Quantum NV Sieve on Grover for Solving Shortest Vector Problem

Hyunji Kim, Kyungbae Jang, Hyunjun Kim, Anubhab Baksi, Sumanta Chakraborty, and Hwajeong Seo *

[1] H. Kim Hansung University, Seoul, South Korea
khj1594012@gmail.com
[2] K. Jang Hansung University, Seoul, South Korea
starj1023@gmail.com
[3] H. Kim Hansung University, Seoul, South Korea
khj930704@gmail.com
[4] A. Baksi Nanyang Technological University, Singapore
anubhab001@e.ntu.edu.sg
[5] S. Chakraborty Techno International New Town, India
sumantapapan@gmail.com
[6] H. Seo Hansung University, Seoul, South Korea
hwajeong84@gmail.com
Corresponding author

Abstract. Quantum computers can efficiently model and solve several challenging problems for classical computers, raising concerns about potential security reductions in cryptography. NIST is already considering potential quantum attacks in the development of post-quantum cryptography by estimating the quantum resources required for such quantum attacks. In this paper, we present quantum circuits for the NV sieve algorithm to solve the Shortest Vector Problem (SVP), which serves as the security foundation for lattice-based cryptography, achieving a quantum speedup of the square root. Although there has been extensive research on the application of quantum algorithms for lattice-based problems at the theoretical level, specific quantum circuit implementations for them have not been presented yet. Notably, this work demonstrates that the required quantum complexity for the SVP in the lattice of rank 70 and dimension 70 is 2^{43} (a product of the total gate count and the total depth) with our optimized quantum implementation of the NV sieve algorithm. This complexity is significantly lower than the NIST post-quantum security standard, where level 1 is 2^{157}, corresponding to the complexity of Grover's key search for AES-128.

Keywords: Shortest Vector Problem · Grover's search · Lattice-based cryptography · Quantum Security.

1 Introduction

As outlined in IBM's roadmap[7], when a stable and robust quantum computer with more than 10,000 qubits is developed, public-key algorithms (such as Rivest, Shamir, Adleman (RSA) and Elliptic curve cryptography (ECC)) may be decrypted within polynomial time through Shor algorithm [1]. Additionally, while classical computers may require a search count of $O(2^k)$ for k-bit data, Grover's search algorithm can achieve results with $O(\sqrt{2^k})$ iterations.

The progression of quantum computing poses a significant threat to contemporary cryptographic systems. Therefore, migration to a secure cryptography system (e.g., post-quantum cryptography) and the analysis of potential quantum attacks are very important issues.

Among the post-quantum cryptography categories, lattice-based cryptography such as Learning With Error (LWE) is gaining attention. NIST finalist also includes many lattice-based cryptography (e.g., Kyber, Dilithium, and Falcon). There are many ways to reduce the quantum complexity of lattice-based cryptography. However, practical quantum attacks on lattice-based cryptography remain under-researched compared to block cipher [2,3,4,5,6].

As mentioned earlier, analyzing potential quantum attacks on various cryptographic algorithms is crucial for establishing robust post-quantum security. In this context, our work proposes a quantum implementation of the NV Sieve algorithm to address the Shortest Vector Problem (SVP) in lattice-based cryptography. Moreover, we present an implementation that considers the optimization of quantum resources, and we estimate the quantum cost of Grover's search for our quantum NV Sieve approach.

1.1 Our Contribution

This paper makes several contributions, which can be summarized as follows.

1. **Applying Grover's Search to Quantum NV Sieve Algorithm on Lattice-based Cryptography.**
   This work firstly presents the quantum NV Sieve algorithm for solving the SVP on quantum computers. By applying Grover's algorithm to the iterative search process in the NV sieve, solutions that satisfy the condition can be found with a quantum advantage (a speedup of square root).
   For practical utility as an exact algorithm for solving SVP, it is necessary to target lattices with dimensions greater than 50. Thus, we implement the quantum NV Sieve algorithm, focusing on lattices with dimensions and ranks greater than 50.

2. **Detailed Resource Estimation of Quantum NV Sieve.**
   Quantum NV Sieve can have multiple solutions in the search process. In this case, an optimal number of Grover iterations is required, and the quantum

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

cost is affected by the number of iterations. Considering this, we provide a detailed estimation of the quantum resources required for the quantum NV sieve when multiple solutions exist[8].

3. **Optimized Implementation of Quantum NV Sieve for High-dimensional Lattice.**
We attempt quantum circuit optimization to obtain an oracle that requires fewer quantum resources. We efficiently implement the quantum NV Sieve logic for high-dimensional lattices by applying the Quantum Carry Save Adder (QCSA) with the Takahashi adder, thereby optimizing the number of qubits and T-depth. In particular, there is a significant decrease in terms of T-depth compared to previous work [7].

4. **Expanding the Research Scope of Cryptanalysis for Lattice-based Cryptography.**
We expand the scope of research by applying Grover's search, rather than the commonly used quantum walks, as a cryptanalysis approach to lattice-based cryptography.

1.2 Organization

The remainder of this paper is organized as follows. In Section 2, related works, such as lattice-based cryptography and Shortest Vector problem (SVP), are covered. In Section 3, the quantum implementation for NV Sieve to solve SVP is introduced. In Section 4, the resource estimation, quantum cost for Grover's search and further discussion of our implementation are provided. Finally, Section 5 concludes our paper.

1.3 Extended Version of ICISC'23

In this paper, we extend our previous work published in ICISC'23 [8]. In ICISC'23, small rank and dimension of the lattice were targeted, and an optimal implementation was not applied. On the other hand, this work targets real-level lattice where NV Sieve algorithm is used, and we achieve low quantum resources through an optimal quantum implementation.

2 Preliminaries

2.1 Lattice-Based Cryptography

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

[8] Detailed resource estimation while varying the parameters (e.g. iteration, γ, etc.) of the NV sieve remains for our future work.

vector (e.g. $x, -x \in L$). Equation 1 represents a lattice, and x is an integer, 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 (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) that constitutes the basis vector has a length of m and consists of a total of components n. The length of each vector and the number of vectors that make up the basis vector, respectively, are called Dimension (m) and Rank (n). Generally, a full-rank lattice is used (i.e., $m = n$).

