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]

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

2023/11/29 ICISC'23

Motivation and Contribution

- **Motivation**
  - 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**.
  - There is a solution search part of the **Sieve algorithm** to solve the Shortest Vector Problem.
  - **Let's apply Grover's search here and get the quantum advantage.**

- **Contribution**
  - **For the first time in our knowledge, Quantum NV Sieve implementation to solve SVP.**
  - **Extension implementation considering multiple conditions (dimension, rank) of lattice-based cryptography.**
  - **Resource estimation for Quantum NV Sieve logic and Grover's search.**

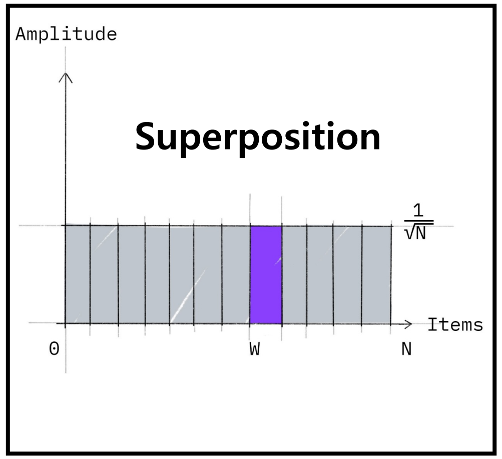
Background

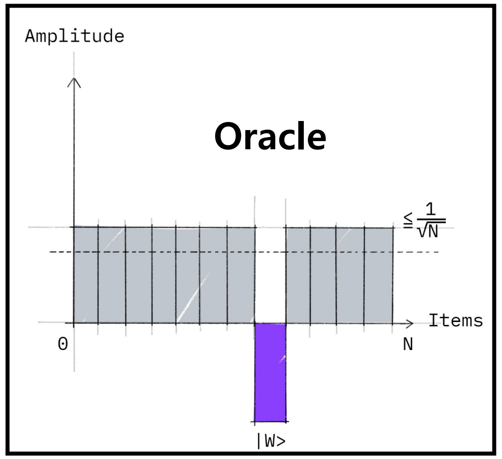
: Grover’s search, Lattice, SVP, Sieve algorithm

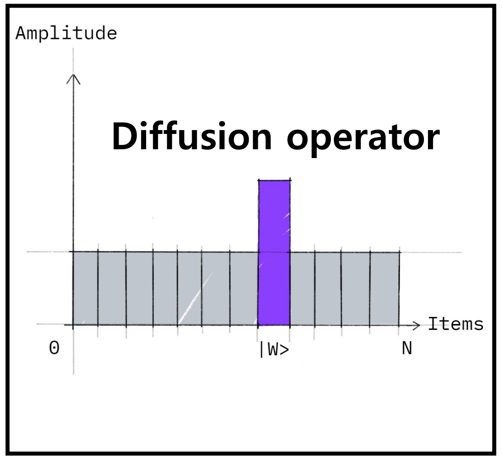
Grover's Search

- **Grover's search algorithm**
  - For a task with a complexity of k-bit, Classical and Grover's search can find a solution with the complexity: $O(2^k)$ **(Classical)** / $O(\sqrt{2^k})$ **(Quantum)**
  - The search target exists in a state of quantum superposition.

    → **A state in which all search targets exist simultaneously as a probability.**
  - Grover's search has two modules:
    1. **Oracle** : Return the solution (i.e. Logic for deriving solutions)
    2. **Diffusion operator** : Increases the probability of the returned solution.
  - **Increases the probability of a solution through repetition of Oracle and Diffusion operator ($\sqrt{2^k}$)**







