Finding Shortest Vector Using Quantum NV Sieve on Grover

Hyunji Kim[1], Kyoungbae Jang[1], Yujin Oh[1], Woojin Seok[2], Wonhuck Lee[2], Kwangil Bae[2], Ilkwon Sohn[2], and Hwajeong Seo[1]

[1]IT Department, Hansung University
[2]Korea Institute of Science and Technology Information (KISTI)

{khj1594012,starj1023, oyj0922}@gmail.com,
{wjseok, livezone, d2estiny, kibae}@kisti.re.kr, hwajeong84@gmail.com

Abstract. Quantum computers, especially those with over 10,000 qubits, pose a potential threat to current public key cryptography systems like RSA and ECC due to Shor's algorithms. Grover's search algorithm is another quantum algorithm that could significantly impact current cryptography, offering a quantum advantage in searching unsorted data. Therefore, with the advancement of quantum computers, it is crucial to analyze potential quantum threats.

While many works focus on Grover's attacks in symmetric key cryptography, there has been no research on the practical implementation of the quantum approach for lattice-based cryptography. Currently, only theoretical analyses involve the application of Grover's search to various Sieve algorithms.

In this work, for the first time, we present a quantum NV Sieve implementation to solve SVP, posing a threat to lattice-based cryptography. Additionally, we implement the extended version of the quantum NV Sieve (i.e., the dimension and rank of the lattice vector). Our extended implementation could be instrumental in extending the upper limit of SVP (currently, determining the upper limit of SVP is a vital factor). Lastly, we estimate the quantum resources required for each specific implementation and the application of Grover's search.

In conclusion, our research lays the groundwork for the quantum NV Sieve to challenge lattice-based cryptography. In the future, we aim to conduct various experiments concerning the extended implementation and Grover's search.

Keywords: Shortest Vector Problem · Lattice based cryptography · Quantum NV Sieve · Quantum attack · Grover's search.

1 Introduction

As outlined in IBM's roadmap [1], if a stable quantum computer with more than 10,000 qubits is developed, public key algorithms (such as Rivest, Shamir, Adle-

[1] https://www.ibm.com/quantum/roadmap

man (RSA) and Elliptic curve cryptography (ECC)) may be decrypted within polynomial time through Shor algorithm [1].

Additionally, If a search count of $O(2^k)$ on a classical computer is required, Grover's algorithm can find results with a maximum of $O(\sqrt{2^n})$ searches.

As quantum computers developed, the current cryptography system is under threat. Therefore, migration to a secure cryptography system and analysis of potential quantum attacks are very important issues.

Among the categories of post-quantum cryptography, there are lattice-based ciphers (e.g. LWE (Learning with Error)). Currently, much research has been conducted on estimating the cost of Grover attacks on symmetric key cryptography [2,3,4,5,6].

However, research on practical quantum attacks on lattice-based cryptography is lacking. As mentioned earlier, to establish a secure post-quantum security system, it is crucial to analyze potential quantum attacks on various cryptographic methods. Therefore, in this paper, we propose a quantum implementation for NV Sieve that can solve SVP (Shortest Vector Problem) for lattice-based cryptography. In addition, we present an implementation considering the dimension and rank expansion of the lattice and estimate the quantum cost for an attack through quantum NV Sieve.

1.1 Our Contributions

1. **For the first time in our knowledge, Quantum NV Sieve implementation to solve SVP**
   There is theoretical research that applies Grover's search to Sieve algorithms to solve SVP [7]. However, as far as we know, there is no implementation for these yet. In this work, we implement NV Sieve, an attack that can threaten lattice-based cryptosystems by solving SVP, as a quantum circuit. Through this, an oracle that can be applied to Grover's search is created.
2. **Extension implementation considering multiple conditions (dimension, rank) of lattice-based cryptography**
   In addition to the basic NV Sieve implementation, we present an extended implementation that takes into account the dimension and rank of the lattice. Our extended implementation can help raise the SVP upper limit that NV Sieve can solve.
3. **Resource estimation for Quantum NV Sieve logic and Grover's search**
   Grover's search algorithm has an advantage that can compute all possibilities at once. By applying Grover's search to NV Sieve, a solution that satisfies the condition can be found with quantum advantage. This approach requires an oracle, and our implementation can be used as an oracle for Grover's search. In this work, we estimate the quantum cost for each case-specific implementation (oracle).

Based on our quantum circuits, we estimate the required quantum resources for Grover's search (on NV Sieve). This is affected by quantum resources and the number of iterations[2]

1.2 Organization of the paper

The remainder of this paper is organized as follows. In Section 2, classical NV Sieve, SVP (Shortest Vector Problem), and background for quantum implementation are described. In Section 3, the implementation of the quantum NV Sieve is proposed. Section 4 demonstrates the results of the experiment and further discussion about that. Finally, Section 5 concludes the paper.

2 Prerequisites

2.1 Lattice

Lattice Lattice (L) is a set of points made up of a linear combination of basis vectors (B). Since it is made up of points, there can be more than one shortest vector (e.g. $x, -x \in L$). Equation 1 represent a lattice, and x is an integer in Equation 1, and $(b_1, ..., b_n)$ means the basis vector.

$$L(b_1, ..., b_n) = \Sigma_{i=1}^{n}(x_i \cdot b_i, x_i \in Z) \tag{1}$$

Basis As noted earlier, the lattice is based on basis vectors. A basis vector (B) is a set of vectors that can constitute all lattice points. The vector (arrow sign) in Figure 1 represents the basis in the lattice. Each vector (b_i) constituting the basis vector has a length of m and consists of a total of n components. Here, the length of each vector and the number of vectors constituting the basis vector, respectively, are called Dimension(m) and Rank(n). Generally, a full-rank lattice is used ($m = n$).

Here, the basis vector consisting of one lattice is not unique. As shown in Figure 1, the basis vectors on a lattice with the same lattice points are different. If a lattice is created with a vector created by multiplying one basis vector by another, the two basis vectors create the same lattice.

However, these basis vectors have a good basis and a bad basis. A good basis is generally composed of a short vector, and a bad basis is created by multiplying the good basis by a matrix such as an unimodular matrix[3] several times. Therefore, finding a bad basis from a good basis is easy because only matrix multiplication several times is required. However, in the opposite case, finding a good basis from a bad basis becomes a very difficult task. This can be seen as similar to generating a public key from a private key in public key

[2] Detailed estimation of Grover's search while varying the parameters of the NV sieve remains for our future work.

