Quantum LLL with an Application to Mersenne Number Cryptosystems

Marcel Tiepelt[1] and Alan Szepieniec[2]

[1] Karlsruhe Institute of Technology, Karlsruhe, Germany
tiepelt@dev-null.de*
[2] Nervos Foundation, alan@nervos.org*

Abstract. In this work we analyze the impact of translating the well-known LLL algorithm for lattice reduction into the quantum setting. We present the first (to the best of our knowledge) quantum circuit representation of a lattice reduction algorithm in the form of explicit quantum circuits implementing the textbook LLL algorithm. Our analysis identifies a set of challenges arising from constructing reversible lattice reduction as well as solutions to these challenges. We give a detailed resource estimate with the Toffoli gate count and the number of logical qubits as complexity metrics.
As an application of the previous, we attack Mersenne number cryptosystems by Groverizing an attack due to Beunardeau et al. that uses LLL as a subprocedure. While Grover's quantum algorithm promises a quadratic speedup over exhaustive search given access to a oracle that distinguishes solutions from non-solutions, we show that in our case, realizing the oracle comes at the cost of a large number of qubits. When an adversary translates the attack by Beunardeau et al. into the quantum setting, the overhead of the quantum LLL circuit may be as large as 2^{52} qubits for the text-book implementation and 2^{33} for a floating-point variant.

Keywords: LLL · lattice reduction · quantum circuit · Grover's algorithm · Mersenne number cryptosystems

1 Introduction

The famous lattice reduction procedure by Lenstra-Lenstra-Lovász (LLL) [13] was the first algorithm to solve high dimensional lattice problems provably in polynomial time. While its original application was in the factorization of polynomials, has since been applied to great effect in many branches of computer algebra, such as integer programming, Diophantine equations, and cryptanalysis. Recently, the National Institute for Standards and Technology (NIST) has launched a project to standardize post-quantum cryptosystems. Out of 69 round 1 candidates 27 were based on the hardness of lattice problems. In round 2

* Part of this work was conducted at the Computer Security and Industrial Cryptography group (COSIC) at KU Leuven, Leuven, Belgium.

14 of the 26 selected candidates are related to lattice problems. The LLL algorithm's application to cryptanalysis extends beyond attacks on lattice-based cryptosystems to other attacks that require shortest vector oracles, e.g., attacks on Mersenne number cryptosystems. Therefore, when analyzing post-quantum cryptosystems, it is natural to analyze lattice reduction procedures in a quantum context.

The original LLL algorithm was based on rational arithmetic and terminates after polynomial many steps as a function of the initial norms of the lattice basis and the lattice rank, i.e., $O(r^6 \log^3 \tilde{B})$ where r is the rank of the lattice and $\tilde{B}$ bound the maximal initial norm of a vector. Furthermore, it appears to perform better in practice than a rigorous analysis would suggest [17]. One can improve on the theoretical and practical run time by considering floating-point variants that approximate the rational numbers to a certain precision. The first floating-point LLL that would provably find a LLL-reduced basis was due to Schnorr [22] who implemented the coefficients of the intermediate Gram-Schmidt matrix as floating-point numbers with precision $O(r + \log \tilde{B})$. Additionally he introduced a new method to update the matrix in every iteration of the lattice reduction. The L^2 algorithm by Stehlé and Nguyen [16] is the only LLL variant which achieves a complexity quadratic in the bound of the initial norms, namely $O(r^5 \log^2 \tilde{B})$. The quadratic complexity results from approximating the coefficients of the Gram-Schmidt matrix using a precision of $1.6r + o(r)$ and performing a more efficient vector size-reduction. Nevertheless, both floating-point variants of the LLL algorithm do improve on the time spent for arithmetic operations during the lattice reduction, but retain the same upper bound on iterations of the main reduction loop. Furthermore, the underlying operations, such as the vector-size reduction remain equivalent to that of the original LLL algorithm.

Quantum Algorithms. With Shor's algorithm [24, 21] most of the deployed public-key cryptosystems are rendered insecure in the context of large-scale quantum computers. The famous result gave rise to research on *post-quantum cryptosystems*, i.e., cryptography that can withstand quantum attacks. In contrast to Shor's famous algorithm, the equally famous result of Grover does not lead to a complete break of cryptosystems. Grover's algorithm [10] promises a quadratic improvement of exhaustive (key) searches, effectively halving the key length of schemes that aim to achieve the same security level. Grover's algorithm depends on the implementation of a function that captures the problem in question by identifying its solutions and distinguishing them from non-solutions.

Mersenne prime key encapsulation A public key encryption scheme on integer relations modulo a Mersenne number was first introduced by Aggarwal et al. [1] and a later refined thereof was developed and submitted to the NIST post-quantum competition by Aggarwal et al. and independently by Szepieniec [25]. The schemes build upon computation over the ring induced modulo a Mersenne prime using secret integers with low Hamming weight.

Shortly after the introduction of Mersenne number cryptosystems by Aggarwal et al. [1], researchers Beunardeau, Connolly, Graud and Naccache [3] presented an attack with an exponential complexity in the Hamming weight of the sparse secret integers. The main idea of the attack is find the correct lattice representation that hides the sparse secrets as the shortest vector; a lattice reduction algorithm can then extract the secrets. The name reflects the strategy of repeatedly partitioning the sparse integers at random until lattice reduction succeeds in extracting the secret. The attack's complexity arises from the relatively small probability of randomly sampling a good partition.

Our contribution. First, we present a quantum circuit representation of the textbook LLL algorithm. Particularly we explicitly present a range of subcircuits that appear in our representation and discuss how they impact the ancillary qubits. We identify a set of challenges arising from reversible lattice reduction and propose explicit quantum circuits to overcome these. Furthermore, we provide an in-depth complexity analysis of our proposed solution in the quantum setting. We analyze the number of ancillary qubits as well as the number of Toffoli gates needed. We compare the estimates to the memory requirements induced by a floating-point implementation and discuss the memory overhead for the general problem of a reversible lattice reduction. We show that, due to the need for reversibility of the lattice reduction, we need to allocate and maintain a large amount of qubits in every iteration to remember the reduction of the basis vectors. While our circuits are based on the rational textbook LLL we show that our results carry over to floating-point variants implementing the same arithmetic operations. Second, we use this quantum LLL algorithm to complete the analysis of a Groverization of the Slice-and-Dice attack on Mersenne number cryptosystems. As a result, a quantum Slice-and-Dice attack on Mersenne number cryptosystems using the LLL algorithm may have a quantum memory overhead as high as 2^{52} qubits.

Roadmap. Section 2 provides an overview of notions and definitions used for our implementation and analysis. We review the classical LLL algorithm which we used as a starting point for our implementation. Furthermore we give an overview of Mersenne number cryptography and explain the attack by Beunardeau et al. In Section 3 we present our quantum circuits for the LLL algorithm by giving a range of quantum component circuits needed to perform the lattice reduction. We estimate the resources needed to implement those constructions and suggest potential improvements. Section 4 shows how to combine Grover's algorithm with the attack by Beunardeau et al. using our quantum LLL circuit as a subroutine. We calculate the needed quantum resources when instantiating the crytosystems to achieve a 128-bit security level against quantum attack, as per the parameters suggested by Aggarwal et al. [1] and by Szepieniec [25].