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

Lattice and Basis

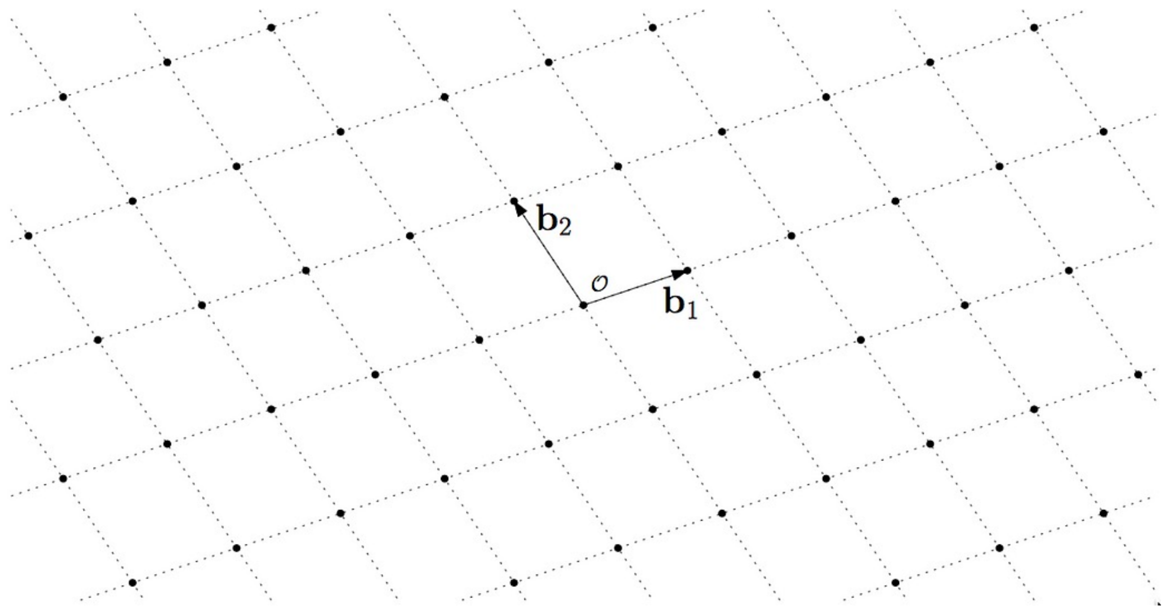
- **Lattice** (L) : a set of points made up of a **linear combination of basis vectors** $(\boldsymbol{B})$.
- $\boldsymbol{x}$ is an integer, and $(\boldsymbol{b_1}, \ldots, \boldsymbol{bn})$ means the basis vector.

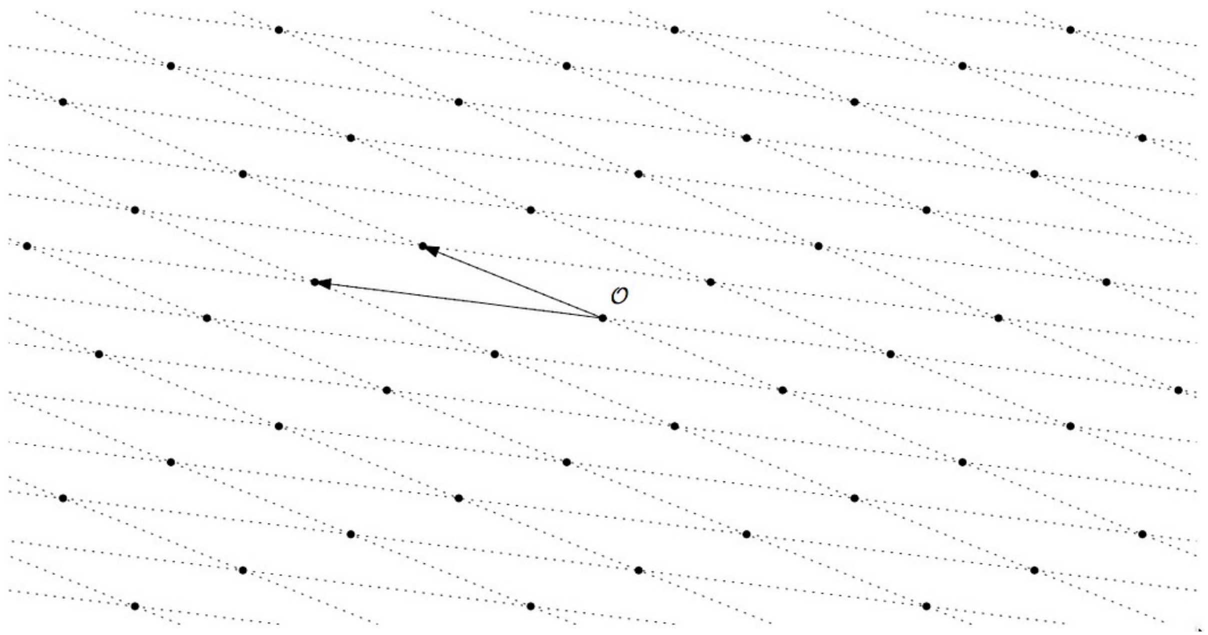
$$L(b_1, \ldots, b_n) = \Sigma_{i=1}^{n}(x_i \cdot b_i, x_i \in Z)$$