[3] https://en.wikipedia.org/wiki/Unimodular_matrix

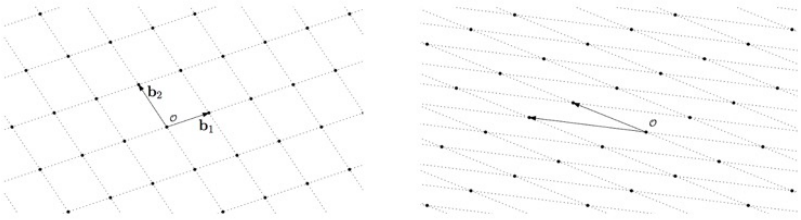

Fig. 1: Two different basis vectors generating the same lattice.

cryptography. (i.e. obtaining a private key by factorizing a very large public key into prime factors.) Similarly, in lattice-based cryptography, a bad basis is used as the public key, and a good basis is used as the private key. Here, the good basis and the bad basis are basis vectors that generate the same lattice. Constructing the public and private keys in this way makes it difficult to decrypt messages in lattice-based encryption.

2.2 Shortest Vector Problem (SVP)

SVP, known as the basic problem of lattice-based cryptography, is the problem of finding the shortest vector on a lattice that is not a zero vector. Miklo's Ajtai [8] revealed that SVP is an NP-hard problem. In addition, it was later revealed that it had almost the same level of difficulty as the Closest Vector Problem (CVP), which is another lattice-based problem. SVP is a problem of finding the shortest vector by using the basis of the lattice as input. However, the solution is not always unique because one vector can have an opposite vector with the same size.

When a bad basis vector is used as input, the difficulty of solving the SVP increases. If a good basis is used as an input, there is a high possibility that the shortest vector will be included in the already input good basis. If a bad basis is used, the opposite scenario occurs. Additionally, as the rank of the lattice (the number of vectors constituting the lattice) increases, it becomes more difficult to solve.

The lattice-based cryptography is generally used when the rank is 500 or higher. Therefore, solving lattice-based cryptography is a very challenging work. Furthermore, as mentioned earlier, one can easily derive a bad basis (public key) from a good basis (private key). However, it is difficult to find a good basis (private key) from a bad basis (public key) due to information asymmetry. Thus, solving lattice-based cryptography is challenging due to its reliance on one-wayness (the computation in one direction is straightforward but difficult in the reverse direction).

In this way, lattice-based cryptography is based on lattice problems (SVP, CVP, etc.), and the security level of lattice-based cryptography is based on the difficulty of solving the lattice problem. For example, RSA's security strength is

based on the difficulty of prime factorization. In other words, lattice-based cryptography is designed by utilizing one-wayness such as information asymmetry. To solve such lattice-based cryptography, the lattice problem must be solved. Solving SVP, a representative lattice problem, lattice-based cryptosystems such as LWE can be threatened.

Algorithms to solve SVP Several algorithms, such as AKS and Sieve, have been proposed to solve the lattice problem, which underpins lattice-based cryptography. However, these algorithms generally target low-dimensional lattices with a rank of about $50 \sim 60$. There are also algorithms that target high-dimensional lattices, but finding the shortest vector in a high-dimensional lattice is a very difficult problem. Therefore, there's a need for an approximate algorithm that can reduce the problem from a high-dimensional lattice to a low-dimensional one. As a result, to solve SVP, an exact algorithm that accurately finds the shortest vector in the low-dimensional lattice is needed and important.

Approximate algorithms that reduce high-dimensional to low-dimensional lattice (e.g., Lenstra, Lenstra, and Lovász (LLL) [9], block Korkine-Zolotarev (BKZ) [10]) have also been widely studied. Also, it is efficient in high-dimension lattices. However, as shown in Figure 2, the method for finding exactly short vectors belongs to the exact algorithm, and the best practical and theoretical SVP solution should be accurate and efficient in low dimensions. Therefore, for now, it is important to take an approach that accurately solves SVP in low dimensions. It is then important to determine the upper limit (highest dimension of lattice) that can be solved.

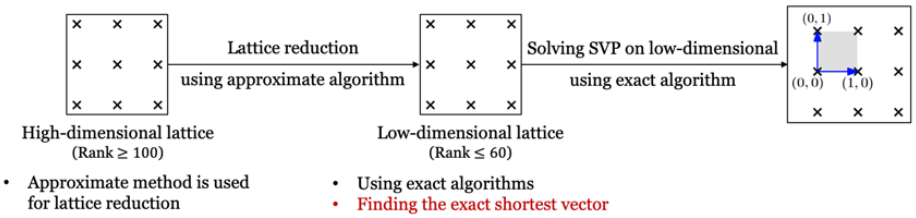

Fig. 2: Flow chart of approximate and exact algorithms for solving SVP.

2.3 Survey on the exact algorithms for SVP

Well-known exact algorithms include AKS [11] and NV Sieve [12]. AKS is the most famous early exact algorithm, but it has the disadvantage of using many parameters and having high time and space complexity. Moreover, due to the absence of optimal parameters, actual implementation, or analysis, it is deemed an impractical algorithm. Subsequently, NV Sieve, an exact algorithm, was introduced to address these limitations of AKS. It offers benefits such as reduced

time and space complexity, practicality, and the possibility for actual implementation and evaluation. Additionally, building upon the NV Sieve algorithm, several Sieve algorithms, including the List Sieve and Gaussian Sieve, have been presented [13,14,15,16,17].

However, only the theoretical complexity of the Sieve algorithm on quantum computers (using Grover's search) has been calculated [7]. There is no practical implementation or analysis for this.

2.4 NV Sieve algorithm

Reasons and overview for selecting the NV Sieve algorithm NV Sieve is more practical and efficient than AKS and serves as the foundation for numerous Sieve algorithms. So, in our work, NV Sieve is selected as an exact algorithm for solving the SVP problem. Although there are algorithms with lower time and space complexity than NV Sieve, quantum computing can incur significant costs when implementing algorithms that require additional procedures. Of course, a simple algorithm is not necessarily efficient when executed on a quantum computer.

