\|^2) \\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)} \sum_{\ell=0}^{\infty} \rho_\sigma(\sqrt{\ell}) \cdot N(\ell) \sum_{\mathbf{x}\in\mathbb{Z}^n : \|\mathbf{x}\|^2=\ell} 1 \\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^n)} \sum_{\ell=0}^{\infty} \rho_\sigma(\sqrt{\ell}) \cdot N(\ell) \cdot (N(\ell) - N(\ell-1)) \\
&\leqslant 2G(D_{\mathbb{Z}^n,\sigma}) \qquad \text{by (17).}
\end{aligned}
$$

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 (19) 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 easily follows 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. □

C Proofs for Section 5

C.1 Proof of Theorem 6

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 9/10, 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 $\tilde{\mathbf{s}}_{\text{fft}}$ 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 $F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) > C$. Therefore, $x \in S_C^L$. Hence, this proves that $x \in S_C^L \cup \{\perp\}$ with probability at least 9/10. Now assume that $S^L_{(1+2\eta)C} \neq \varnothing$ and let $\tilde{\mathbf{s}}_{\text{enum}} \in S^L_{(1+2\eta)C}$. 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

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

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

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

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

$$= \frac{1}{D}\sum_j \cos\left(\frac{2\pi}{p}\langle \mathcal{O}_W(j), (\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1)\rangle\right)$$

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

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

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

$$= \frac{1}{D}\,F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}})$$

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 $\mathcal{A}^{\mathcal{O}'_W}((\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}, 1)) \leqslant \frac{C}{D} + \frac{C}{D}\eta = (1+\eta)\frac{C}{D}$ 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. □

C.2 Proof of Lemma 5

Recall that for any list L and any $x > 0$, we defined

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

In the proof of [MAT22, Theorem 5.2], it is shown that for any threshold[7] X,

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

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 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,

$$\Pr_L[F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) \leqslant C \text{ for the first } 2N_{\text{enum}}(\mathbf{s}_{\text{enum}}) \text{ values of } \tilde{\mathbf{s}}_{\text{enum}}] \geq 1 - \mu. \tag{26}$$

On the other hand,

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

It is easy to check by taking the derivative that Φ, the cdf of the normal distribution, satisfies the following inequality for all $y \geq 0$:

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

Furthermore, Φ is an increasing function so Φ^{-1} is also increasing. Hence,

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

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

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

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

Therefore,

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

From this we get that $\phi_{\mathrm{fp}}(\mu)$ is a polynomial factor in all the relevant parameters. Now observe that by the integral definition of $\Phi()$,

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

when

$$\eta \leqslant \frac{\sqrt{2\pi}\mu}{8\phi_{\mathrm{fp}}(\mu)}.$$

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 (27), with probability at least $1-\mu$, this element must be $\mathbf{s}_{\mathrm{enum}}$ because all the other elements satisfy $F_L(\tilde{\mathbf{s}}_{\mathrm{enum}}, \tilde{\mathbf{s}}_{\mathrm{fft}}) \leqslant C$. Therefore, by a union bound, the probability that the algorithm returns $\mathbf{s}_{\mathrm{enum}}$ is at least $1 - \frac{3\mu}{4} - \mu \geq 1 - 2\mu \geq 1 - \nu$. This concludes the proof. □

C.3 Proof of Lemma 5

Recall that for any list L and any $x > 0$, we defined

$$S^L_x = \left\{ \tilde{\mathbf{s}}_{\mathrm{enum}} : \exists\ \tilde{\mathbf{s}}_{\mathrm{fft}}, F_L(\tilde{\mathbf{s}}_{\mathrm{enum}}, \tilde{\mathbf{s}}_{\mathrm{fft}}) > x \right\}.$$

In the proof of [MAT22, Theorem 5.2], it is shown that for any threshold[8] X,

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

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

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

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

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

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

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 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,

$$\Pr_L[F_L(\tilde{\mathbf{s}}_{\text{enum}}, \tilde{\mathbf{s}}_{\text{fft}}) \leqslant C \text{ for the first } 2N_{\text{enum}}(\mathbf{s}_{\text{enum}}) \text{ values of } \tilde{\mathbf{s}}_{\text{enum}}] \geq 1 - \mu. \tag{27}$$

On the other hand,

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

It is easy to check by taking the derivative that Φ, the cdf of the normal distribution, satisfies the following inequality for all $y \geq 0$:

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

Furthermore, Φ is an increasing function so Φ^{-1} is also increasing. Hence,

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

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

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

Therefore,

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

From this we get that $\phi_{\text{fp}}(\mu)$ is a polynomial factor in all the relevant parameters. Now observe that by the integral definition of $\Phi()$,

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

when

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

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 (27), 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. □