2 Preliminaries

LLL algorithm. The LLL algorithm is a well-known lattice reduction algorithm with many applications in cryptanalysis. Given a modular equation $y \equiv \sum_i \alpha_i x_i \bmod p$ with $\prod_i x_i < p$ the LLL algorithm is able to determine the unique solution in polynomial time [12]. The solution is determined by constructing a lattice of rank r represented by the canonical basis $(b_1, \ldots, b_r)$. On input of the basis matrix, the lattice reduction procedure computes a reduced basis $\tilde{B}$ for a fixed approximation factor δ such that the Gram-Schmidt orthogonalized basis $\tilde{B}$ satisfies $||\tilde{b}_i^*||^2(\delta - \frac{1}{4}) \leq ||\tilde{b}_{i+1}^*||^2$ for all $1 \leq i \leq r$ [12, ch. 10]. In general the length of the shortest vector can be approximated with the Minkowski theorem as the first minimum of the lattice: $\lambda_1(\mathcal{L}) \leq \sqrt{\gamma_r} \det(\mathcal{L})^{\frac{1}{r}}$ where γ_r is Hermite's constant. Depending on the approximation factor δ and the difference in length between the first and the second minima of the lattice the shortest vector might not be found.

The algorithm first computes an orthogonal basis using the Gram-Schmidt procedure followed by an iterative approach examining each two subsequent vectors of the basis. Each iteration considers the plane spanned by the two vectors $b_i, \tilde{b}_{i+1}$ and attempts to reduce the vector $\tilde{b}_{i+1}$ by an integer multiple of the vector $\tilde{b}_i$. Then, if the two vectors fulfill the Lovász condition, $\delta||\tilde{b}_i^*||^2 \leq ||\tilde{b}_{i+1}^*||^2 + {}^{(\tilde{b}_{i+1}|\tilde{b}_i^*)}/_{||\tilde{b}_i^*||^2}$, the vector $\tilde{b}_{i+1}$ is size-reduced and projected onto the hyperplane spanned by all lesser vectors. If the ordering is not fulfilled, the vectors are swapped such that the vector b_{i+1} is compared to the lesser vectors in the following iterations. Intuitively the Lovász condition ensures that the vectors are approximately orthogonal and allows to reorder them if this is not the case. The LLL algorithm terminates as soon as each subsequent pair of basis vectors fulfills the Lovász condition and if they are ordered by length with respect to the approximation factor.

The pseudo-code in Algorithm 2.1 reflects the classical approach using *exact* values, implemented with rational numbers to compute the coefficients of the Gram-Schmidt matrix M.

Algorithm 2.1: LLL Algorithm

```
Input: Basis $B = (b_1, b_2, ..., b_r)$
Output: Reduced Basis $B^*$
$B^*, M \leftarrow$ Gram-Schmidt orthogonalization(B)
$L_i \leftarrow$ Compute length of vectors $b_i$ for $i \in \{1, 2, ..., r\}$
$k \leftarrow 2$
while $k \leq r$ do
    Reduce $b_k$ by $\lfloor m_{k,k-1} \rceil b_{k-1}$ and update $M$ s.t. $B = MB^*$
    if Lovász condition holds on $b_k$ and $b_{k-1}$ then
        for $j = k - 2$ to 0 do
            Reduce $b_k$ by $\lfloor m_{k,j} \rceil b_j$ and update $M$ s.t. $B = MB^*$
        $k+ = 1$
    else
        Swap $b_k$ and $b_{k-1}$
        Update length $L_i, L_{i-1}$ and Gram-Schmidt matrix $M$ to reflect swap
        $k \leftarrow max(2, k - 1)$
```

The complexity of the LLL algorithm is dominated by the main loop and further depends on the size of the numbers that being processed. The number

of iterations of the main loop can be bounded by defining the potential of the lattice: $D = \prod_{i=0}^{r} ||b_i^*||^{2i}$ as shown by Nguyen and Valle [Thm 10][18]. Whenever the Lovàsz condition is not fulfilled two vectors are swapped and D decreases by a factor of δ. Since D is an integer larger than 0, the number of swaps is bounded by the logarithm of D, bound by $\tilde{B}^{2d}$, where $\tilde{B}$ is the maximum initial norm of the input vectors.
Withing the main loop the most expensive operation is to update the Gram-Schmidt matrix of size rd. Furthermore the rationals in the naive LLL algorithm have bit length at most $O(r \log \tilde{B})$. The operations within LLL on these numbers can be computed in at most square time. This results in an overall complexity of $O(r^5 d \log^3 \tilde{B})$ [18].

Schnorr [22] introduced a more efficient variant using floating-point approximations with precision $O(r + \log \tilde{B})$ for the coefficients of the Gram-Schmidt matrix. The approximation of the coefficients significantly reduces the computational effort of each loop iteration, resulting in a complexity of $O(r^4 \log \tilde{B}(r + \log \tilde{B})^2)$ operations.

The result was later improved by Nguyen and Stehlé [16] introducing the L^2 algorithm, the first provable lattice reduction algorithm with complexity quadratic in the norm of the input vectors. The L^2 algorithm adopts the approach to approximate the Gram-Schmidt coefficients using floating-point numbers and combines it with a provable floating-point procedure mimicking the Gram-Schmidt orthogonalization process. The algorithm is the best known result for a lattice reduction with a complexity of $O(r^5(r + \log \tilde{B}) \log \tilde{B})$.

Reversible floating-point arithmetic. The use of floating-point representations in classical computing usually implies the loss of information due to rounding errors. Translating this inherently non-reversible process into an unitary operations requires preserving additional information, *e.g.*, the inputs of the initiating arithmetic operation. Bennett [11] showed how to translate any ordinary Turing machine running in time T and space S into a reversible Turing machine running in time $O(T^{1+\epsilon})$ and space $O(S \log T)$, making it clear that such an operation can be implemented on a quantum computer. While this gives a constructive approach for a reversible Turing machine it may not be clear how to translate this into an circuit. The first reversible floating-point adder meeting the requirements of industrial standards was introduced by Nachtigal et al. [15, 14]. While the quantum gate-cost of their circuit was linear in the number of (qu)bits, the output also includes $O(n)$ garbage qubits. Nguyen and Meter [6] presented improved floating-point arithmetic circuits with only a constant number of garbage qubits. However, *both representations must preserve one of the inputs to allow reversibility.*
There is no widely adopted procedure to implement floating-point numbers in a quantum circuit. The classical approach with two registers containing mantissa and exponent seems to be a feasible scheme. However, one would need an extra register to keep track of a "remainder" which is discarded in classical computing. Another approach was introduced by Wiebe et al. [26] by implementing small rotations onto single qubits which allows to mimic and compute on mantissa and

exponent. Due to the lack of standardized or widely adopted representation of floating-point arithmetic in the quantum setup we do not optimize nor establish any arithmetic circuits for floating-point operations.

Grover's algorithm. We assume the reader to be familiar with the basic notions of quantum computation (a standard text is Nielsen & Chuang [19]). Grover's algorithm improves the search of an unordered data set with a quantum computer gaining a quadratic improvement [10] over an unstructured classical search. The algorithm employs a procedure that starts out with a superposition of all elements in the data set, such that each element is equally likely to be observed. A successive application of the *Grover iteration* improves the success probability of observing a target element as described below.

Let there be an unordered set of cardinality N containing M target elements. The goal is to extract any one of the targets. Consider $\mathcal{H}$ to be a Hilbert space which describes the state space of a quantum system. Furthermore assume the access to a black-box operator $\mathcal{U}_{\mathcal{O}} : \mathcal{H} \to \mathcal{H}$ which computes $|x, c\rangle \mapsto |x, c \oplus f(x)\rangle$ and implements the function f mapping target elements to 1 and non-target elements to 0. The operator therefore splits the state space into a direct sum of subspaces such that the *good* subspace for $f(x) = 1$ represents the target elements and the *bad* subspace for $f(x) = 0$ represents the non-target elements. Let ρ denote the initial success probability of observing a target element. Then Grover's algorithm requires $\rho^{-\frac{1}{2}}$ iterations until a target element is observed with high probability.