Algorithm 1: NV Sieve algorithm for finding short lattice vectors

Input: An reduced basis (B) in lattice (L) using the LLL algorithm, a sieve factor γ ($\frac{2}{3} < \gamma < 1$), S is an empty set, and a number N
Output: A non-zero short vector of L

for $i = 1$ to N **do**
    $S \leftarrow$ Sampling B using sampling algorithm
end for
Remove all zero vectors from S.
$S_0 \leftarrow S$
Repeat
    $S_0 \leftarrow S$
    $S \leftarrow$ `latticesieve`$(S, \gamma R)$ using Algorithm 2.
    Remove all zero vectors from S.
until S becomes an empty set.
Return $v_0 \in S_0$ such that $||v_0|| = \min||v||, v \in S_0$

Details of NV Sieve algorithm Algorithm 1 briefly shows the main process of NV Sieve. The goal of NV Sieve is to find the shortest vector excluding zero vectors while losing as few vectors as possible. The input is the basis vector of the lattice reduced through the approximate algorithm (i.e., LLL), and the output is the shortest vector, not the zero vector. As mentioned earlier, the shortest vector may not be one. In addition, γR, the sieve factor, is a geometric element in the range of $\frac{2}{3} < \gamma R < 1$, and the closer it is to 1, the better. The reduction range

of the lattice, which will be explained later, is determined by the corresponding sieve factor.

The overall structure is as follows. First, a set S is generated by randomly sampling from the basis received as input. Next, the zero vector is removed from S to generate S_0, and then the `latticesieve` is repeatedly performed with S and γ as input. After this, the output vectors with zero vectors removed are stored in S_0, and the process is repeated until S becomes an empty set. Finally, it is completed by returning the shortest vector among the vectors belonging to S_0.

Algorithm 2: The `latticesieve` algorithm in NV Sieve

Input: A subset S in L and sieve factor γ $(0.666 < \gamma < 1)$
Output: S' (Short enough vector, not zero vector)

1: Initialize C, S' to empty set.
2: $R \leftarrow max_{v \in S}||v||$
3: **for** $v \in S$ **do**
4: **if** $||v|| \leq \gamma R$ **then**
5: $S' \leftarrow S' \cup \{v\}$
6: **else**
7: **if** $\exists c \in C||v - c|| \leq \gamma R$ **then**
8: $S' \leftarrow S' \cup \{v - c\}$
9: **else**
10: $C \leftarrow C \cup \{v\}$
11: **end if**
12: **end if**
13: **end for**
14: **return** S'

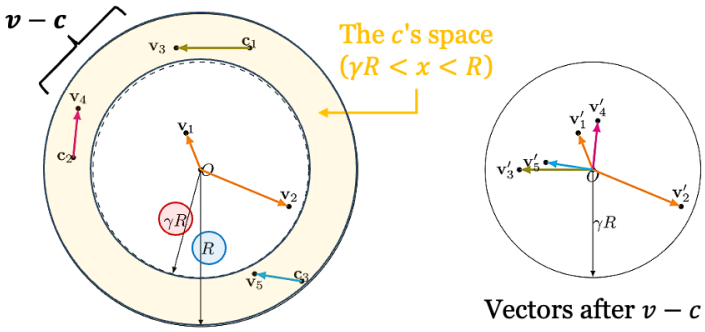

Fig. 3: The core logic in NV Sieve (See line 7 in Algorithm 2).

Algorithm 2 shows the lattice sieve algorithm in NV Sieve and shows the detailed process. This sieve algorithm is the core logic of NV Sieve, and its purpose is as follows.

- In order to minimize the loss for short vectors, a point on the lattice called c is randomly selected. c is a sufficient number of points on the lattice belonging to $\gamma R < x < R$ and belongs to the yellow area in Figure 3.
- The search range (γR) is reduced by the sieve factor γ to obtain a vector shorter. Here, R means the maximum length among the vectors belonging to the vector set received as input.

The core logic of the NV sieve mentioned earlier in more detail is as follows.

1. First, initialize C and S'. Afterward, vectors with a length shorter than γR are stored in S'. (S' is used to store vectors within the γR range.)
2. However, there will be vectors longer than γR. For this, the process as in line 7 is performed to minimize loss for short vectors on the lattice, which is the goal of NV Sieve.
   A vector longer than γR is subtracted from a point on the lattice called c. If the result is shorter than γR, then it is stored in S'. If the length is longer than γR, it is stored in C. In other words, when the vector after subtraction starts from O (origin point), if it is within the range of γR, it is stored in S'.
3. Finally, by returning S', vectors with a length shorter than γR are selected. By performing this process repeatedly, sufficiently short vectors are obtained, and the shortest vector among them is found.

Important factors related to the complexity The parts that affect the complexity of NV Sieve's algorithm are as follows. The first part is measuring the number of points in c. There are a sufficiently large number of points on the lattice, and we need to find a point c that can be used to create a vector with a length shorter than γR. Therefore, it is important to find the number of c. Next, as the size of the initially given vector set S increases, complexity increases. As the rank of S increases, the number of c also increases because c is also a vector on the lattice and a subset of S. This is related to the complexity related to the number of c mentioned above. Additionally, as mentioned earlier, lattice problems with large ranks are difficult to solve, so the size of the target basis vector set affects the complexity of the algorithm.

2.5 Grover's search algorithm

Grover's search algorithm is a quantum search algorithm for tasks with n-bit complexity and has $O(\sqrt{2^n})$ of complexity ($O(2^n)$ for classical). The data (n-bit) for the target of the search must exist in a state of quantum superposition, so given by:

$$H^{\otimes n} |0\rangle^{\otimes n} (|\psi\rangle) = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle \tag{2}$$

Thanks to quantum advantage, all search targets are computed simultaneously as a probability.

Grover's algorithm consists of two modules: Oracle and Diffusion operator. Oracle is a quantum circuit that implements logic that can return a solution to the problem to be solved. Then it returns a solution by inverting the decision qubit at the end of the circuit as follows.

$$f(x) = \begin{cases} 1 \text{ if } Oracle_{\psi(n)} = Solution \\ 0 \text{ if } Oracle_{\psi(n)} \neq Solution \end{cases} \tag{3}$$

Afterwards, the probability of the returned solution is amplified through the diffusion operator. By repeating this process, the probability of observing the correct solution is increased. The number of such repetitions is expressed as Grover iteration. The most important thing in Grover's search is the optimal implementation of the quantum circuit that designs the oracle.