D Quantum algorithm w/o QRACM for the FFT threshold problem

Here, for simplicity, we assume that we want to decide if $|f_L(\mathbf{x})| > \delta^+$, under the promise that $|f_L(\mathbf{x})| \notin [\delta^-, \delta^+]$. In the original problem, we were using the real part. We focus on the PROMISE-INPUT-SPARSE-FFT-THRESHOLD problem. Assume that we can create the superposition

$$|\psi\rangle = \frac{1}{\sqrt{Z}} \sum_j w_j \, |\mathbf{u}_j\rangle \, . \tag{28}$$

We then perform a QFT on $|\psi\rangle$ to obtain

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

If we measure $|\widehat{\psi}\rangle$, we obtain $\mathbf{x}$ with probability

$$p(\mathbf{x}) = \frac{|\widehat{f_L}(\mathbf{x})|^2}{|G| \cdot Z}.$$

Consider the random variable X defined by $\Pr[X = \mathbf{x}] = p(\mathbf{x})$. Let us reformulate the problem:

- We have access to a sampler for X.
- We do not know the values of $p(\mathbf{x})$.
- $p(\mathbf{x}) \notin [\eta^-, \eta^+]$ for some given η^-, η^+.
- We want to decide if there exists $\mathbf{x} \in G$ such that $p(\mathbf{x}) > \eta^+$.

We can relate the new parameters to the old one as follows:

$$\begin{aligned} |f_L(\mathbf{x})| \notin [\delta^-, \delta^+] \;&\Rightarrow\; \frac{|f_L(\mathbf{x})|^2}{|G| \cdot Z} \notin \left[\frac{(\delta^-)^2}{Z \cdot |G|}, \frac{(\delta^+)^2}{Z \cdot |G|} \right] \\ &\Rightarrow\; \eta^{\pm} = \frac{(\delta^{\pm})^2}{Z \cdot |G|}. \end{aligned}$$

We now consider the following algorithm:

- sample $\mathbf{x}_0, \ldots, \mathbf{x}_{N-1}$ from X,
- compute $\tilde{p}(\mathbf{x}) = \frac{1}{N} \, |\{\, j : x_j = \mathbf{x} \,\}|$ for every $\mathbf{x}$,

– check if $\tilde{p}(\mathbf{x}) > (\eta^- + \eta^+)/2$ for some $\mathbf{x}$.

It is clear that with careful optimisations, this algorithm runs in time $O(N)$, up to some small polynomial factors. We now need to analyse the success probability of this algorithm. By the Dvoretzky–Kiefer–Wolfowitz inequality [DKW56], there exists a constant A such that for all $\varepsilon > 0$,

$$\Pr\left[\sup_{\mathbf{x}\in G} |p(\mathbf{x}) - \tilde{p}(\mathbf{x})| > \varepsilon\right] \leqslant A \cdot e^{-2N\varepsilon^2}.$$

Here the probability is over the choice of the $\mathbf{x}_0, \ldots, \mathbf{x}_{N-1}$. A result by Massart [Mas90] shows that $A = 2$. For the algorithm to work, we need to estimate $p(\mathbf{x})$ within $\pm(\eta^+ - \eta^-)/2$ so we let $\varepsilon = (\eta^+ - \eta^-)/2$. If want our algorithm to succeed with probability at least $1 - \nu$, we need to choose N such that

$$A \cdot e^{-2N\varepsilon^2} = \nu \qquad \Leftrightarrow \qquad N = \frac{\ln(A) - \ln(\nu)}{2\varepsilon^2}.$$

E 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. To run our code, run `git clone https://github.com/malb/lattice-estimator/` in the directory where `estimates.py` is located.

```
# -*- 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, tanh, coth, 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(coth(pi**2 * Xs.stddev**2))
    ) / 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)

    coeff = 1 / (1 - exp(-1 / 2 / params.Xs.stddev**2))
    tmp_alpha = pi**2 * params.Xs.stddev**2
    tmp_a = exp(8 * tmp_alpha * exp(-2 * tmp_alpha) * tanh(tmp_alpha)).n(30)
    T_guess = coeff * (
        ((2 * tmp_a / 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)
            # if t == 0 then p is irrelevant, so we early abort that loop if that's the case once we hit t==0 twic
            if p[1].y["t"] == 0 and p[0] > 2:
                break
        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)

        coeff = 7 / (
            ZZ(6) * (1 - exp(-1 / (3 * params.Xs.stddev**2))) ** (3 / ZZ(2))
        )
        tmp_alpha = pi**2 * params.Xs.stddev**2
        tmp_a = exp(8 * tmp_alpha * exp(-2 * tmp_alpha) * tanh(tmp_alpha)).n(30)
        T_guess = (
            coeff
            * ((27 * tmp_a**2 / (8 * e)) ** (k_enum / 4))
            * sqrt(2 ** (k_enum * H) * p ** k_fft * 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 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",
    )
```