- **Lattice has two elements: Rank** (n) and **Dimension** (m)
  - **Rank** : the number of vectors constituting the basis vector
    - Rank= 3 → $(00_{(2)}, 10_{(2)}, 11_{(2)})$
  - **Dimension** : the length of each vector
    - Dimension = 4 → $(0000_{(2)}, 0100_{(2)}, 1100_{(2)})$
  - Generally, a full-rank lattice is used $(m = n)$.
  - **In Lattice-based Cryptography** : $m, n \geq 500$

Lattice and Basis

- There are several **basis** for representing a **lattice**.
  - The basis vectors on the same lattice are different.





Good and bad basis

Good basis vs **bad basis**

Generally composed of **a short vector**

Finding a good basis from a bad basis becomes a **very difficult task.**

Created **by multiplying the good basis by a matrix** such as an unimodular matrix several times.

Finding a bad basis from a good basis is **easy** because **only matrix multiplication several times** is required.

- **In lattice-based cryptography:**
  - **a bad basis** is used as **the public key**
  - **a good basis** is used as **the private key**
- The two 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.

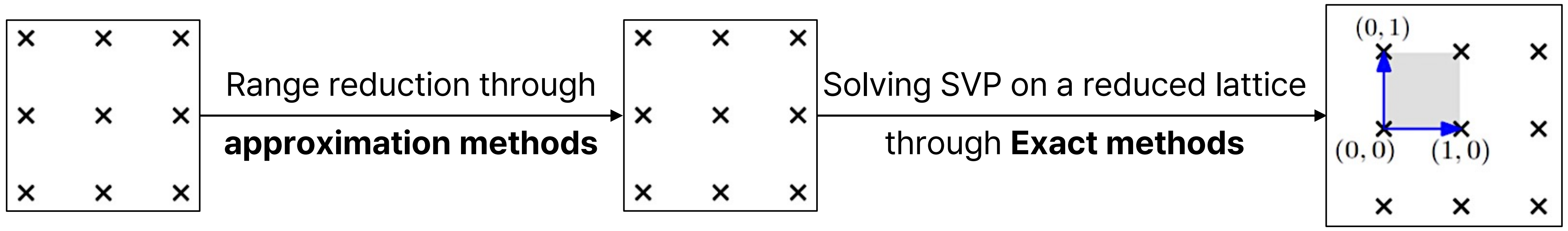
*https://en.wikipedia.org/wiki/Unimodular_matrix

Shortest Vector Problems (SVP)

- **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**.
- **Lattice-based cryptography is based on lattice problems (SVP, CVP, etc.)**
  - The security level of lattice-based cryptography is based on the difficulty of solving the lattice problem.
- **Solving SVP → lattice-based cryptosystems such as LWE can be threatened.**
- **Difficulty**
  - **If a bad basis vector is used as input**, **the difficulty of solving the SVP increases.** → Hard
  - **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. → **Easy**
  - Additionally, **as the rank (n) and dimension (m) of the lattice increases**, it becomes **more difficult to solve.**

Lattice cryptanalysis and Shortest Vector Problems

After reducing the range of lattice through approximation in a high dimension,
SVP is solved by performing the exact method with the reduced vectors as input.