Mersenne number key encapsulation. Mersenne number cryptosystems compute over the integer ring $\mathbb{Z}/p\mathbb{Z}$ where $p = 2^n - 1$ is a Mersenne number such that all integers have bit length at most n. The encapsulation schemes follow the idea of establishing a shared noisy one-time pad using a noisy Diffie-Hellman protocol. The later is embedded into a framework featuring de-randomization and re-encryption to achieve CCA security in the (quantum) random oracle model. Two sparse integers $a, b \in \mathbb{Z}/p\mathbb{Z}$ are chosen uniformly random with Hamming weight ω during the key generation. Together with an uniformly random $G \in \mathbb{Z}/p\mathbb{Z}$ and $H = aG + b \mod p$ they form the secret and the public key: $sk := (a, b), pk := (G, H)$. The schemes suggested by Aggarwal et al. [1] and by Szepieniec [25] are based on the *Mersenne low Hamming combination search problem (LHCS)*: given a tuple $(G, aG + b \mod p)$ with parameters as above, find the integers a and b. The problem is believed to be hard for classical as well as quantum computers.

Slice-and-Dice attack. Beunardeau et al. [3] presented an attack on the LHCS Problem exploiting the uniqueness of the sparse solutions to the equation $H = aG + b \mod p$. The attack revolves around the idea of partitioning the binary expansion of unknown integers into intervals. The intervals are used to construct a lattice which is then reduced to find the shortest vectors. If the intervals are chosen "correctly", the constructed lattice hides the secrets a, b as shortest vectors: Consider the segmentation of a, b into partitions P_a, P_b. Since the two integers are sparse there are relatively few ones, such that there exists a partition where each block represents a "small" number. Such a partitioning

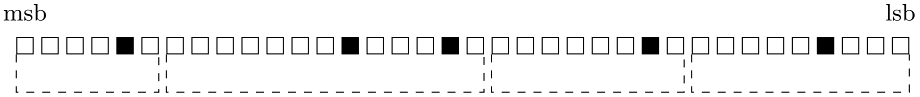


Fig. 1: Partition of a sparse integer into intervals. The blocks represent the binary expansion, the black blocks as ones and the white blocks as zeros. Interpreting an interval as integer yields a small number, e.g., the first interval represents $2^3 = 8$.

is depicted in Figure 1. Each partition is uniquely defined by a set of starting positions $p_{a,i}, p_{b,j}$ inducing the partitioning into blocks.

The intention of the attack is to sample these starting positions and construct a lattice accordingly as in Equation (1) such that there exists an exponential gap in terms of vectors' length between those representing the secrets and any other vectors of the lattice. De Boer et al. [4] show that this occurs if and only if all the ones of the secrets fall into a certain range of each interval.

$$\mathcal{L}_{a,b,H} = \left\{ (x_1, ..., x_k, y_1, ...y_l, z) \middle| \sum_{i}^{k} 2^{p_{a,i}} x_i G + \sum_{j}^{l} 2^{p_{b,j}} y_j \equiv H \mod p \right\} \tag{1}$$

For correctly sampled starting positions the shortest vector represents the secrets and can be extracted using a lattice reduction technique, *e.g.*, the LLL algorithm. In order to apply the LLL algorithm one needs the canonical representation of the lattice basis as matrix from the partitions P_a and P_b. Consider the case where the integers are partitioned into $k + l$ parts and let $\boldsymbol{\mathcal{I}}$ denote the $(k + l) \times (k + l)$ sized identity matrix. Then the matrix in Equation (2) as a set of row vectors form a basis for the lattice $\mathcal{L}_{a,b,H}$. Each row vector represents an interval and is constructed to fulfill Equation (1). After performing the lattice reduction with sampled partitions one needs to check the Hamming weight of the reduced basis vectors. If the Hamming weight of the shortest vectors equals the Hamming weight of the secrets, the attack was successful [3, Sec. 2].

Algorithm 2.2: Slice-and-Dice

```
Input: Basis $H, G, p, \omega$
Output: $a, b$
while True do
    $P_a \leftarrow$ sample $\omega$ random intervals covering the range $(0, p)$
    $P_b \leftarrow$ sample $\omega$ random intervals covering the range $(0, p)$
    $B \leftarrow$ construct $\mathcal{L}_{a,b,H}(H, G, p, P_a, P_b)$
    $B^* \leftarrow LLL(B)$
    if $\exists b^* \in B^*$ s.t. $Ham(bin(b^*)) = 2 \cdot \omega$ then
        Return $b^*$
```

$$\left(\begin{array}{c|c} \mathcal{I} & \begin{array}{ll} 0 & -2^{p_{a,1}} G H^{-1} \mod p \\ \vdots & \qquad\vdots \\ 0 & -2^{p_{b,l}} H^{-1} \mod p \end{array} \\ \hline \begin{array}{c} 0 \dots 0 \\ 0 \dots 0 \end{array} & \begin{array}{ll} 1 & \qquad -H \\ 0 & \qquad p \end{array} \end{array}\right) \tag{2}$$

Algorithm 3.1: Conditioned Loop

```
$k \leftarrow 0$
while $k \leq r$ do
    ApplyTask(k)
    if $SomeCondition(k)$ then
        $k \leftarrow k + 1$
    else
        $k \leftarrow k - 1$
```

3 Quantum Lattice Reduction Algorithm

The textbook description of the LLL algorithm by Joux [12, ch. 10] using rational numbers serves as a starting point for our implementation. We chose this variant over the more efficient approximate floating-point techniques due to the natural correspondence of implementing (quantum) arithmetic with rational numbers, in particular, the impossibility to *forget* rounding errors in the quantum setting: floating-point operations as in the classical sense require forgetting information, whereas fractional arithmetic does not. However, the results from our implementation of a conditioned loop and the uncomputation of the Gram-Schmidt matrix carry over to the floating-point variants. We use the following list of quantum registers throughout our implementation:

$|B\rangle$ The basis matrix spanning the lattice.
$|B^*\rangle$ The Gram-Schmidt orthogonalized basis.
$|L\rangle$ The lengths of the vectors in B.
$|M^{(k)}\rangle$ The Gram-Schmidt matrix M for each iteration k.
$|\mathcal{L}\rangle$ A single qubit containing the result of checking the Lovász condition
$|K\rangle$ The counter of the main loop.
$|ctl\rangle$ Generic control qubits.

Furthermore, we make use of the following subcircuits implementing basic arithmetic operations:

Addition $\mathcal{A} : |x, y, 0\rangle \mapsto |y, y, x + y\rangle$
Multiplication $\mathcal{M} : |x, y, 0\rangle \mapsto |x, y, x + y\rangle$
Division $\mathcal{D} : |x, y, 0\rangle \mapsto |r, y, \lfloor {}^{x}\!/_{y} \rfloor\rangle$, where r is the remainder of x/y

3.1 Quantum designs.

Conditional loops. Conditioned *while*-loops are widely used in classical computing to execute a certain task based on the current state of a variable, which may or may not be modified within the loop. Consider the loop in Algorithm 3.1 that runs until the variable k reaches the value of r. The loop may or may not run for an infinite time depending on the outcome of the condition.

When translating this structure to a quantum circuit, k may be in superposition, say $|K\rangle$, representing multiple different values at the same time. The conditioned loop may require a different number of iterations of a task for each value represented by $|K\rangle$. However, the algorithm in Figure 3.1 applies the loop to the register k in every iteration. Our quantum implementation in Circuit 2 controls the application of the loop task based on a lower and upper control qubit, checking if the counter $|K\rangle$ is within its valid limitation. In order to assure that the loop is applied often enough, one needs to find an upper bound on the number of iterations, and hence on the counter $|K\rangle$. The loop then consists of bound(K) many cycles where the task is applied if the counter is within its valid limits.