It is important to note that the basis vector comprising a single lattice is not unique. As depicted in Figure 1, basis vectors corresponding to the same lattice points differ within a lattice. When a lattice is constructed using a vector generated by the multiplication of one basis vector with another, the two distinct basis vectors give rise to an identical lattice.

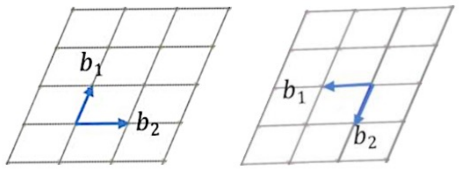


Fig. 1: Two different bases generating the same lattice.

However, these basis vectors can be categorized into good basis and bad basis. A good basis typically comprises short vectors, while a bad basis is often created by repeatedly multiplying the good basis by a matrix, such as an unimodular matrix[9]. Consequently, deriving a bad basis from a good basis is relatively straightforward, requiring only several matrix multiplications. In contrast, the inverse process of extracting a good basis from a bad one poses a significant challenge[10]. In lattice-based cryptography, the bad basis serves as the public key, while the good basis acts as the private key. Since they generate the same lattice, constructing public and private keys in this manner significantly complicates decryption in lattice-based cryptography.

9 https://en.wikipedia.org/wiki/Unimodular_matrix

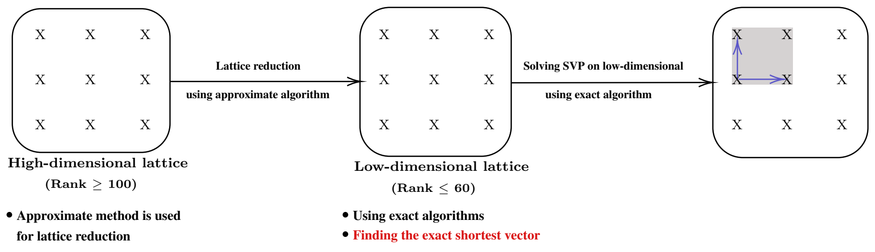
10 This is similar to generating a public key from a private key in public key cryptography (i.e., obtaining a private key by factorizing a large public key).

2.2 Shortest Vector Problem (SVP)

The Shortest Vector Problem (SVP), fundamental to lattice-based cryptography, involves finding the shortest nonzero vector within a lattice. Miklo's Ajtai [9] demonstrates that SVP is an NP-hard problem.

SVP finds the shortest vector using the lattice vector as input. However, the solution is not uniquely determined, since a vector can have another vector of equal magnitude. The challenge of solving SVP becomes difficult when a bad basis vector is input. With a good basis as input, there is a higher likelihood that the shortest vector is already present within the input basis. On the contrary, the use of a bad basis leads to the opposite result. The problem becomes increasingly complex with the growing rank of the lattice, defined by the number of constituent vectors.

Lattice-based cryptography typically involves lattices with ranks of 500 or higher, making its solution extremely difficult. Additionally, as noted earlier, deriving a good basis (i.e., private key) from a bad basis (i.e., public key) is challenging due to information asymmetry. In essence, the intricacy of lattice-based cryptography is primarily attributed to its reliance on one-way processes - easy in one direction but difficult in the reverse. To compromise such cryptographic systems, one must solve the underlying lattice problems. In short, by solving SVP, a lattice problem, lattice-based cryptographic schemes like LWE are threatened.



Lattice reduction
using approximate algorithm
Solving SVP on low-dimensional
using exact algorithm
High-dimensional lattice
(Rank ≥ 100)
• Approximate method is used for lattice reduction
Low-dimensional lattice
(Rank ≤ 60)
• Using exact algorithms
• Finding the exact shortest vector

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

Algorithms to Solve SVP To solve the lattice problems, approximate and exact algorithms must be used together (see Figure 2). In that, after the approximate algorithm is applied, to solve SVP, an exact algorithm is needed and important to find the shortest vector in the low-dimensional lattice. Approximate algorithms that reduce the high-dimensional to the low-dimensional lattice (e.g. Lenstra, Lenstra and Lovász (LLL) [10], block Korkine-Zolotarev (BKZ) [11]) have been widely studied. Also, sieve algorithms, such as AKS [12] and NV Sieve [13], have been proposed to solve SVP, which underpin lattice-based cryptography. These exact algorithms generally target low-dimensional lattices with a rank of about $50 \sim 60$.

2.3 Survey on the Exact Algorithms for SVP

Prominent exact algorithms in the field include AKS and NV Sieve. AKS, recognized as one of the earliest exact algorithms, suffers from drawbacks such as the numerous parameters and high time and space complexities. Therefore, AKS is largely impractical.

In response to the limitations of AKS, the NV Sieve algorithm was developed. It addresses the shortcomings of its predecessor by offering reduced time and space complexities, enhanced practicality, and the actual implementation. Building on the NV Sieve framework, several other Sieve algorithms have been introduced, as evidenced by studies such as Wang et al. [14], Zhang et al. [15], Laarhoven et al. [16], Becker et al. [17], and Micciancio et al. [18].

To date, only the theoretical complexity of applying the Sieve algorithm on quantum computers using Grover's search has been calculated [19]. Practical implementations of these quantum adaptations are still lacking. Therefore, in this paper, we implement the quantum NV Sieve, and discuss the quantum advantages that arise from applying Grover's search.

2.4 Classical NV Sieve Algorithm

Overview of the Classical NV Sieve Algorithm Algorithm 1 briefly shows the process of NV Sieve[11].

Algorithm 1: NV Sieve algorithm for finding short lattice vectors

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

```
 1: for $i = 1$ to $N$ do
 2:     $S \leftarrow$ Sampling $B$ using sampling algorithm
 3: end for
 4: Remove all zero vectors from $S$.
 5: $S_0 \leftarrow S$
 6: Repeat
 7:     $S_0 \leftarrow S$
 8:     $S \leftarrow$ latticesieve($S, \gamma R$) using Algorithm 2.
 9:     Remove all zero vectors from $S$.
10: until $S$ becomes an empty set.
11: return $v_0 \in S_0$ such that $||v_0|| =$min$||v||, v \in S_0$
```

First, a set S is generated by randomly sampling the basis received as input. Next, 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

[11] In our research, the NV Sieve algorithm is chosen as the exact algorithm to address the SVP in efficiency.