The diffusion operator has a fixed implementation method and is often excluded from resource estimation [5,18] because the overhead is so small that it is negligible. Therefore, the final efficiency is determined depending on the quantum circuit in the oracle.

2.6 Quantum Circuit

Qubits A qubit (quantum bit) is the basic unit of computation in a quantum computer and can have probabilities of 0 and 1 at the same time (superposition). So, 2^n states can be expressed with n qubits. Additionally, qubits exist in a superposition state and are calculated, but are determined as a single classical value the moment they are measured. In quantum computing, classical bits are used to store the results of measuring the state of the qubit.

Quantum Gates Quantum gates operate as logical gates in quantum circuits. By applying a quantum gate to a qubit, the state of the qubit can be controlled. There are several quantum gates (see Figure 4). Each gate can be used to configure superposition, entanglement, invert, and copy, and can be utilized to perform various operations such as addition and multiplication.

3 Quantum NV Sieve for solving SVP

3.1 System Overview

According to the results of theoretical calculations, Quantum NV Sieve with Grover's search is expected to have less time complexity than classical NV Sieve ($log_2^{0.415}$ to $log_2^{0.312}$) [7]. However, no implementation is presented. To the best of our knowledge, our work presents the first implementation of various cases of the NV Sieve algorithm for solving SVP using quantum circuits. However, given the current state of quantum computers in the Noisy Intermediate-Scale Quantum

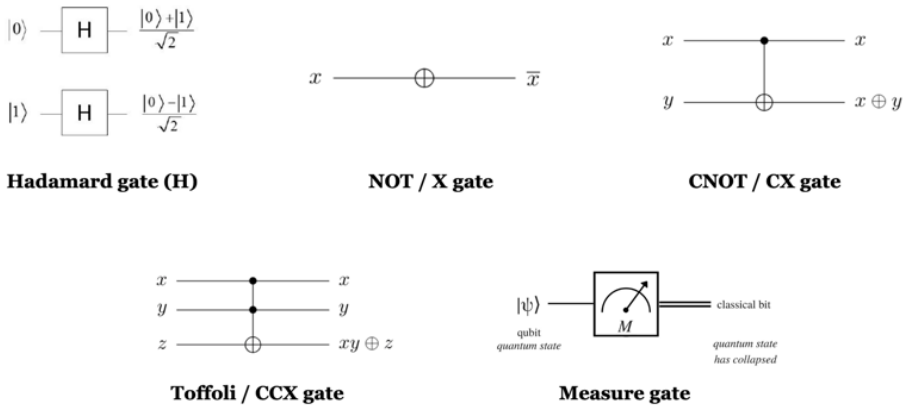

Fig. 4: Quantum gates.

(NISQ) era and the challenges encountered during implementation, achieving results akin to the theoretical complexity remains challenging. Starting with our work, we plan to further improve our approach, which we remain for our future work.

As noted earlier, solving SVP, the fundamental problem of lattice-based encryption, can threaten grid-based encryption systems (e.g., LWE). Furthermore, among several algorithms, the Exact algorithm, that accurately finds short vectors, is an important part of the process of solving lattice problems.

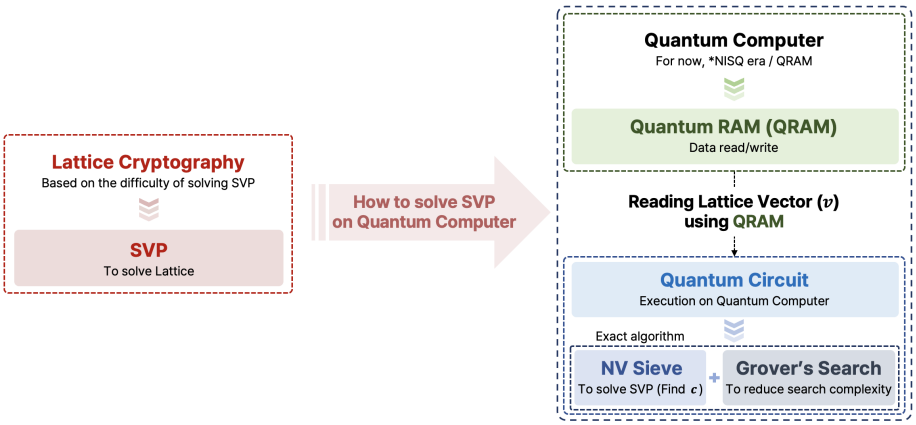

Fig. 5: Overview of Quantum NV Sieve.

We implemented the NV Sieve algorithm, which solves the SVP problem among several lattice problems (e.g., SVP, CVP, etc.), on a quantum computer.

Figure 5 shows the overview of the quantum NV Sieve algorithm. In other words, this is the overall relationship between the quantum NV Sieve we present and the configuration diagram for solving SVP on a quantum computer.

First, since Grover's search must be applied, the logic of the NV Sieve (oracle) must be implemented using the quantum circuit. In other words, since the purpose of NV Sieve is to find the short vector c that satisfies the condition ($||v - c|| \leq \gamma R$), the NV Sieve logic for searching c must be implemented as an oracle. Then, Grover's search algorithm should be performed on the implemented oracle. The factors that determine the performance of NV Sieve in classical are finding the number of large numbers of c and the corresponding computational and memory complexity. However, when using quantum NV Sieve, it is possible to calculate numerous cases for c at once. Therefore, there are advantages in terms of computational and memory complexity.

Meanwhile, v, which is not the search target but is a vector on the lattice, needs to be loaded from quantum memory. However, it is difficult to access actual QRAM (Quantum RAM). In addition, many studies, such as [19], are conducted on the premise that queries can be made to QRAM. Therefore, in this implementation, QRAM is implemented as a very simple quantum circuit (Explicit QRAM: data is written directly to the quantum circuit, and the value is loaded from the corresponding memory qubit).

3.2 Implementation of NV Sieve on Quantum Circuit

We implement/design the quantum circuit for line 7 in Algorithm 2. It operates classically except where quantum NV sieve algorithms are used. In other words, quantum is applied to operations on a sufficiently large number of c. In a classical computer, we need to know how many c there are and perform a size comparison on all c. However, in the implementation of the quantum NV sieve, a size comparison is performed on all cases of c at once. Details are described in Algorithm 3.