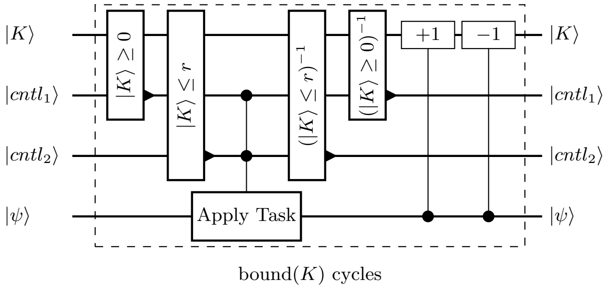


Fig. 2: Conditioned quantum loop with lower and upper limit control qubits.

Therefore, our implementation of the quantum *while*-loop is a *for*-loop with some reasonable termination bound. The important takeaway here is that conditioned loops always have worst-case run time. In the particular case of the LLL algorithm the Line 6 of Algorithm 2.1 is such a construction.

Uncomputation. The copy-uncompute trick, first introduced by Bennett et al. [2], is a well known procedure to reset quantum memory to a known state. The purpose of resetting garbage qubits is to avoid unwanted interference when using subroutines in a larger circuit. Additionally, uncomputed qubits may be reused as ancillary qubits for following operations. The trick allows to design reversible operations where the number of "used" output qubits matches the number of "used" input qubits. For example, consider the following addition procedure where the register y is reset to a known state $|0\rangle$ such that $|y|$ many qubits can be reused after the operation:

$$|x, y, 0^{max(|x|,|y|)+1|}\rangle \mapsto |x, y, x+y\rangle \mapsto |x, 0^{|y|}, x+y\rangle \tag{3}$$

However, not all operations within the LLL algorithm can be implemented in a way, that allows to reuse all input qubits in such generic way. Consider a function that computes a vector-size reduction: a vector $b_i \in B$ is length reduced by an

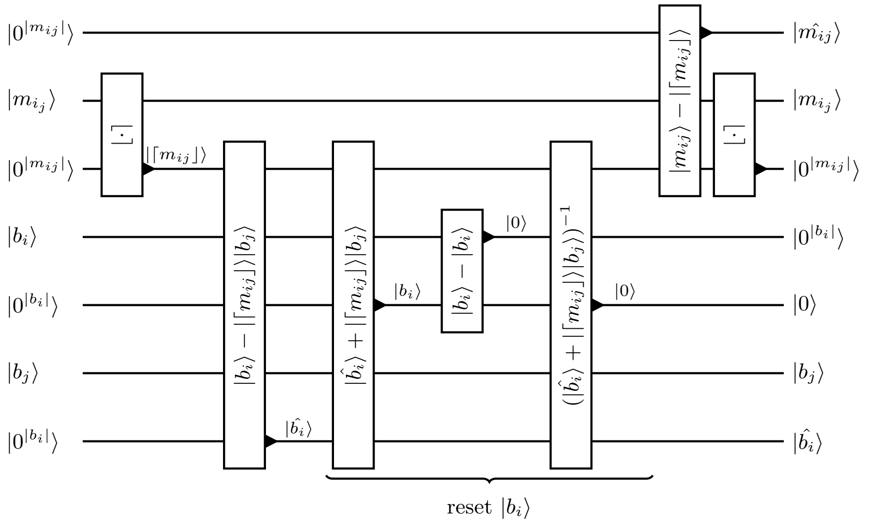


Fig. 3: Vector size-reduction and uncomputation of original vector b_i.

integer multiple of some vector b_j by a factor of $\lceil m_{ij}\rfloor$. On a basic level the following operations are performed:

$$\lceil m_{ij}\rfloor \leftarrow \mathrm{round}(m_{ij}) \tag{4}$$

$$\hat{b_i} \leftarrow b_i - \lceil m_{ij}\rfloor b_j \tag{5}$$

$$\hat{m_{ij}} \leftarrow m_{ij} - \lceil m_{ij}\rfloor \tag{6}$$

In the classical case, the computation of the new value $\hat{m_{ij}}$ overriding the value m_{ij} results in the loss of the information required to recompute b_i from the size-reduced vector $\hat{b_i}$. In order to be able to reverse the size-reduction it is necessary to maintain addition information, mainly the old value m_{ij}. Translating this procedure into the quantum context results in Circuit 3: the size-reduced basis vector $|\hat{b_i}\rangle$ is computed from the (rounded) coefficient $|m_{ij}\rangle$ and some vector $|b_j\rangle$. Then the new Gram-Schmidt coefficient $|\hat{m_{ij}}\rangle$ is computed from $|m_{ij}\rangle$ and its rounded value. With the quantum registers $|m_{ij}\rangle$ and $|\hat{b_i}\rangle$ the value of the vector $|b_i\rangle$ can be first recomputed. This can then be used to reset the original quantum register $|b_i\rangle$ into a zero state, and then the value of $|b_i\rangle$ can be uncomputed. At last either the quantum register containing the rounded value $|\lceil m_{ij}\rfloor\rangle$ or the quantum register $|m_{ij}\rangle$ can be uncomputed. However, neither of them can be uncomputed with out the information contained in the other. Therefore, either $|m_{ij}\rangle$ or $|\lceil m_{ij}\rfloor\rangle$ have to be preserved when using reversible arithmetic operations.

Figure 4 shows a dependency graph visualizing the operation in question. The directed edges depict the flow of information, whereas the rounded nodes represent quantum registers and the squared nodes represent the functions. In order to uncompute an input value of a function, one needs to preserve the output as well as the other inputs. Since the node $\lceil m_{ij}\rfloor$ is both input and

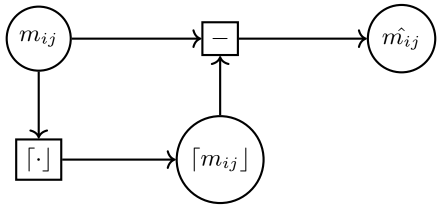


Fig. 4: Information flow when updating Gram-Schmidt matrix coefficients.

Algorithm 3.2: GSO Algorithm [12, Alg 10.3]

```
Input: Basis $B = (b_1, b_2, ..., b_r)$
Output: Orthogonal basis $B^*$ and transformation $M$
for $i \leftarrow 1$ to $r$ do
    $b_i^* \leftarrow b_i$
    for $j \leftarrow 1$ to $i - 1$ do
        $m_{i,j} \leftarrow \frac{(b_i|b_j^*)}{||b_j^*||^2}$
        $b_i^* \leftarrow b_i^* - m_{i,j} b_j^*$
```

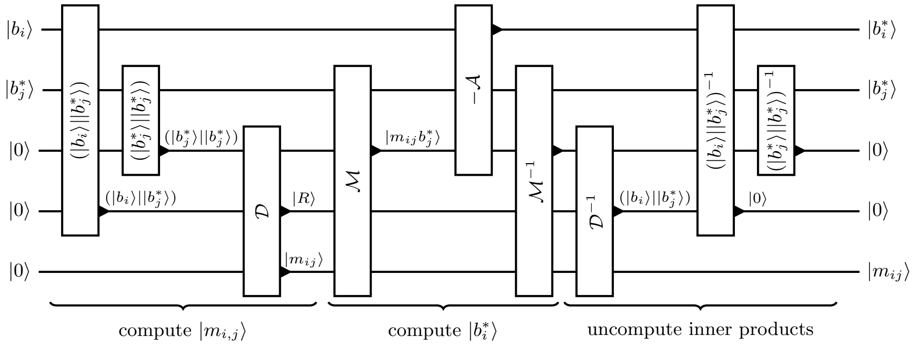


Fig. 5: Single iteration of the quantum Gram-Schmidt orthogonalization inner loop.

output of a function it cannot be uncomputed while serving as information for uncomputation.

Overall, the size-reduction operator requires preserving the initial coefficient of the Gram-Schmidt matrix for reversibility. In the quantum case, this operation is applied in every iteration of the main loop in Line 6 of Algorithm 2.1.