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. The purpose of NV Sieve is as follows:

- **Minimizing the loss for short vectors:** The goal of NV Sieve is to find the shortest vector that excludes 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. For this, a point on the lattice called c is randomly selected. Then, an additional computation is performed using c. c is a sufficient number of points on the lattice belonging to $\gamma R < x < R$[12].

- **Reducing the search range (γR):** The range 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.

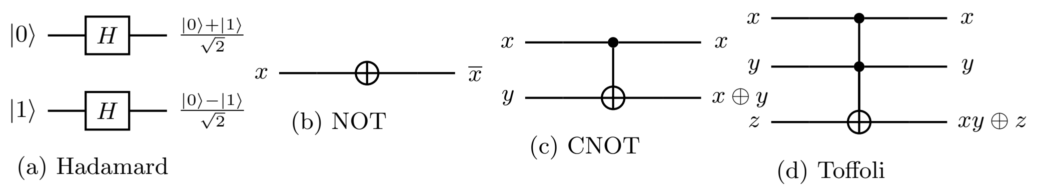


Fig. 3: Quantum gates.

2.5 Quantum Circuit

Qubits A qubit is the basic unit of computation in a quantum computer and can have probabilities of 0 and 1 in the superposition state ($|\psi\rangle$). This attribute allows k qubits to represent 2^k states. While qubits inherently exist in a state of superposition and are processed accordingly, they collapse to a singular classical value upon measurement. Additionally, multiple qubits are affected by each other through entanglement, and quantum computing utilizes this characteristic.

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 3). Each gate can be used to configure superposition, entanglement, and inversion. Therefore, these gates are instrumental in executing a range of computational tasks, including addition and multiplication on quantum circuits.

[12] 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 is determined by the corresponding sieve factor.

2.6 Grover's Search Algorithm

Grover's search algorithm is a quantum search algorithm for tasks with k-bit complexity and has $O(\sqrt{2^k})$ complexity ($O(2^k)$ for classical computer). The k-bit data for the target of the search must exist in a state of quantum superposition[13], given by:

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

Grover's search algorithm is composed of two main modules (Oracle and Diffusion operator):

1. Oracle is a quantum circuit designed to implement the logic necessary to return a solution to the problem at hand. It achieves this by inverting the decision qubit at the circuit's conclusion as follows. The crucial aspect of Grover's search with low cost lies in the optimal implementation of the quantum circuit that constitutes Oracle.

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

2. Diffusion operator serves to amplify the probability of the solution returned by the Oracle. By repeating this process, the observing the correct solution is increased, referred to as Grover iteration. However, it is often omitted from resource estimations [5,20], as its overhead is considered minimal and therefore negligible.

3 Quantum NV Sieve for Solving SVP

3.1 System Overview

In this section, we briefly describe proposed methods. Figure 4 illustrates the overview of the system of our quantum NV Sieve. As mentioned earlier (Section 2.4), the NV Sieve minimizes vector loss by searching for multiple vectors and identifying short vectors within a specified range, serving as a lattice reduction method. We apply Grover algorithm to the process that searches for random vectors c in the lattice to reduce search complexity. In order to implement the NV Sieve on Grover's search, a quantum oracle is required, and we propose an optimal implementation for it along with the quantum cost.

3.2 Quantum Implementation of NV Sieve's Core Logic

Target Core Logic of the NV Sieve. Algorithm 2 shows the `latticesieve` algorithm in NV Sieve. To find short vectors by reducing the size of the lattice,

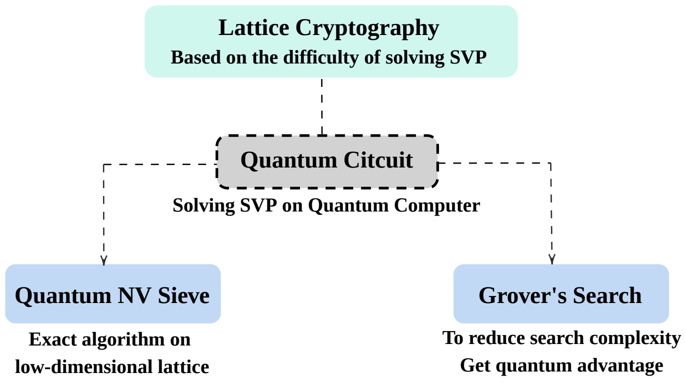

Lattice Cryptography
Based on the difficulty of solving SVP
Quantum Citcuit
Solving SVP on Quantum Computer
Quantum NV Sieve
Exact algorithm on low-dimensional lattice
Grover's Search
To reduce search complexity
Get quantum advantage

Fig. 4: System overview for quantum NV Sieve.

the core logic to find the vector c is implemented as a quantum circuit (see Figure 5).

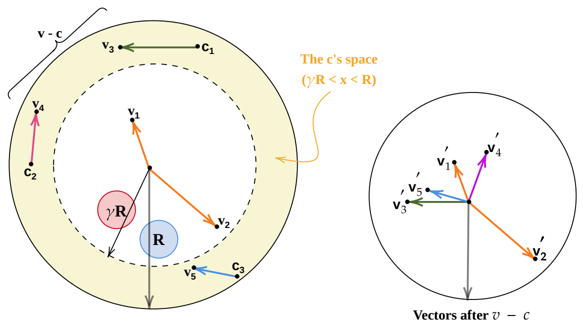


Fig. 5: The core logic in NV Sieve ($\exists c \in C||v - c|| \leq \gamma R$).

1. First, initialize C and S'. Afterward, the vectors with a length shorter than γR are stored in S'

[13] Thanks to quantum advantage, all targets are computed simultaneously.

Algorithm 2: The `latticesieve` algorithm in NV Sieve

```
Input: A subset $S$ in $L$ and sieve factor $\gamma$ ($0.666 < \gamma < 1$)
Output: $S'$ (Short 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 $S'$
```

2. However, there will be vectors longer than γR. For this, the process as in line 7 of Algorithm 2 is performed to minimize the 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.
3. Finally, by returning S', some shorter vectors than γR are selected. By performing this process repeatedly, sufficiently short vectors are obtained, and the shortest vector among them is found.
4. In summary, the core logic of the NV Sieve, which is our target operation for the oracle, is $\exists c \in C||v - c|| \leq \gamma R$.