The overall steps in Algorithm 3 are as follows:

1. **Data load from explicit QRAM (line 3):** It is difficult to actually access QRAM. Therefore, we implement a simple explicit QRAM on a quantum circuit. This is actually close to QROM (Quantum Read-only Memory) because it can only read data to be used.
2. **Prepare c in superposition state (line 4∼5):** Apply the Hadamard gate to c, Grover's search target, and prepare it in a superposition state. Since v is not a search target, it doesn't make it a superposition state.
3. **Prepare $(sqr\_rR)^2$ (line 6):** Prepare the squared γR.
4. **Complement function for signed vector (line 8∼9):** For data involving signed vectors, the complement operation is utilized to repurpose the adder as a subtractor. When comparing vector magnitudes at the conclusion of the quantum circuit, the complement operation is currently applied solely to positive vectors.
5. **Two-qubit addition (line 11∼12):** For vector elements $(v + \bar{c})$, a 2-qubit ripple carry adder is applied between v and the complements of c.

Algorithm 3: The quantum NV Sieve on the quantum circuit.

Input: Quantum circuit (QNV), A subset S in L and sieve factor γ ($\frac{2}{3} < \gamma < 1$)
Output: c_0, c_1

Initiate quantum registers and classical registers. ▷ $carry, qflag, sqr\_result$, etc.
// Input setting (Each vector is allocated 2-qubits)
$v_0, v_1 \leftarrow$ Data load from memory qubits
$QNV.Hadamard(c_0)$
$QNV.Hadamard(c_1)$
$QNV.x(sqr\_rR[i])$ ▷ $0 \leq i < 4$

// Two's complement for subtraction using adder
$c_0 \leftarrow$ Two's complement$(QNV, c_0, qflag0, zero)$
$c_1 \leftarrow$ Two's complement$(QNV, c_1, qflag1, zero)$

// $v + \overline{c}$
$c_0 \leftarrow$ Addition$(QNV, v_0, c_0, carry)$
$c_1 \leftarrow$ Addition$(QNV, v_1, c_1, carry)$

// Two's complement for correct squaring
$c_0 \leftarrow$ Two's complement_negative$(QNV, c_0, qflag2, carry, zero)$
$c_1 \leftarrow$ Two's complement_negative$(QNV, c_1, qflag3, carry, zero)$

// Duplicating qubit for squaring
$dup\_c_0 \leftarrow QNV.cx(c_0, dup\_c_0)$
$dup\_c_1 \leftarrow QNV.cx(c_1, dup\_c_1)$

// Squaring elements of vectors
$sqr\_result[1] \leftarrow$ Squaring$(QNV, c_0, dup_c_0, sqr_result[0], sqr_result[1], carry, 4)$
$sqr\_result[3] \leftarrow$ Squaring$(QNV, c_1, dup_c_1, sqr_result[2], sqr_result[3], carry, 4)$

// Addition for squared results to calculate the size of the vector
$sqr\_result[3] \leftarrow$ Addition$(QNV, sqr_result[1], sqr_result[3], carry, 4)$

// Two's complement for subtraction using adder
$sqr\_result[3] \leftarrow$ Two's complement_4bit$(QNV, sqr_result[3], qflag4, carry, zero, zero1, zero2)$

// Size comparison between $(rR)^2$ and $(||v - c||)^2$ ($(rR)^2$ - $(||v - c||)^2$)
$sqr\_result[3] \leftarrow$ Addition$(QNV, sqr_rR, sqr_result[3], carry, 4)$ ▷ No square root
return c_0, c_1

6. **Apply complement function for 2 qubits to ensure correct square value (line 14∼15):** In the complement system, 11_2 is –1, but if the complement operation for negative numbers is not performed before the square operation, 11_2 is recognized as 3. Then, the result is 9. Therefore the complement operation must be applied for the correct result of squaring.
7. **Duplicate the target qubits for squaring (line 17∼18):** In a quantum circuit, performing calculations on identical qubits is not feasible; therefore, the value must be copied to a different qubit.
8. **Squaring each element to calculate the size of the vector (line 20∼21):** The size of a vector is the root of the sum of the squares of each element. However, in our oracle only size comparison between γR and $||v-c||$ is required. Therefore, the root operation is removed in our approach. So we only need the squaring operation of the vector at this stage.
9. **4-qubit addition of each element of the vector after squaring (line 23):** To calculate the size of a vector, a square operation is required for each element.
10. **4-qubit complement for positive values (line 25):** The value after squaring is naturally a positive value. However, as in the previous part of the quantum circuit, we perform a complement operation to use the adder as a subtractor. Here, since it is the value after squaring a 2 qubit vector, a complement operation on 4 qubits must be performed.
11. **Size comparison through 4-qubit addition for $(\gamma R)^2$ and ($\overline{||v-c||^2}$) (line 27)**: As mentioned earlier, the size of the vector can be obtained by performing the root operation. However, in our method, since the only purpose is size comparison, the root operation is not performed.
12. **Check the MSB (Most Significant Bit):** We have to check the MSB of the result value performed in step 9. If MSB is 0, $(\gamma R)^2$ is larger than $\overline{||v-c||^2}$. Therefore, MSB of 0 means that $||v-c||^2$ is a short vector that falls within the range of the condition. Therefore, we can add vector $v-c$ to the list that stores short vectors (Classical).
    Conversely, if MSB is 1, it means that it is a negative sign, which means that it is a vector that does not satisfy the condition. Therefore, it is not added to the short vector list.

Implementation details for core functions in Quantum NV Sieve

– **Data load and input Setting $(v, c, (\gamma R)^2)$:** Figure 6 shows the data load phase using a quantum circuit with 4 qubit memory cells. In our implementation, we use a simple QRAM structure. After allocating a memory qubit for value storage, the values are stored in the corresponding memory qubit. Afterward, the cx gate is used to read the values stored in the memory qubit, and the values are loaded into the input vector v. In other words, it is the process of copying values from quantum memory to input qubits for the oracle. Additionally, Grover's search is repeated for each v, and v is not a search target, so it is not prepared in a superposition state.

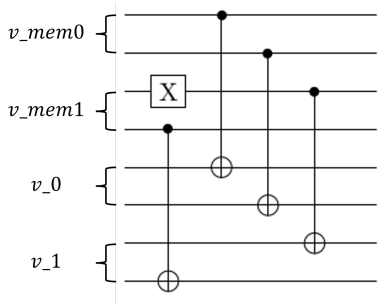

Fig. 6: Input vector (v) load with Explicit QRAM.