3.2 Miscellaneous Circuits

Quantum Gram-Schmidt orthogonalization. The circuit implementing a single iteration of the quantum Gram-Schmidt orthogonalization (QGSO) in Figure 5 is based on the Pseudo-code 3.2 of the classical procedure by Joux [12].

Rounding to the nearest integer. The vector length reduction $b_i \leftarrow b_i - \lceil m_{ij} \rfloor b_j$ appearing in the LLL algorithm requires rounding the rational number $m_{ij} = {}^{m_n}/_{m_d}$ to the nearest integer. The rounding to the closest integer in positive direction is based on the decimal part. Therefore, we need to compute

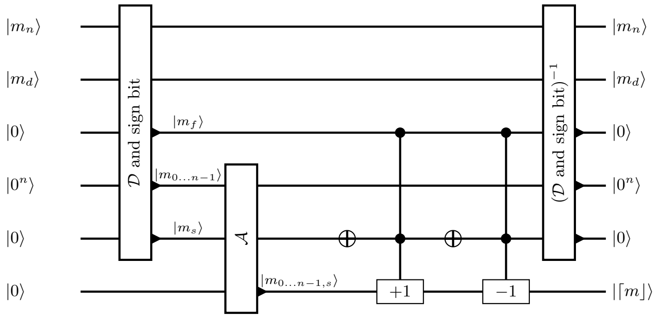


Fig. 6: Rounding to the nearest integer in positive direction.

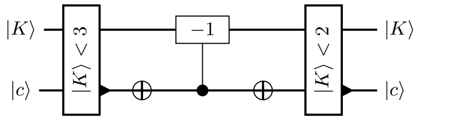


Fig. 7: Circuit implementing the operation $K := max(2, K - 1)$

the division of ${}^{m_n}/_{m_d}$ with some precision. In our example we choose a precision of 1 (qu)bit. The integer part is stored in register $|m_{0...n-1}\rangle$, the decimal part in register $|m.f\rangle$. The sign (qu)bit of the rational number is stored in the register $|m_s\rangle$. The rounded value equals the integer part plus or minus 1, depending on the decimal part and the sign (qu)bit as below:

$$\begin{aligned} m_s = 1 \wedge m.f = 1 &\Leftrightarrow \lceil m \rfloor = \lfloor m \rfloor - 1 \\ m_s = 0 \wedge m.f = 1 &\Leftrightarrow \lceil m \rfloor = \lfloor m \rfloor + 1 \\ m.f = 0 &\Leftrightarrow \lceil m \rfloor = \lfloor m \rfloor \end{aligned}$$

We implement our vector size-reduction using the rounding operator in Figure 6. The gate cost of the rounding circuit is $2\mathcal{D}_T + 3\mathcal{A}_T$ and referred to as $\mathcal{R}_T$.

max(2, k-1). Circuit 7 implements the $|K\rangle = max(2, |K\rangle - 1)$ operator that is required to conditionally decrement the counter in the second branch of the main loop.

LLL. Circuit 8 shows the LLL algorithm with the respective conditioned loops and the controls of the branching. In each cycle we cache coefficients of the Gram-Schmidt matrix in a new register $|M\rangle$ to preserve reversibility.

3.3 Resource estimate

Gate count. Toffoli gates are universal for quantum computing and can be constructed from the *Clifford+T* group, whereas the cost of such a gate depends mostly on the number of T-gates. Selinger [23] gives a construction of a Toffoli gate with T-depth of 1. Therefore, the Toffoli gate count is closely related to the full cost of a quantum circuit. In addition, Fowler, Mariantoni and Cleland [7] show that the number of logical qubits scales with the number of Toffoli gates. Consequently, we use the Toffoli gate count as a complexity metric in this work.

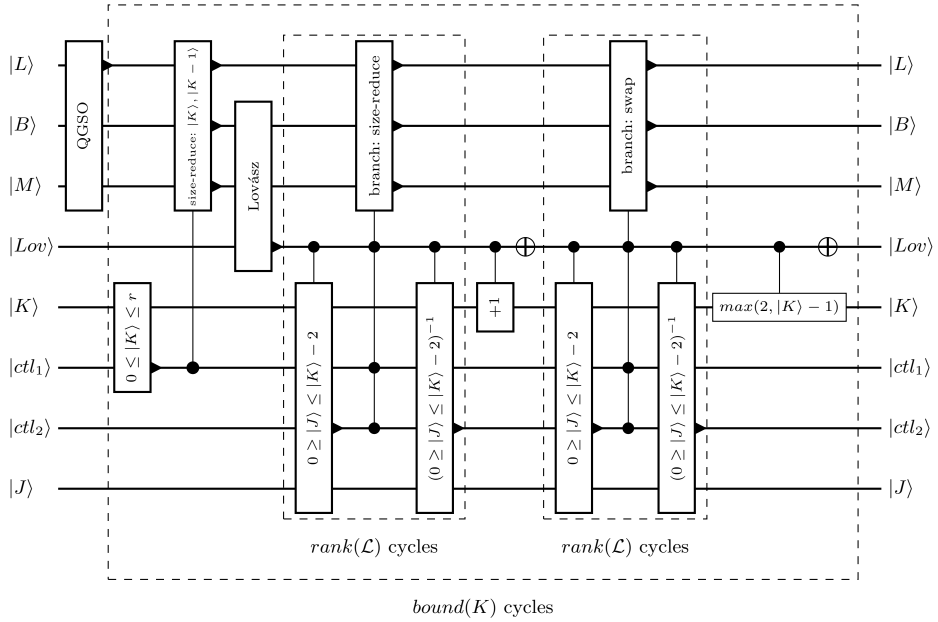


Fig. 8: Quantum circuit implementing the LLL algorithm where the *branch: size reduce* represents the size reduction if the Lovász holds and the *branch: swap* the respective other branch. Note that each cycle of the main loop requires additional qubits for the Gram-Schmidt matrix M.

Let m be the (qu)bit-length of the numbers processed in the LLL algorithm. We use the Toffoli gate count of the following implementations of quantum circuits in our analysis:

- Curracaro [5] presented a quantum adder using $2m + O(1)$ Toffoli gates referred to as $\mathcal{A}_T$. Note that a subtraction can be implemented using the same gates.
- Gidney [9] achieves a multiplication circuit using windowing and table lookups in $O({}^{m^2}\!/_{\log n})$ Toffoli gates as $\mathcal{M}_T$.
- Rines and Chuang [20] propose a division circuit comprising $4m^2$ Toffoli gates as $\mathcal{D}_T$.

Let $r := rank(\mathcal{L})$ be the rank of the lattice such that there are r basis vectors: $(b_1, \ldots, b_r)$ with $b_i \in \mathbb{R}^d$. The frequently used inner product $(b_i|b_j)$ or $||b_j||^2$ can be constructed from $d \cdot \mathcal{M}_T \cdot \mathcal{A}_T$ Toffoli gates. We will refer to this count as $\mathcal{I}_T$. Note that the Toffoli gate depth is significantly lower when applying vector multiplication in parallel and enabling a tree like addition procedure.

QGSO. The quantum GSO consists of an outer loop iterating the r basis vectors and an inner loop iterating the respective lesser vectors. Each iteration consists of the computation of a coefficient of the Gram-Schmidt matrix by computing two inner products and a division, which intermediate result has to be uncomputed with total cost: $2(2\mathcal{I}_T + \mathcal{D}_T)$. Furthermore the vector $|b_i^*\rangle$ is computed with a scalar multiplication and a vector subtraction. The cost in Toffoli gates with uncomputation is: $2d(\mathcal{M}_T + \mathcal{A}_T)$. Overall the quantum GSO requires $\frac{r^2-r}{2}\left(4\mathcal{I}_T + 2\mathcal{D}_T + 2d(\mathcal{M}_T + \mathcal{A}_T)\right)$ Toffoli gates.