3.3 Quantum Implementation of NV Sieve

This section describes step-by-step the quantum implementation of NV Sieve. Algorithm 3 and Figure 6 show all the steps of the quantum NV Sieve. We apply Grover's search to find a random vector c. After subtracting between the two vectors v and c, check whether the vector exists within the reduced lattice range. For this, two's complement is required, and it is implemented keeping in mind that conditional statements cannot be used in quantum computers.

1. **Initialization:** We set dim to dimension+2, and $rank$ to rank. $sqr\_bitsize$ equals $2 \cdot dim$. Also, $carry$, $qflag$, $cflag$, $zero$, and $\gamma \cdot R_{sqr}$ is initialized.

2. **STEP 1. Input settings:** Q_v and Q_c are the qubits for the lattice vectors v and c. H gates are applied to all the quantum registers (Q_c) for c. Also, $\gamma \cdot R_{sqr}$ is allocated to $Q_{\gamma \cdot R_{sqr}}$. It is used to check that the output vector can satisfy the condition. By using squared values γR^2, we do not need the square-root operation to calculate the vector size.

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

```
Input: Reduced lattice vector($\{v_0, ..., v_n\}$), dimension and rank of the lattice, A subset $S$ in $L$ and sieve factor $\gamma$ ($\frac{2}{3} < \gamma < 1$)
Output: $\{c_0, ..., c_n\}$
 1: Initiate quantum and classical registers ($carry, qflag, cflag, \gamma \cdot R_{sqr}$, $zero$, $c_n$.)
 2: Let $Q_v$ and $Q_c$ be the qubits for lattice vectors $v$ and $c$   ▷ $Q_c$ is the search target
 3: Let $Q_{\gamma \cdot R_{sqr}}$ be the qubits for square of $\gamma \cdot R$

 4: // STEP 0: Extend $dim$ and $rank$ for addressing the overflow
 5: $dim \leftarrow$ dimension+2, $rank \leftarrow$ rank
 6: $sqr\_bitsize \leftarrow 2 \cdot dim$

 7: // STEP 1: Input setting ($Q_v, Q_c, Q_{\gamma \cdot R_{sqr}}$)
 8: $Q_{v_0}, \cdots, Q_{v_n} \leftarrow v_0, \cdots, v_n$   ▷ Using X gate
 9: All(H)$|Q_c$
10: $Q_{\gamma \cdot R_{sqr}} \leftarrow \gamma \cdot R_{sqr}$   ▷ $0 \le i < sqr\_bitsize$

11: // STEP 2: Upscaling to address overflow
12: for $i$ in $rank$ do
13:     UPSCALING($Q_v[i], vflag[i]$)
14: end for

15: // STEP 3: Two's complement for subtraction using adder
16: for $i$ in $rank$ do
17:     COMPLEMENT_pos($Q_c[i]$, $qflag[i], zero$)   ▷ Outputs are $\overline{Q_c}$
18: end for

19: // STEP 4: $Q_v + \overline{Q_c}$ ($= Q_v - Q_c$)
20: for $i$ in $rank$ do
21:     TAKAHASHI_ADDER($Q_v[i], Q_c[i]$)   ▷ Store to $Q_c[i]$, [21]
22: end for

23: // STEP 5: Two's complement for correct squaring
24: for $i$ in $rank$ do
25:     COMPLEMENT_neg($Q_c[i], qflag[rank + i], zero$)
26: end for

27: // STEP 6: Duplicating qubit for squaring
28: $Dup\_Q_c \leftarrow Q_c$

29: // STEP 7: Squaring elements of vectors ($Q_c$ and $Dup\_Q_c$)
30: for $i$ in $rank$ do
31:     $output[i]$ = MUL($Q_c[i], Dup\_Q_c[i]$)
32: end for

33: // STEP 8: Addition for squared results to obtain the size of the vector
34: $result \leftarrow$ QCSA($output$)   ▷ [22]

35: // STEP 9: Two's complement with $sqr\_bitsize$
36: COMPLEMENT_compare($result[0 : sqr\_bitsize]$, $qflag[2 \cdot (rank - 2)]$, $zero$)

37: // STEP 10: Size comparison between $Q_{\gamma \cdot R_{sqr}}$ and $(||Q_v - Q_c||)^2$
38: TAKAHASHI_ADDER($Q_{\gamma \cdot R_{sqr}}$, $result[0 : sqr\_bitsize]$)

39: // STEP 11: Measurement
40: All(Measure)$|Q_c$

41: return $\{c_0, ..., c_n\}$
```

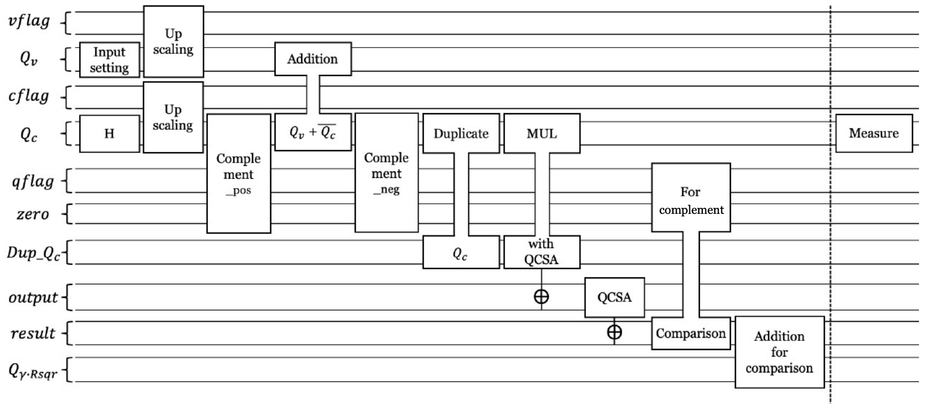


Fig. 6: Overall Quantum Circuit for Quantum NV Sieve.

3. **STEP 2. Upscaling:** Algorithm 4 shows the upscaling process in the quantum circuit. In the upscaling step, for Q_v and Q_c, we duplicate the upper data bit into the two empty upper qubits. This approach expands the range while preserving the data and its sign intact. For example, a value $1100_{(2)}$ becomes $111100_{(2)}$, maintaining the same value. To achieve this, we store the value of the third-highest qubit (the MSB of the data) into the $qflag$ and then replicate it into the top two qubits using a CNOT gate.