Figure 7 shows the input setting process for c, the search target. What must be found through Grover's search algorithm is the c value that satisfies the condition, and the $v-c$ vector at that time must be returned. Therefore, the Hadamard gate is applied to all qubits for c, generating a superposition state with the same probability of 0 and 1.

Next, a process is needed to set $(\gamma R)^2$ required for the conditional expression. This applies the x gate to the qubit to express 1 (the same as setting the v value). However, the γR is determined in each iteration. So, in our implementation, its squaring value is calculated in a classical method and then set as input. Therefore, since it is a square value for 2 qubit data, 4 qubits are allocated.

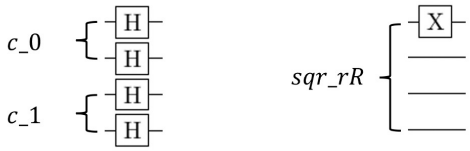

Fig. 7: Preparation c (c_0 and c_1) and $(\gamma R)^2$ (sqr_rR).

– **Two's Complement (2-qubit, 4-qubit, positive and negative cases):** Figure 8 shows the quantum circuit for 2's complement for positive values. As mentioned earlier, an additional qubit (ancilla qubit, $qflag$) is needed as a control qubit. When the target qubit to which the complement will be applied is c, the MSB is $c[1]$ (lowest qubit). Therefore, the value of the lower qubit is copied to the control qubit through the cx gate. Here, bit inversion and addition of LSB and 1 must be performed only when the value is positive. However, if MSB is zero, the value of the control qubit is 0, so the value of the control qubit must be inverted. But, after applying the x gate to $qflag$, the value of the control qubit is 1, so complement logic is performed. After bit inversion, to add the value of 1 to the LSB, create a new qubit

array, input $qflag$ as the lowest bit first, and then append the value of 0. Afterwards, addition is performed through a 2 qubit adder between the 1's complement (2 qubits) and the new qubit array (2 qubits).

The quantum circuit for 2's complement for the negative values is performed to calculate the correct squaring on the signed data. This uses control qubits like two's complement when positive. However, for negative numbers, the MSB itself is 1, so there is no need to apply the x gate to $qflag$ (omit the x gate for $qflag$). Therefore, the bit is inverted through the cx gate without additional work. Afterward, the process for adding 1 to LSB is also performed in the same way.

The 2's complement quantum circuit for 4 qubits is similar to the 2-qubit complement quantum circuit, which is performed only when the number is positive. However, since it is 4 qubits, the MSB is $c[3]$ (Only the index of MSB is different). Therefore, after copying the value to $qflag$, apply the x gate to invert all bits. Afterwards, a new qubit with the state of [0,0,0,1] is assigned and a 4-qubit addition is performed. Through this, 2's complement operations on 4 qubits can be performed.

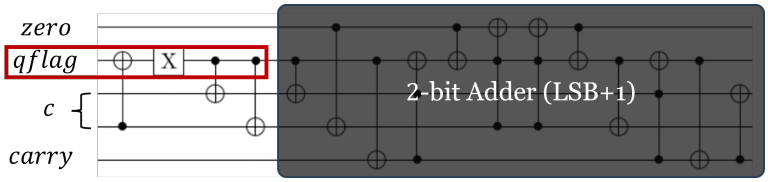

Fig. 8: Two's complement quantum circuit for a positive value (2-qubit).

- **Addition:** Addition is a very important and basic operation among quantum circuit operations. In this implementation, Ripple-Carry Adder is used for 2-qubit and 4-qubit addition. These are the method proposed in Cuccaro's paper [20].
- **Squaring:** The square operation is necessary to find the size of the vector. An integer square operation is performed on the value converted to a positive number through the 2's complement. Figure 9 depicts the square operation. The squaring is equivalent to multiplying the same value, so a and b are the same value. However, in quantum circuits, operations using the same qubit repeatedly are impossible. In other words, as shown in Figure 9, the value a must be copied to another qubit (b) through the cx gate. Additionally, performing a multiplication on 2 qubits affects up to 4 qubits, so two 4-qubit arrays (ab and ba) are created to store the result.

  The process is as follows. First, multiply a and b, which represent the same value, like integer multiplication. However, all elements are qubits and therefore have a value of 0 or 1. If even one element is 0, the result value is 0, and only if both elements are 1, the result value is 1. These operations correspond to the ccx (Toffoli) gate. Therefore, the ccx gate is applied to

all elements of a and b. Afterward, the results are saved in an appropriate location. Here, the location where the calculation results are saved is the gray part of the red box in Figure 9. However, since both ab and ba are 4-qubit arrays, the top two qubits of ab are set to 0, and the MSB and LSB of ba are set to 0. Finally, the square operation is completed by applying a 4-qubit adder to ab and ba. The adder used is CDKM adder, an in-place ripple carry adder, so the result value is stored in ba. Figure 10 represents a quantum circuit for the square operation.

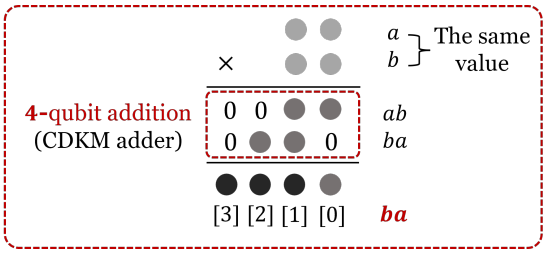

Fig. 9: The integer squaring for 2-qubits.

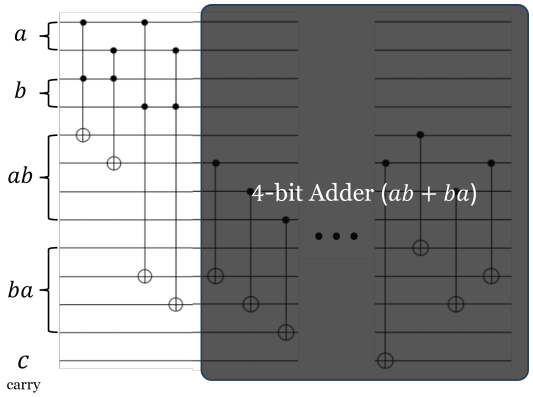

Fig. 10: Quantum circuit of squaring for 2-qubits.