High dimension
(Rank ≥ 100)

Filtering out vectors on
a high-dimensional lattice

Low dimension
(Rank ≤ 60)

Find exactly the shortest vector
using the exact algorithm

- **We have to solve SVP for Lattice cryptanalysis**
  - In high dimensions, use the Approximate method for range reduction
  - **In low dimension, use the Exact algorithm for finding the exact shortest vector**
  - The best practical/theoretical **SVP solution must be accurate and efficient in low dimensions**.
  - After accurately solving the SVP in the lower dimensions, **it is important to determine the highest dimension that can be solved.**

Survey on the exact algorithms for SVP

- Well-known exact algorithms include AKS [11] and NV Sieve [12].

- **AKS**
  - The most famous early exact algorithm
  - But it has the **disadvantage of using many parameters and having high time and space complexity.**
  - It is deemed an **impractical algorithm** (due to the absence of optimal parameters and actual implementation)

- **NV Sieve**
  - An exact algorithm, was introduced **to address these limitations of AKS.**
  - It offers **benefits such as reduced time / space complexity and practicality**
  - 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].

- Only **the theoretical complexity of the Sieve algorithm on quantum computers** (using Grover's search) has been calculated [7].

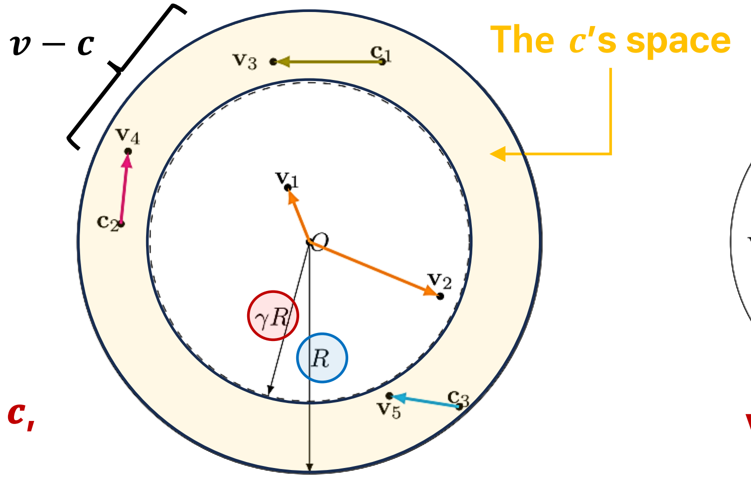
NV Sieve Logic

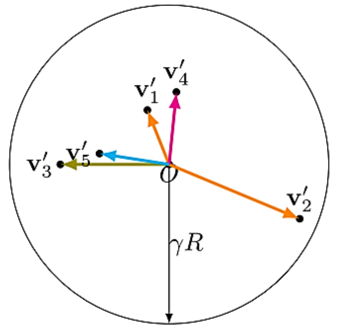
- **Purpose** : In order to avoid loss for short vectors, we **randomly select c** to reduce the range and obtain **vectors shorter than γR**.

- **Input** : Vector on the lattice with maximum length R

- **Output** : Vectors on the lattice shorter than γR

- **Logic of NV Sieve**
  1. Maximum length among vectors in $S \rightarrow R$
     Initiate C, S'
  2. **Vectors with a length shorter than γR are stored in S'.**
  3. **Vectors with a length longer than γR are subtracted from the point c,**
     - stored in S' if the length is shorter than γR
     - stored in C if the length is longer than γR.
  4. Return S'





Vectors after $v - c$

- That is, **when a short vector that satisfies the conditions is found**, it is stored in S' and then moved to the next input.