Algorithm 4: Quantum implementation for UPSCALING function

```
Input: ($Q_v[i]$, $vflag$) or ($Q_c[i]$, $cflag$), and $dim$
Output: $Q_v[i]$ or $Q_c[i]$
1: // Copy the MSB to the upper 2 qubits using CNOT gate
2: CNOT|($Q_v[i][dim-3], vflag[i]$)
3: CNOT|($vflag[i], Q_v[i][dim-2]$)
4: CNOT|($vflag[i], Q_v[i][dim-1]$)

5: CNOT|($Q_c[i][dim-3], cflag[i]$)
6: CNOT|($cflag[i], Q_c[i][dim-2]$)
7: CNOT|($cflag[i], Q_c[i][dim-1]$)

8: return $Q_v[i]$ or $Q_c[i]$
```

4. **STEP 3. Two's complement for positive value:**
Algorithm 5 shows the two's complement for a positive value. For the subtraction operation, we apply the complement operation to the operand before using an adder. In this step, the complement operation is performed

Algorithm 5: Quantum implementation for COMPLEMENT_pos function

Input:$Q_c[i]$, $qflag[i]$, $zero$
Output: Q_c or $\overline{Q_c}$

```
1: // Copy the MSB of $Q_c$ to $qflag$ to check the sign bit
2: CNOT($Q_c[dim-1], qflag$)

3: // Invert $qflag$ to take complement only when positive
4: X|$qflag$

5: // Invert $Q_c$
6: for $i$ in $dim$ do
7:     CNOT|($qflag, Q_c[i]$)
8: end for

9: // Create a new array of qubits and append $qflag$ to LSB
10: $NEW\_Q_c = []$
11: $NEW\_Q_c$.append($qflag$)

12: // Append 0 so that it has the same length as $Q_c$
13: for $i$ in $dim - 1$ do
14:     $NEW\_Q_c$.append($zero[i]$)
15: end for

16: // Addition for $LSB + 1$
17: TAKAHASHI_ADDER($NEW\_Q_c, Q_c$)

18: return $Q_c$ or $\overline{Q_c}$
```

only for positive vector elements. Therefore, it is crucial to identify the most significant bit. However, conditional statements cannot be used in quantum implementations. Thus, we utilize a $qflag$ to ensure that the complement operation is carried out only when the value is positive. After copying the value of the most significant qubit in $qflag$, we apply the X gate to it. If the target qubit is positive, the value of $qflag$ will be inverted to 1 in this process. Consequently, using $qflag$ as a control qubit allows the application of one's complement operation, effectively flipping the entire data (only for positive cases). To complete the two's complement operation, we add 1 to the Least Significant Bit (LSB). For this purpose, we create a new qubit (New_Q_c), to which we add $qflag$ to the lowest qubit and fill the rest with zeros (except for LSB). When this qubit is added to the qubits that are applied to one's complement, the two's complement is calculated. Additionally, we employ the Takahashi adder [21][14] to add the new qubit to the target qubit (one's complement applied).

5. **STEP 4. $Q_v - Q_c$:** In this step, the subtraction operation between the two vectors is performed by adding the complemented Q_c and Q_v. The Takahashi adder is also employed in this process to perform the addition.

6. **STEP 5. Two's complement for negative value:** STEP 5 is dedicated to applying the two's complement operation exclusively to negative numbers, for accurate square operations. For example, performing a square operation directly on $1110_{(2)}$ would result in a square of -2, which yields 14. However, by applying the two's complement first and then squaring, the square of $0010_{(2)}$ is 4. Therefore, we obtain the correct squared result. Algorithm 6 demonstrates the two's complement operation for negatives. It is largely similar to the operation for positives, with the key difference that the MSB is already 1 for negatives, thus omitting the application of the X gate on $qflag$.

7. **STEP 6. Duplication of squaring:** In quantum implementation, operations on the same qubit cannot be performed. So, the value of Q_c is stored in $Dup\_Q_c[i]$ for the square operation. For this, we use a CNOT gate, and the CNOT gate acts as a copying value of a controlled qubit into the empty (target) qubit.

8. **STEP 7. Squaring Q_c:** Figure 7 illustrates the squaring operation using Quantum CSA with Takahashi adder. First, a Toffoli gate is applied on two qubits assigned with identical values to multiply each element by one another. Subsequently, the results are stored in dim qubits, each of which has a length of $sqr\_bitsize$. According to line 8 of Algorithm 7, values are stored in the indices corresponding to each element. This process is repeated for all qubits and yields the arrays of dim qubits. Then, utilizing QCSA

[14] It is an in-place adder and requires fewer qubits than other adders [23,24].

Algorithm 6: Quantum implementation for COMPLEMENT_neg function

```
Input:$Q_c[i]$, $qflag[i]$, $zero$
Output: $Q_c$ or $\overline{Q_c}$
 1: // Copy the MSB of $Q_c$ to $qflag$ to check the sign bit
 2: CNOT($Q_c[dim-1], qflag$)

 3: // Invert $Q_c$ (no need X gate for $qflag$)
 4: for $i$ in $dim$ do
 5:     CNOT|($qflag, Q_c[i]$)
 6: end for

 7: // Create new array of qubits and append $qflag$ to LSB
 8: $NEW\_Q_c = []$
 9: $NEW\_Q_c$.append($qflag$)

10: // Append 0 so that it has the same length as $Q_c$
11: for $i$ in $dim-1$ do
12:     $NEW\_Q_c$.append($zero[i]$)
13: end for

14: // Addition for $LSB+1$
15: TAKAHASHI_ADDER($NEW\_Q_c, Q_c$)

16: return $Q_c$ or $\overline{Q_c}$
```

with the Takahashi adder applied, all results are summed at the same time. This results in an array of qubits with a length of *sqr_bitsize*. When this process is repeated for the *rank* of times, the squaring for all vector elements is completed.