Main Loop. The gate count of the main loop of the textbook LLL implementation is closely related to the worst-case analysis of the classical case.

The computation of the first branch consists of the computation of a length reduction for r vectors. Each reduction requires a rounding operator, the actual size-reduction and the computation of the new Gram-Schmidt coefficient $|m_{ij}\rangle$ with cost: $2\mathcal{R}_\mathcal{I} + 2d(\mathcal{M}_T + \mathcal{A}_T) + \mathcal{A}_T$. The remaining coefficients in the same row of the matrix are updated using $r(\mathcal{M}_T + \mathcal{A}_T)$ Toffoli gates. The first branch is dominated by the terms $(2rd + r^2)(\mathcal{M}_T + \mathcal{A}_T)$.

The second branch consists of swapping the vectors and updating the Gram-Schmidt matrix accordingly with a Toffoli cost of $r(2(\mathcal{M}_T + \mathcal{A}_T))$. Uncomputation cost is negligible due to caching the matrix in every iteration. Both branches have to be applied subsequently.

The computation of the Lovász qubit and the initial length reduction is negligible and therefore omitted from the calculation. The number of cycles of the main loop can be bound using by the classical worst-case with $\text{bound}(K) := r^2 \log \tilde{B}$, e.g. by Nguyen [17], where $\tilde{B}$ bounds the norms of the input basis. Expressing the gate count with regards to the (qu)bit length m of the arithmetic operations the Toffoli gate count is in the order of:

$$O\left(2\log\tilde{B}\left(r^3 d + r^4\right)\left(\frac{m^2}{\log m} + 2m\right)\right) \tag{7}$$

Logical qubits. The number of logical qubits is derived from representing the basis matrix $|B\rangle$ and the Gram-Schmidt matrix $|M\rangle$ with rd coefficients where each one has qubit length $m = r\log\tilde{B}$ for the textbook implementation. One requires to cache $r^2 \log\tilde{B}$ such matrices. Additionally our implementation requires $r^2\log\tilde{B}$ for the counter $|K\rangle$, as well as $max(r,d)\cdot m$ ancillary qubits for arithmetic operations. The total number of qubits for a reversible LLL algorithm is thus:

$$r^4 d \log^2\tilde{B} + r^2\log\tilde{B} + max(d,r)\cdot m \tag{8}$$

Physical qubits. We note that the number of physical qubits to implement a fault tolerant circuit using an error correcting architecture is significantly higher.

3.4 Improvements

Intermediate uncomputation. We propose a time/space trade-off to improve on the number of logical qubits required. During the course of the lattice reduction we suggest to independently run the (quantum) LLL circuit on intermediate lattice basis' and to recompute the Gram-Schmidt matrices. This allows to reset the cached matrices to a known value, i.e., a quantum register with a zero vector, and to reuse these qubits in the following operations.

The vector size-reduction takes as input the matrices $B^{(i)}$ and $M^{(i)}$ and outputs the reduced basis $B^{(i+1)}$ with updated Gram-Schmidt matrix $M^{(i+1)}$. With

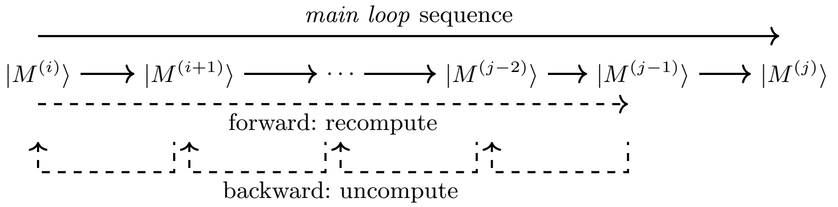

main loop sequence
$|M^{(i)}\rangle \longrightarrow |M^{(i+1)}\rangle \longrightarrow \cdots \longrightarrow |M^{(j-2)}\rangle \longrightarrow |M^{(j-1)}\rangle \longrightarrow |M^{(j)}\rangle$
forward: recompute
backward: uncompute

Fig. 9: Sequential uncomputation of Gram-Schmidt matrices

every such operation new memory to cache the matrix $M^{(i+1)}$ is required, while $B^{(i)}$ can be reset to a known state. Let $|M|$ be the number of qubits required to store the Gram-Schmidt matrix ($|B|$ respectively for the basis matrix). Then the following sequence is computed over the course of the main loop:

$$\begin{aligned}
|B^{(i)}\rangle|M^{(i)}\rangle|0^{|B|}\rangle|0^{|M|}\rangle &\xmapsto{size-reduce} |0^{|B|}\rangle|M^{(i)}\rangle|B^{(i+1)}\rangle|M^{(i+1)}\rangle \\
|B^{(i+1)}\rangle|M^{(i+1)}\rangle|0^{|B|}\rangle|0^{|M|}\rangle &\xmapsto{size-reduce} |0^{|B|}\rangle|M^{(i+1)}\rangle|B^{(i+2)}\rangle|M^{(i+2)}\rangle \\
&\dots \\
|B^{(j-2)}\rangle|M^{(j-2)}\rangle|0^{|B|}\rangle|0^{|M|}\rangle &\xmapsto{size-reduce} |0^{|B|}\rangle|M^{(j-2)}\rangle|B^{(j-1)}\rangle|M^{(j-1)}\rangle \\
|B^{(j-1)}\rangle|M^{(j-1)}\rangle|0^{|B|}\rangle|0^{|M|}\rangle &\xmapsto{size-reduce} |0^{|B|}\rangle|M^{(j-1)}\rangle|B^{(j)}\rangle|M^{(j)}\rangle
\end{aligned}$$

Given the registers $|M^{(j-1)}\rangle|B^{(j)}\rangle|M^{(j)}\rangle$ it is impossible to reconstruct the matrices $|B^{(i)}\rangle|M^{(i)}\rangle$. However, the sequence was computed from the initial basis $|B^{(i)}\rangle$ and the Gram-Schmidt matrix $|M^{(i)}\rangle$, hence one can use the initial state to uncompute the sequence:

Forward steps Recompute $|M^{(i+1)}\rangle, |M^{(i+2)}\rangle, \dots, |M^{(j-2)}\rangle, |M^{(j-1)}\rangle$.

Backwards steps Each Gram-Schmidt matrix can be used to uncompute its successor, e.g., $|M^{(j-2)}\rangle$ is used to uncompute $|M^{(j-1)}\rangle$.

By following this procedure, one can free all the quantum memory but the initial register $|M^{(i)}\rangle$ and the last cached state $|M^{(j)}\rangle$. The number of cached qubits in the sequence can been reduced from $(j+1)\cdot \mathit{sizeOf}(M)$ to $2\cdot \mathit{sizeOf}(M)$ as in Figure 9. The trade-off allows to reset $j \cdot \mathit{sizeOf}(M)$ many qubits using j iterations of the main loop and $j \cdot \mathit{sizeOf}(M)$ *temporary* ancillary qubits. Let bound(K) be the number of iterations of the main loop in our implementation and hence the number of cached Gram-Schmidt matrices. The trade-off is optimal for $j = (\text{bound}(K))^{\frac{1}{2}}$. This is equivalent to performing an uncomputation sequence every $(\text{bound}(K))^{\frac{1}{2}}$ iterations of the main loop.
By applying this improvement the number of cached Gram-Schmidt matrices can be reduced by a square root to $r(\log \tilde{B})^{\frac{1}{2}}$ in our textbook implementation. The largest term from Equation (8) on the number of logical qubits is now reduced to $r^3 d \log \tilde{B}(\log \tilde{B})^{\frac{1}{2}}$. The enhancement carries over to the naive floating point variant and the loop implementing the L^2 algorithm.