3.3 Implementation for dimension expansion

Increasing dimension means that the range of bits that can be expressed by each element of the vector increases. In other words, operations on 2 qubits must be changed to operations on n ($n < 2$) qubits. In our work, we implement the case

where the dimension is increased to 4. This indicates that our implementation is scalable in terms of dimensionality. In this case, the 2-qubit adder must become a 4-qubit adder, and the 2-qubit two's complement must become a 4-qubit two's complement. Therefore, in accordance with this increased data range, the range of functions for calculations must also be expanded.

3.4 Implementation for rank expansion

Even if the rank of the input vector increases, the formula for calculating the size of the vector remains the same. Therefore, it is implemented by allocating additional qubits as needed depending on the number of extended rank. Neither the type nor the scope of the operation used changes. The same operation is performed on the elements of the new vector. In the case of the addition, it can be implemented by adding another vector to the result of adding two vectors. Hence, our implementation offers scalability as the rank of the input lattice vector increases.

4 Evaluation

4.1 Experiment Environment

Our implementation utilized Qiskit [4], a quantum computing platform. The cloud platform provides IBM's real hardware and simulators. Additionally, programming can be possible using Python and Qiskit's grammar, allowing access to the quantum computing environment. We use the 'matrix_product_state' simulator, which can provide relatively large-scale qubits.

4.2 Result of Quantum NV Sieve

Table 1 shows the results of each step of our implementation for quantum NV Sieve. The complement expression of x is $\overline{x}$, and the abbreviation of the previous step is sometimes used in the next step to prevent the output term from becoming long. On the other hand, we present results for `Default`, `Ex_DIM`, and `Ex_RANK`. The extension to dimension (`Ex_DIM`) increases the length of the vector ($v_0 = \{0, 1\}$ to $v_0 = \{0, 0, 0, 1\}$). The extension to rank (`Ex_RANK`) increases the number of elements ($V = \{v_0, v_1\}$ to $V = \{v_0, ..., v_n\}$).

Through the result of quantum NV Sieve logic, we present a scalable implementation that takes into account various situations on the lattice. Correct values are output at all steps. This allows us to verify the suitability of our quantum NV Sieve for practical implementation. Furthermore, this extended implementation can help raise the SVP upper limit that NV Sieve can solve. In our work, we confirmed that the NV Sieve algorithm operates accurately on a quantum circuit. Based on our work, we can expect that the possibility of solving the larger problem will increase as the scale of quantum computers expands.

[4] https://qiskit.org/

Table 1: Results from each step of quantum NV Sieve to check whether it has been implemented correctly. (`Default`: 2-qubit and 2 rank, `Ex_DIM`: 4-dimension and 2 rank, `Ex_RANK`: 2-qubit and 3 rank)

Output	`Default`	`Ex_DIM`	`Ex_RANK`
v_0	00	0111	00
v_1	01	0011	01
v_2	None	None	01
c_0	01	1001	01
c_1	00	0101	01
c_2	None	None	11
$(\gamma R)^2$	0001	00000001	0001
$\overline{c_0}$ (when positive)	11	1001	11
$\overline{c_1}$ (when positive)	00	1011	11
$\overline{c_2}$ (when positive)	None	None	11
$v_0 + \overline{c_0}$: (vc_0)	11	0000	11
$v_1 + \overline{c_1}$: (vc_1)	01	1110	00
$v_2 + \overline{c_2}$: (vc_2)	None	None	00
$(vc_0)^2$	01	00000000	01
$(vc_1)^2$	01	00000100	00
$(vc_2)^2$	None	None	00
$(vc_0)^2 + (vc_1)^2 + (vc_2)^2$: $(Sum_v c)$	0001	0000100	0001
$\overline{Sum_v c}$	1110	11111100	1111
$\gamma R + \overline{Sum_v c}$	1111	11111101	0000
MSB	1	1	0
Shots	100		

4.3 Resource Estimation of Quantum NV Sieve

Table 2 shows the resource estimation of quantum NV Sieve. Since this is a resource estimate for Oracle, the result also includes resources for reverse operation. Contrary to the traditional Grover's search that identifies a single solution, the NV Sieve yields multiple outcomes. That is, it may produce multiple short vectors meeting the condition, with probabilities varying based on the number of shots. Therefore, determining the correct Grover's iteration is a very important issue.

The required quantum resources increase as the rank and dimension of the target vector increase. Even if the dimension is doubled, the total quantum cost increases by about 8.38 times, and even if the rank increases by just one, the total cost increases by about 1.98 times. However, a real lattice will have larger dimensions and ranks. Therefore, if the dimension and rank increase simultaneously, the quantum cost of the quantum NV Sieve is expected to increase enormously.

$$2 \cdot \#gates \cdot FD \cdot iter \tag{4}$$

Additionally, when applying Grover, the total number of gates ($\#gates$) mentioned in Table 2 must be multiplied by full depth (FD). Then, we need to multiply by 2 (reverse operation) and multiply by the number of Grover's iterations ($iter$). In other words, the formula for calculating Grover's attack cost is as shown in Equation 4. That is, in addition to quantum resources (i.e. the number of gates and circuit depth), Grover's iteration affects the attack cost. Table 3 is calculated from Table 2 and Equation 4. Table 3 shows the required quantum resources for Grover's search on NV Sieve. The number of qubits in every case increases by 1 because of the decision qubit.

Table 2: Resource Estimation of Quantum NV Sieve logic.

Case	#CNOT	#1qCliff	#T	T-depth	full depth	#Qubit
`Default`	157	29	62	206	603	50
`Ex_RANK`	213	39	90	298	866	57
`Ex_DIM`	457	145	194	582	1576	117

Table 3: Required quantum resources for Grover's search on NV Sieve.

Case	Total gates	Full depth	T-depth	Quantum cost	#Qubit
`Default`	$1.9375 \cdot 2^{(8+r)}$	$1.1777 \cdot 2^{(10+r)}$	$1.6094 \cdot 2^{(8+r)}$	$1.1409 \cdot 2^{(19+2r)}$	51
`Ex_RANK`	$1.3359 \cdot 2^{(9+r)}$	$1.6914 \cdot 2^{(10+r)}$	$1.1641 \cdot 2^{(9+r)}$	$1.1298 \cdot 2^{(20+2r)}$	58
`Ex_DIM`	$1.5547 \cdot 2^{(10+r)}$	$1.1367 \cdot 2^{(10+r)}$	$1.5391 \cdot 2^{(11+r)}$	$1.1964 \cdot 2^{(22+2r)}$	118

※: r is the number of iterations.