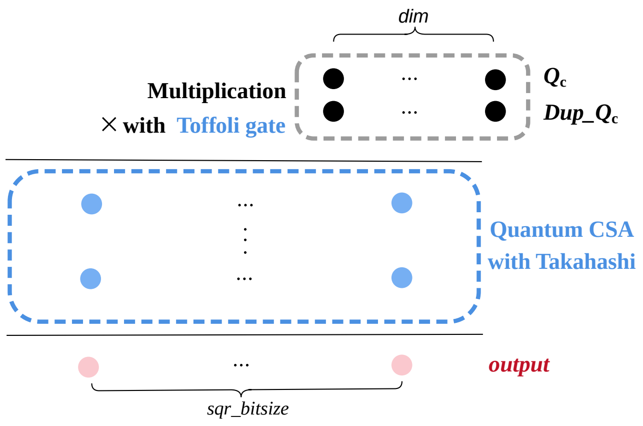


Fig. 7: Squaring using QCSA with Takahashi adder.

Algorithm 7: Quantum implementation for MUL function

```
Input: $Q_c[i], Dup\_Q_c[i]$
Output: $output[0 : sqr\_bitsize]$
 1: // Setting for multiplication
 2: Let $a$ be $Q_c[i]$.
 3: Let $b$ be $Dup\_Q_c[i]$.
 4: $input$ = [[0 for $i$ in do($sqr\_bitsize$)] for $j$ in do ($dim$)]

 5: // Multiply all elements of $Q_c[i]$ and $Dup\_Q_c[i]$ using Toffoli gate
 6: for $i$ in $dim$ do
 7:     for $j$ in $dim$ do
 8:         Toffoli$|(a[i], b[j], input[i][i+j])$
 9:     end for
10: end for

11: // Addition of the results for each element at once
12: $output$ = QCSA($input$)

13: return $output[0 : sqr\_bitsize]$
```

9. **STEP 8. Addition of squared results:** The square of the vector's magnitude is computed by summing the squared results of each element of the vector. For this purpose, the Quantum CSA is utilized to aggregate all the resultant values at a time.

10. **STEP 9. Two's complement:** The two's complement for positive value is calculated. Although the logic is identical to the Algorithm 5, a notable distinction is that the range is set to *sqr_bitsize*.

11. **STEP 10. Comparison of sizes ($Q_{\gamma \cdot R_{sqr}}$ and $(||Q_v - Q_c||)^2$):** Finally, the result of STEP 9, with the complement applied, is compared with $Q_{\gamma \cdot R_{sqr}}$. For comparison, the Takahashi adder is used to add the two values. If the MSB of the derived result is 0, it indicates that Q_c has been found, which can bring the vector Q_v into the range by making it a short vector. Conversely, if the MSB is 1, then it means that vector that meets the conditions does not exist.

4 Experiment and Evaluation

4.1 Experiment Environment

In this section, we present the quantum cost of our quantum NV Sieve implementation. For this, we use ProjectQ, which is an open-source quantum programming tool. It provides ClassicalSimulator which simulates quantum circuits and ResourceCounter which estimates the quantum resources (e.g. qubits, gates, etc.).

4.2 Result of Quantum NV Sieve

Table 1 shows the results of each step of our implementation for the quantum NV Sieve. STEP is in Algorithm 3, and the expression of the complement of x is $\overline{x}$. We set the rank and dimension to 5 as a toy example to show the correctness of our oracle. So *rank* remains 5 and *dim* becomes 7 to handle the overflow. In STEP 1, the quantum registers set the lattice vectors (i.e., test vector). In STEP 2, the vectors are upscaled, but the value is the same as in STEP 1. In STEP 3, we perform the complement operation of Q_c when the vector element is positive (no operations are performed on Q_v and $Q_{\gamma \cdot R_{sqr}}$). In STEP 4, the subtraction is performed on Q_v and Q_c ($Q_v + \overline{Q_c}$). In STEP 5, the complement operation with negative elements is used. In STEP 7, the squaring operation is performed for each element. In STEP 8, we calculate the squares of the vector sizes. In STEP 10 and 11, we get the result of quantum NV Sieve with the above test vector. It is -40 in decimal and $11111111011000_{(2)}$ in the quantum state (measured). The two values are the same, and this means that our oracle can derive the correct vector. Also, the MSB being 1 signifies that the result vector is not the short vector (if we use another test vector, MSB as 0 can be derived).

Table 2: Resource Estimation of quantum NV Sieve oracle (`R10D10` means the rank and the dimension of the lattice are 10).

Case	#CNOT	#1qCliff	#T	T-depth (Td)	Full depth (FD)	Qubit (M)	Td-M	FD-M
`R10D10`	$2^{16.1767}$	$2^{13.9067}$	$2^{15.7118}$	$2^{7.6037}$	$2^{11.5264}$	$2^{12.5454}$	$2^{20.1491}$	$2^{24.0718}$
`R20D20`	$2^{18.9097}$	$2^{16.6143}$	$2^{18.4212}$	$2^{8.4470}$	$2^{14.2190}$	$2^{15.2640}$	$2^{23.7110}$	$2^{29.4830}$
`R30D30`	$2^{20.5672}$	$2^{18.2624}$	$2^{20.0695}$	$2^{8.9454}$	$2^{15.2810}$	$2^{16.9202}$	$2^{25.8656}$	$2^{32.2012}$
`R40D40`	$2^{21.7859}$	$2^{19.4808}$	$2^{21.2880}$	$2^{9.3531}$	$2^{16.0566}$	$2^{18.1329}$	$2^{27.4860}$	$2^{34.1896}$
`R50D50`	$2^{22.6998}$	$2^{20.3885}$	$2^{22.1958}$	$2^{9.6165}$	$2^{16.6674}$	$2^{19.0527}$	$2^{28.6692}$	$2^{35.7201}$
`R60D60`	$2^{23.4836}$	$2^{21.1729}$	$2^{22.9802}$	$2^{9.8595}$	$2^{17.1715}$	$2^{19.8329}$	$2^{29.6924}$	$2^{37.0044}$
`R70D70`	$2^{24.1348}$	$2^{21.8228}$	$2^{23.6301}$	$2^{10.0660}$	$2^{17.6005}$	$2^{20.4848}$	$2^{30.5508}$	$2^{38.0853}$

Table 1: Results from each step of quantum NV Sieve to check whether it has been implemented correctly (Rank and Dimension are 5; $rank = 5$, $dim = 7$, and $sqr\_bitsize = 14$).