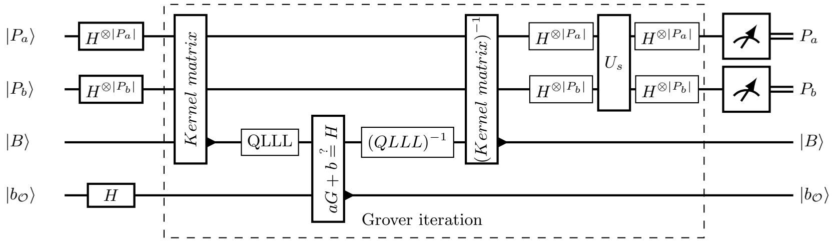
Floating-point LLL. Our results regarding conditioned loops and the reversibility of reducing vectors by an integer multiple carry over to naive floating-point variants. In particular, the LLL algorithm shares certain computational methods with its provable floating point variants, such as the vector size-reduction by an integer multiple or the conditioned loop. The significant increase in efficiency is based on the the representation and processing of the Gram-Schmidt coefficients. Applying our analysis on Schnorr's [22] variant results in the use of $\geq r^2 d \log \tilde{B} (\log \tilde{B})^{\frac{1}{2}}$ qubits. The L^2 algorithm due to Nguyen and Stehlé [16, Fig. 4] implements an equivalent vector length-reduction. It follows that the L^2 algorithm [16, Theorem 2] with a precision of $l = 1.6d + o(d)$ would require $\geq r^2 d \log \tilde{B}(1.6d + o(d))$ many qubits to preserve reversibility in the quantum setup. However, we did not analyze the L^2 algorithm in detail and do not claim that this is optimal.

Execution parameters. The (classical) worst-case of the number of iterations can be improved by adjusting the parameter δ resulting in a loss of "quality" of the reduced basis with regards to the length of the reduced vectors. Nguyen and Stehlé [17] evaluate the running time of LLL for different domains, such as random knapsack problems, and suggest that the complexity is lower on average. We leave an analysis of specific problem instances as a future line of work. However, we do give an heuristic value for the Slice-and-Dice attack in the next section.

4 Quantum Slice-and-Dice

The combination of the Slice-and-Dice attack with Grover's algorithm utilizes our quantum representation of the LLL algorithm from the previous section as a subroutine. Given the public-key of a Mersenne number cryptosystem one starts out with a superposition of all possible partitions which in turn represent all possible starting positions of intervals. The lattice reduction subroutine distinguishes those partitions that allow extraction of the secret sparse vectors using a lattice reduction, and those that do not allow said extraction. First, the subroutine computes the basis matrix, which is the direct translation of the classical case. Each partition is now represented by the superpositions $|P_a\rangle, |P_b\rangle$. The lattice reduction is performed and the "correct" partitions are identified. This procedure can be implemented using canonical modular operations to recompute the public key and comparing to the input. Afterwards the process has to be inverted to uncompute all operations. Figure 10 shows Grover's algorithm with our subroutine.

Identification of the shortest vector. The attack is successful if and only if the Grover oracle can successfully distinguish the sparse secret integers in the reduced basis from other short vectors. Then the oracle qubit has to be flipped according to the secret integers and respectively not-flipped by other short vectors. In the classical case, Beunardeau et al. identify the solutions by computing the sum of the Hamming weights of the resulting basis vectors. This is not sufficient in the quantum setting due to the existence of other vectors with



The U_S operator flips all states except 0

Fig. 10: Groverized Slice-and-Dice with a LLL oracle.

low Hamming weight.
De Boer et al. [4] showed that the lattice contains vectors of the form $(0,\ldots,0,2^m,-1,0,\ldots,0)$ of length $(4^{|P_i|}+1)^{\frac{1}{2}}$. Moreover, the authors bound the minimal size of the partition based on an approximation of the first minimum by Gama and Nguyen [8]: $\lambda_1(\mathcal{L}) \approx (n/2\pi e)^{\frac{1}{2}} \cdot 2^{\frac{n}{r}}$. Furthermore, De Boer et al. show that the lattice reduction is successful if the intervals have size at least $n/r + \Theta(n)$.

In the quantum case, the input basis is a superposition of all possible partitions. Therefore, there exists a lattice construction that contains a partition with small block size such that the vectors described by De Boer et al. are shorter than the vectors representing the secrets. Furthermore, these vectors have low Hamming weight and may thus be misidentified as solutions. Therefore, in the quantum setting, we recompute the public component $H' = a'G + b'$ for every basis vector and flip the oracle qubit on the outcome of the comparison with H.

Number of Grover iterations. The number of iterations is determined by the initial success probability of measuring a correct partition. The success probability depends directly on the number of correct partitions and hence on the positions of the ones, therefore varies with every secret key that is attacked. De Boer et al. [4] analyze the probability that the bits of a secret key fall into the "correct" interval. The authors showed that the lattice reduction succeeds in identifying the secret if the average size of the intervals is $O(n/\omega + \log n)$. Based on an upper and lower bound for a valid size of the intervals they derive the probability to recover the secret key on input of a random lattice as $\frac{1}{2} - c(\frac{r}{\omega})^2 + o(1)$. Since the dimension of the lattice in the case of the quantum attack is fixed to $r = 2\omega$ the expected number of partitions allowing to recover the secrets is

$$\mathbf{E}[\textit{valid partitions}] = \sum_{i=1}^{N} \left(\frac{1}{2} - 8c + o(1)\right)^{2\omega} . \tag{9}$$

In the setup of the quantum Slice-and-Dice attack the number N of all partitions is equal to the number of all possible starting positions: $N = n^{2\omega}$. The number of Grover iterations follows as

$$\left(\frac{N}{M}\right)^{\frac{1}{2}} = \left(\frac{n^{2\omega}}{n^{2\omega}\left(\frac{1}{2} - 8c + o(1)\right)^{2\omega}}\right)^{\frac{1}{2}} = \left(\frac{2}{1 - 8c + 2o(1)}\right)^{\omega} . \tag{10}$$

Bounding LLL iterations. Consider a partition of the binary expansion of the public key integer $aG + b - H \mod p$ for the Slice-and-Dice attack into intervals. Each interval is represented by a single basis vector. De Boer et al. [4] bound the maximal size of the initial vectors for a successful attack as

$$\frac{n}{2\omega}\left(1 - \frac{r(r-1)\log\gamma}{2n} - \frac{r\log r}{2n}\right), \textit{where } \gamma \textit{ is Hermite's constant}.$$

We suggest the following Heuristic 1 to bound the initial potential of a "correctly" partitioned public key and hence the required number of swap operations.

Heuristic 1 (Number of swaps for a correct Slice-and-Dice Lattice) *Let $\mathcal{L}$ be a lattice constructed from "correct" partitioning a public key $aG + b \equiv H \mod p$ of bit length at most n into ω parts. Then the LLL algorithm finds the shortest vectors after at most ${}^{n}/_{2\omega}$ swaps.*

Consider a register in superposition representing lattices constructed from partitions in superposition. The superposition represents lattices that hide the secrets and those that do not. After performing LLL iterations according to 1 the lattice representations hiding the secrets will be fully reduced. The other lattices might not be fully reduced yet. However, their basis vectors do not hide the secret vectors and are thus not relevant for flipping the oracle qubit. Therefore, one can bound the number of LLL iterations for the Slice-and-Dice attack according to Heuristic 1.

Instantiation. Consider the instantiation of a Mersenne number cryptosystem with the bit-length $n = 756839 \geq 2^{19}$ and Hamming weight $\omega = 128 = 2^7$ of the sparse integers. An implementation of the Groverized attack uses 2ω intervals where each interval represents a number in $\mathbb{Z}_n$. The quantum registers in superposition representing the partitions $|P_A\rangle, |P_B\rangle$ are can be implemented by $\geq 2^{27}$ qubits.