4.4 Further discussion

According to our implementation mentioned above, it is expected that quantum gain can be obtained through Grover's search. Of course, there will certainly be implementation challenges as follows. Also, in the current quantum computing environment, it is believed that there will be many difficulties from an implementation perspective to derive results similar to the theoretically proposed complexity of the quantum NV Sieve.

- **Grover's iteration:** Since iteration affects the cost, finding an iteration for a problem that has multiple solutions is the most important challenge in the practical implementation of Quantum NV Sieve.

- **Increase the upper limit:** The important thing to solve the current SVP is to accurately find short vectors and increase the upper limit of the dimension that can be solved. In other words, the Sieve algorithm belongs to the exact algorithm, and it is important to solve it accurately starting from low dimensions. Therefore, we should start experimenting with low dimensions and ranks, as we do now, and then work our way up to higher limits.

- **Optimizing the oracle circuit:** In order to improve the efficiency of the quantum NV Sieve and maximize the benefits that arise from applying quantum, it is thought that optimal implementation of the oracle will be important. In other words, it appears that the optimal implementation of the oracle (NV Sieve quantum circuit), which determines the efficiency of quantum costs in Grover's search, must be progressed to solve SVP on a higher-dimensional and rank lattice and obtain greater quantum advantages.

- **NISQ era:** As the resource estimation results indicate, quite a bit of attack cost is required despite the small dimensions and rank. Therefore, it is believed that there will be limitations in allowing general users to treat lattice vectors with higher rank and dimension. In other words, it is thought that solving SVP for high dimensions (50∼60 dimensions) such as classical is difficult for now.

5 Conclusion

In conclusion, there are quantum threats to traditional cryptographic systems, especially as quantum computing technology advances. While the most of research has focused on the potential impact of Grover's algorithm on symmetric key cryptography, the field of quantum attacks on lattice-based cryptography on Grover's search remains underexplored.

To address this gap and solve SVP on quantum computers, our work introduces a practical implementation of Quantum NV Sieve, designed to solve the SVP for hacking lattice-based cryptography. This implementation is an oracle that is a vital component of Grover's search algorithm. Furthermore, our work extends the Quantum NV Sieve implementation to handle various conditions (i.e., expansion of dimensions and rank of the lattice) thereby increasing its applicability and impact.

We estimate the quantum resources required for each case-specific implementation (oracle) and predict the cost of Grover's attacks when applied in conjunction with their Quantum NV Sieve. Like this, in a rapidly evolving quantum field, our research addresses the new potential quantum threats practically.

In our future work, we plan to find the correct Grover's iteration in the condition that there are multiple solutions, and successfully sieve the short vectors.

References

1. P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, Ieee, 1994. 2
2. K. Jang, S. Choi, H. Kwon, H. Kim, J. Park, and H. Seo, "Grover on korean block ciphers," *Applied Sciences*, vol. 10, no. 18, p. 6407, 2020. 2
3. K. Jang, S. Choi, H. Kwon, and H. Seo, "Grover on speck: quantum resource estimates," *Cryptology ePrint Archive*, 2020. 2
4. K. Jang, G. Song, H. Kim, H. Kwon, H. Kim, and H. Seo, "Efficient implementation of present and gift on quantum computers," *Applied Sciences*, vol. 11, no. 11, p. 4776, 2021. 2
5. K. Jang, A. Baksi, H. Kim, G. Song, H. Seo, and A. Chattopadhyay, "Quantum analysis of aes," *Cryptology ePrint Archive*, 2022. 2, 9
6. M. Rahman and G. Paul, "Grover on katan: Quantum resource estimation," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–9, 2022. 2
7. T. Laarhoven, M. Mosca, and J. Van De Pol, "Finding shortest lattice vectors faster using quantum search," *Designs, Codes and Cryptography*, vol. 77, pp. 375–400, 2015. 2, 6, 9
8. M. Ajtai, "The shortest vector problem in l2 is np-hard for randomized reductions," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 10–19, 1998. 4
9. P. Q. Nguyen and B. Vallée, *The LLL algorithm*. Springer, 2010. 5
10. C.-P. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," *Mathematical programming*, vol. 66, pp. 181–199, 1994. 5

11. M. Ajtai, R. Kumar, and D. Sivakumar, "A sieve algorithm for the shortest lattice vector problem," in *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pp. 601–610, 2001. 5
12. P. Q. Nguyen and T. Vidick, "Sieve algorithms for the shortest vector problem are practical," *Journal of Mathematical Cryptology*, vol. 2, no. 2, pp. 181–207, 2008. 5
13. X. Wang, M. Liu, C. Tian, and J. Bi, "Improved nguyen-vidick heuristic sieve algorithm for shortest vector problem," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pp. 1–9, 2011. 6
14. F. Zhang, Y. Pan, and G. Hu, "A three-level sieve algorithm for the shortest vector problem," in *International Conference on Selected Areas in Cryptography*, pp. 29–47, Springer, 2013. 6
15. T. Laarhoven, "Sieving for shortest vectors in lattices using angular locality-sensitive hashing," in *Advances in Cryptology–CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I 35*, pp. 3–22, Springer, 2015. 6
16. A. Becker, N. Gama, and A. Joux, "Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search," *Cryptology ePrint Archive*, 2015. 6
17. D. Micciancio and P. Voulgaris, "Faster exponential time algorithms for the shortest vector problem," in *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pp. 1468–1480, SIAM, 2010. 6
18. S. Jaques, M. Naehrig, M. Roetteler, and F. Virdia, "Implementing grover oracles for quantum key search on aes and lowmc," in *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, pp. 280–310, Springer, 2020. 9
19. M. Kaplan, G. Leurent, A. Leverrier, and M. Naya-Plasencia, "Quantum differential and linear cryptanalysis," *arXiv preprint arXiv:1510.05836*, 2015. 11
20. S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," *arXiv preprint quant-ph/0410184*, 2004. 15