STEP (Alg. 3)	Quantum variable	Values
STEP 1	Q_v	$\{1, 3, 1, 1, 5\}$
	Q_c	$\{8, 1, 4, 4, 6\}$
	$Q_{\gamma \cdot R_{sqr}}$	32
STEP 2	Q_v	$\{1, 3, 1, 1, 5\}$
	Q_c	$\{8, 1, 4, 4, 6\}$
	$Q_{\gamma \cdot R_{sqr}}$	32
STEP 3	$\overline{Q_c}$ (when positive)	$\{-8, -1, -4, -4, -6\}$
STEP 4	$Q_v + \overline{Q_c}$	$\{-7, 2, -3, -3, -1\}$
STEP 5	$\overline{Q_v + \overline{Q_c}}$ (when negative)	$\{7, 2, 3, 3, 1\}$
STEP 6	Dup_Q_c	$\{7, 2, 3, 3, 1\}$
STEP 7	$\overline{Q_c} \cdot Dup\_Q_c$(Squaring for each element)	$\{49, 4, 9, 9, 1\}$
STEP 8	Sum_{Q_c}	72
STEP 9	$\overline{Sum_{Q_c}}$ (when positive)	-72
STEP 10	$(\gamma \cdot R)^2 + \overline{Sum_{Q_c}}$	-40
STEP 11	**Output**	$11111111011000_{(2)}$ (-40)
	MSB	1 (not short vector)

Table 3: Quantum cost for Grover's search on quantum NV Sieve.

Case	#Total gates	T-depth (Td)	Full depth (FD)	Qubit (M)	Quantum cost	Td-M	FD-M
R10D10	$2^{18.1267}$	$2^{8.6073}$	$2^{12.5264}$	$2^{12.5456}$	$2^{30.6532} \cdot r$	$2^{21.1529}$	$2^{25.0720}$
R20D20	$2^{20.8481}$	$2^{9.4470}$	$2^{15.2190}$	$2^{15.2640}$	$2^{35.0664} \cdot r$	$2^{24.7110}$	$2^{32.4830}$
R30D30	$2^{22.5012}$	$2^{9.9454}$	$2^{16.2810}$	$2^{16.9202}$	$2^{37.7823} \cdot r$	$2^{26.8656}$	$2^{33.2012}$
R40D40	$2^{23.7199}$	$2^{10.3531}$	$2^{17.0566}$	$2^{18.1329}$	$2^{39.7765} \cdot r$	$2^{28.4860}$	$2^{35.1895}$
R50D50	$2^{24.6308}$	$2^{10.6165}$	$2^{17.6674}$	$2^{19.0527}$	$2^{41.2938} \cdot r$	$2^{29.6692}$	$2^{36.7201}$
R60D60	$2^{25.4150}$	$2^{10.8595}$	$2^{18.1715}$	$2^{19.8329}$	$2^{42.5865} \cdot r$	$2^{30.6924}$	$2^{38.0044}$
R70D70	$2^{26.0655}$	$2^{11.0660}$	$2^{18.6005}$	$2^{20.4848}$	$2^{43.6661} \cdot r$	$2^{31.5509}$	$2^{39.0853}$

4.3 Resource Estimation of Quantum NV Sieve

Table 2 shows the resources of the quantum NV Sieve oracle according to the case (rank and dimension of the lattice). From R10D10 to R20D20, the quantum resources increase the most. This is a natural phenomenon because it is a section where the rank and dimension are doubled. Therefore, each time rank and dimension increase by 10, and the amount of increased resources decreases. This can be seen more clearly in quantum costs, which we will see in the next section.

The NV Sieve algorithm, which we target as an exact algorithm, is meaningful when it involves at least 50 ranks and dimensions. In general, the ranks and dimensions typically used in cryptographic algorithms are over 500. In such large-sized lattices, approximate algorithms (e.g., LLL, BKZ) are employed to reduce the lattice size. Typically, these reductions bring the dimension down to around 50-60. Therefore, our implementation must target lattices with dimensions of at least 50. Consequently, in Table 2, the important part to observe is from R50D50 to R70D70.

The parts with the high complexity of our oracle are STEP 7 and STEP 8 in Algorithm 3, where the square operation is performed with the Toffoli gate. The Toffoli gate is used only in the squaring and in the part that calculates the square of the size of the vector. Therefore, we estimate the T-depth in STEP 7 and STEP 8. In particular, in the part where the multiplication of each qubit for the square operation is performed, it can be seen that the T-depth consistently increases by 70 as the rank and the dimension are expanded by 10. As the dimension and the rank increase, the number of target qubits to calculate also increases regularly. Therefore, the T-depth increases regularly in the multiplication part. Additionally, this part would be resource intensive if schoolbook multiplication was performed. However, we achieve a low T-depth using Quantum CSA [15], despite the high dimension and the rank of the lattice. Additionally, we further reduce T-depth by not using the reverse part in QCSA for qubit reuse. This is because the operand in the square operation process is not a value that is used again in our implementation.

Our previous work [7] shows T-depth of 1,756 in R2D4. This means that our implementation achieves less T-depth even with much higher dimensions and ranks, while our previous work has a high T-depth even with small dimensions.

[15] The T-depth of our Toffoli gate is 4.

That is, our implementation is optimized in terms of T-depth (T-depth of this work: $2^{10.0660}$ in `R70D70`, T-depth of [7]: $2^{10.7780}$ in `R2D4`). In addition, to save the number of qubits used, the Takahashi adder is used, which does not require ancilla qubits. This reduces the value of Td-M and FD-M.

4.4 Quantum Cost of Quantum NV Sieve

Figure 8 and Table 3 show the quantum cost of our quantum NV Sieve on Grover's search. While calculating the quantum cost [25] of Grover's search, the total number of gates ($\#gates$) and Full depth (FD) mentioned in Table 2 must be multiplied by iteration (i.e., $\#gates \cdot FD \cdot r$).

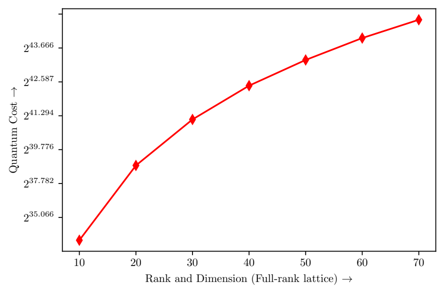


Fig. 8: Quantum cost for quantum NV Sieve on Grover's search ($r = 1$)

Figure 8 shows the quantum cost and the increase according to rank and dimension. As shown in Table 3, when rank and dimension increase, the increment in the number of quantum resources decreases. In addition, it usually targets a lattice with rank and dimension of about $50 \sim 60$ and can be attacked at a quantum cost of about 2^{42}.