The oracle is constructed for a lattice of rank $2\cdot(\omega+2) \geq 2\omega$ whereas the basis vectors are of equally many coefficients, hence $d \geq 2\omega$. The basis matrix contains $(2\omega + 2)^2 \geq 2\omega^2$ coefficients. Following the heuristic the necessary number of iterations of the main loop can be bounded by ${}^{2^{19}}/_{2^8} = 2^{11}$. We base the estimate of the number of logical qubits on the largest term in Equation 8 with application of the trade-off: $r(\log\tilde{B})^{\frac{1}{2}}m)$, where m is the (qu)bit length of the coefficients of the Gram-Schmidt matrix. The Toffoli gate count is based on Equation 7.

Textbook implementation. Considering only the largest term an implemenation following our circuit representation requires a total of $r^3 d\log\tilde{B}(\log\tilde{B})^{\frac{1}{2}} \geq 2^{24}\cdot 2^{8}\cdot 2^{11}\cdot 2^{\frac{19}{2}} \geq 2^{52}$ qubits to cache the Gram-Schmidt matrices. The number of Toffoli gates would be as high as $2^{85} + 2^{66}$.

Fp-LLL à la Schnorr. Implementing the coefficients of the Gram-Schmidt matrix as floating point numbers with precision $m = r + \log\tilde{B}$ the number of qubits is reduced to at least $2^{44} + 2^{33} \geq 2^{44}$. The number of Toffoli gates would be reduced to $2^{65} + 2^{54}$.

***L^2*.** The L^2 algorithm can implement the attack in at most 2^{33} qubits and using at most 2^{55} Toffoli gates.

5 Conclusion

Our quantum circuit representation of the LLL algorithm suggests that quantum lattice reduction requires a large number of logical qubits. We do not claim optimality. The construction of reversible circuits that require less quantum memory as well as the detailed analysis of floating-point variants are lines of future work. The presented quantum design can be adapted to be used in similar reduction processes or to be used as subroutines, e.g., as a quantum shortest vector oracle. Another line of thought goes towards the evaluation of problem classes where the number of iterations of the main loop can be bounded tightly.
Our result challenges the idea that Grover's algorithm improves the attack complexity by a square root. Taking Grover's promise as given to evaluate the security of cryptosystems leads to a very pessimistic security estimates. In the case of the attack on Mersenne number cryptosystems an instantiation of the LLL algorithm as Grover's oracle requires a large amount of qubits as well as a large Toffoli gate count, throwing into question the practicality of the attack even in the context of large scale quantum computers.

Acknowledgements. The authors should like to thank the anonymous reviewers for their helpful feedback and suggestions. This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work was supported by the European Commission through the Horizon 2020 research and innovation programme under grant agreement H2020-DS-LEIT-2017-780108 FENTEC, by the Flemish Government through FWO SBO project SNIPPET S007619N and by the IF/C1 on Cryptanalysis of post-quantum cryptography. Alan Szepieniec was supported by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen), and is now supported by the Nervos Foundation.

References

1. Aggarwal, D., Joux, A., Prakash, A., Santha, M.: A new public-key cryptosystem via mersenne numbers. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. pp. 459–482. Springer International Publishing, Cham (2018)
2. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. SIAM J. Comput. **26**(5), 1510–1523 (Oct 1997). https://doi.org/10.1137/S0097539796300933
3. Beunardeau, M., Connolly, A., Géraud, R., Naccache, D.: On the hardness of the mersenne low hamming ratio assumption. IACR Cryptology ePrint Archive **2017**, 522 (2017)
4. de Boer, K., Ducas, L., Jeffery, S., de Wolf, R.: Attacks on the AJPS mersenne-based cryptosystem. In: PQCrypto. LNCS, vol. 10786, pp. 101–120. Springer (2018)
5. Cuccaro, S.A., Draper, T.G., Kutin, S.A., Moulton, D.P.: A new quantum ripple-carry addition circuit. arXiv quant-ph/0410184 (2004)
6. Duc Nguyen, T., Van Meter, R.: A space-efficient design for reversible floating point adder in quantum computing. ACM Journal on Emerging Technologies in Computing Systems **11** (06 2013). https://doi.org/10.1145/2629525

7. Fowler, A.G., Mariantoni, M., Martinis, J.M., Cleland, A.N.: Surface codes: Towards practical large-scale quantum computation. Phys. Rev. A **86**, 032324 (Sep 2012). https://doi.org/10.1103/PhysRevA.86.032324
8. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Proceedings of the Theory and Applications of Cryptographic Techniques. pp. 31–51. EUROCRYPT'08, Springer-Verlag (2008)
9. Gidney, C.: Windowed quantum arithmetic. arXiv:1905.07682 (2019)
10. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: ACM STOC. pp. 212–219. STOC '96, ACM, New York, NY, USA (1996). https://doi.org/10.1145/237814.237866
11. H. Bennett, C.: Time/space trade-offs for reversible computation. SIAM J. Comput. **18**, 766–776 (08 1989). https://doi.org/10.1137/0218053
12. Joux, A.: Algorithmic Cryptanalysis. Chapman & Hall/CRC, 1st edn. (2009)
13. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**(4), 515–534 (Dec 1982). https://doi.org/10.1007/BF01457454
14. Nachtigal, M., Thapliyal, H., Ranganathan, N.: Design of a reversible single precision floating point multiplier based on operand decomposition. In: 10th IEEE International Conference on Nanotechnology. pp. 233–237 (Aug 2010). https://doi.org/10.1109/NANO.2010.5697746
15. Nachtigal, M., Thapliyal, H., Ranganathan, N.: Design of a reversible floating-point adder architecture. In: 2011 11th IEEE International Conference on Nanotechnology. pp. 451–456 (Aug 2011). https://doi.org/10.1109/NANO.2011.6144358
16. Nguyen, P.Q., Stehlé, D.: Floating-point lll revisited. In: Cramer, R. (ed.) EUROCRYPT 2005. pp. 215–233. Springer Berlin Heidelberg (2005)
17. Nguyen, P.Q., Stehlé, D.: Lll on the average. In: Hess, F., Pauli, S., Pohst, M. (eds.) Algorithmic Number Theory. pp. 238–256. Springer Berlin Heidelberg (2006)
18. Nguyen, P.Q., Valle, B.: The LLL Algorithm: Survey and Applications. Springer Publishing Company, Incorporated, 1st edn. (2009)
19. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, New York, NY, USA, 10th edn. (2011)
20. Rines, R., Chuang, I.: High performance quantum modular multipliers. arXiv abs:1801.01081 (2018)
21. Roetteler, M., Naehrig, M., Svore, K.M., Lauter, K.E.: Quantum resource estimates for computing elliptic curve discrete logarithms. In: IACR Cryptology ePrint Archive (2017)
22. Schnorr, C.P.: A more efficient algorithm for lattice basis reduction. J. Algorithms **9**(1), 47–62 (Mar 1988). https://doi.org/10.1016/0196-6774(88)90004-1
23. Selinger, P.: Quantum circuits of t-depth one. Phys. Rev. A **87**, 042302 (Apr 2013). https://doi.org/10.1103/PhysRevA.87.042302
24. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: FOCS. pp. 124–134. SFCS '94, IEEE Computer Society, Washington, DC, USA (1994). https://doi.org/10.1109/SFCS.1994.365700
25. Szepieniec, A.: Ramstake. NIST Submission (2017), https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions
26. Wiebe, N., Kliuchnikov, V.: Floating point representations in quantum circuit synthesis. New Journal of Physics **15**(9), 093041 (sep 2013). https://doi.org/10.1088/1367-2630/15/9/093041