Even if the number of iterations increases, the number of iterations in a multi-solution problem is not that large, and so it is not expected to deviate significantly from the cost of the current attack.

Furthermore, research on LLL which is one of the approximate algorithms, is conducted on quantum simulator. If applied to a lattice reduced through their technology, it is believed that our implementation for solving SVP will be more realistic and operate more powerfully.

4.5 Further Discussions

Comparison with Quantum Cost of Other Quantum Attacks A direct comparison is challenging since the target problem is different from the symmetric key problem, and the target lattice is not the entire lattice. Taking into account this fact, the quantum NV Sieve can be a quantum attack that solves the fundamental problem of lattice-based cryptography with a quantum cost of approximately 2^{43}. This is approximately the square root of the cost of Grover's search for a 64-bit symmetric key cipher.

Additional Optimizing Point of the Quantum Implementation It is believed that by employing a Toffoli gate with a lower T-depth, our T-depth can be further reduced. Additionally, while using the Draper adder in QCSA, although it may increase the number of qubits required, it is expected to yield a lower circuit T-depth.

Considerations for the Complexity of NV Sieve The factors that affect the complexity of NV Sieve's algorithm are as follows.

1. **Multiple solutions** In our approach, unlike key search using Grover's search, multiple solutions may exist[16]. For the multi-solution scenario in Grover's algorithm, the number of iterations ($r = \lfloor \frac{\pi}{4} \cdot \sqrt{\frac{N}{M}} \rfloor$) required differs from that of the one-solution Grover's search (N is a search space, M is the number of solutions, [26,27]). In this work, we primarily focus on presenting a correct and optimized oracle and estimating the cost associated with Grover's search.

2. **Search space (N):** It is decided by rank and dimension ($N = 2^{rank \cdot dimension}$). As N increases, the difficulty of solving the NV Sieve increases. It is related to the number of points in c. There is a sufficiently large number of points (e.g. v, c) on the lattice. Among this, we need to find a point c that can be used to create a vector with a length shorter than γR, and it is important to find the number of c. Because it is difficult to find c in a high-rank lattice, the size of the set of target basis vectors affects the complexity of the algorithm.

3. **Length (R) of the longest vector among input vectors (v):** The input vectors fall within the range $\gamma < v < R$. Consequently, the maximum length (R) in the input vector set is determined (e.g., rank and dimension are 2, then R is $2\sqrt{2}$). The complexity will vary depending on the condition of the value of R which determines the search space.

Quantum Query Complexity In noted above, the quantum query complexity of the NV Sieve algorithm varies with several parameters, such as N, R r, γ, etc. However, if the oracle is appropriately implemented, and the optimal number of iterations is determined, it follows the theoretical complexity of Grover's search

[16] 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 that meet the condition in the quantum NV Sieve. Therefore, determining the correct Grover iteration is another important issue.

(see Equation **??**). In this scenario, the number of solutions, denoted as M, is significantly smaller than the entire search space N.

Consequently, the quantum query complexity is expected to be $O(\sqrt{N})$. From a query complexity perspective, this suggests a quantum advantage over classical computers.

5 Conclusion

In this work, we introduce the quantum NV Sieve algorithm as a solution to the Shortest Vector Problem (SVP), a pivotal challenge in lattice-based cryptography. Furthermore, we aim to achieve a lattice of $50 \sim 60$ dimensions, which are commonly used in algorithms to solve SVP. By applying Grover's search algorithm to the vector search component of the existing NV Sieve, our implementation achieves a significant advantage in search (i.e., query) complexity. Notably, for optimal implementation, we employ QCSA with Takahashi adder, achieving a substantial reduction in depth compared to our previous work. This means that our approach facilitates a more efficient quantum circuit, even in higher-dimensional lattices than our previous implementation.

This work shows that the required quantum complexity for the SVP on the lattice of rank 70 and dimension 70 is 2^{43} (a product of total gate count and total depth) with our optimized quantum implementation of the NV sieve algorithm. This complexity is significantly lower than the NIST post-quantum security standard, where level 1 is 2^{157}, corresponding to the complexity of Grover's key search for AES-128.

Ultimately, our work contributes to expanding the scope of research in the field of cryptanalysis by applying Grover's search to the cryptanalysis of lattice-based cryptography.

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, 8
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. H. Kim, K. Jang, Y. Oh, W. Seok, W. Lee, K. Bae, I. Sohn, and H. Seo, "Finding shortest vector using quantum NV Sieve on Grover," *Cryptology ePrint Archive*, 2023. 3, 19, 20

8. H. Kim, K. Jang, Y. Oh, W. Seok, W. Lee, K. Bae, I. Sohn, and H. Seo, "Finding shortest vector using quantum NV Sieve on Grover," in *International Conference on Information Security and Cryptology*, 2023. 3
9. 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. 5
10. P. Q. Nguyen and B. Vallée, *The LLL algorithm*. Springer, 2010. 5
11. 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
12. 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
13. 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
14. 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
15. 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
16. 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
17. 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
18. 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
19. 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. 6
20. 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. 8
21. Y. Takahashi, S. Tani, and N. Kunihiro, "Quantum addition circuits and unbounded fan-out," *arXiv preprint arXiv:0910.2530*, 2009. 11, 14
22. P. Gossett, "Quantum carry-save arithmetic," *arXiv preprint quant-ph/9808061*, 1998. 11
23. 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. 14
24. T. G. Draper, S. A. Kutin, E. M. Rains, and K. M. Svore, "A logarithmic-depth quantum carry-lookahead adder," *arXiv preprint quant-ph/0406142*, 2004. 14
25. NIST, "Call for additional digital signature schemes for the PQC standardization process." `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf`. 20

26. G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, “Quantum amplitude amplification and estimation,” *Contemporary Mathematics*, vol. 305, pp. 53–74, 2002. 21
27. M. Boyer, G. Brassard, P. Høyer, and A. Tapp, “Tight bounds on quantum searching,” *Fortschritte der Physik: Progress of Physics*, vol. 46, no. 4-5, pp. 493–505, 1998. 21