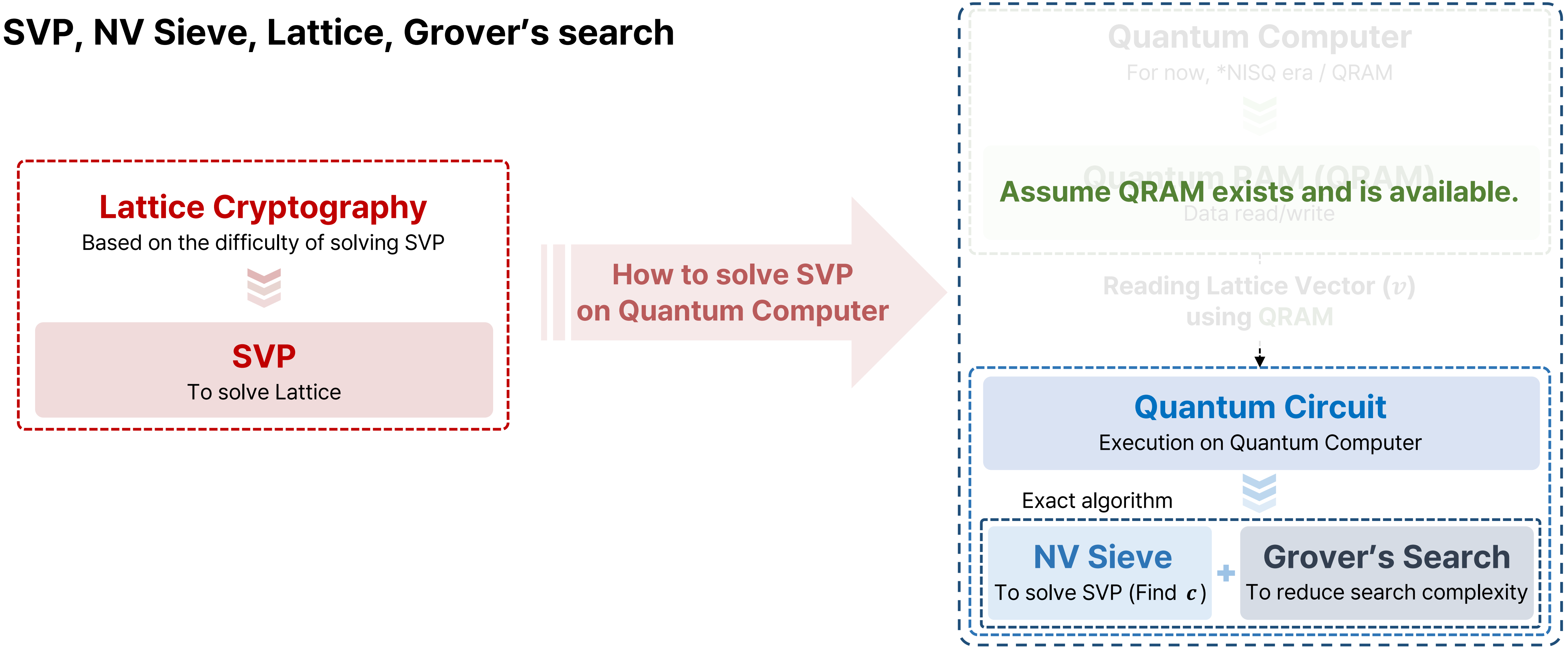
  → **Finding c has a significant impact on the complexity of the NV Sieve.**
  - We do not know **the many points that can be c**
  - We do not know **if any of them satisfy the conditions.**

Quantum NV Sieve

: Proposed method

Overview of Proposed Method

- **SVP, NV Sieve, Lattice, Grover's search**

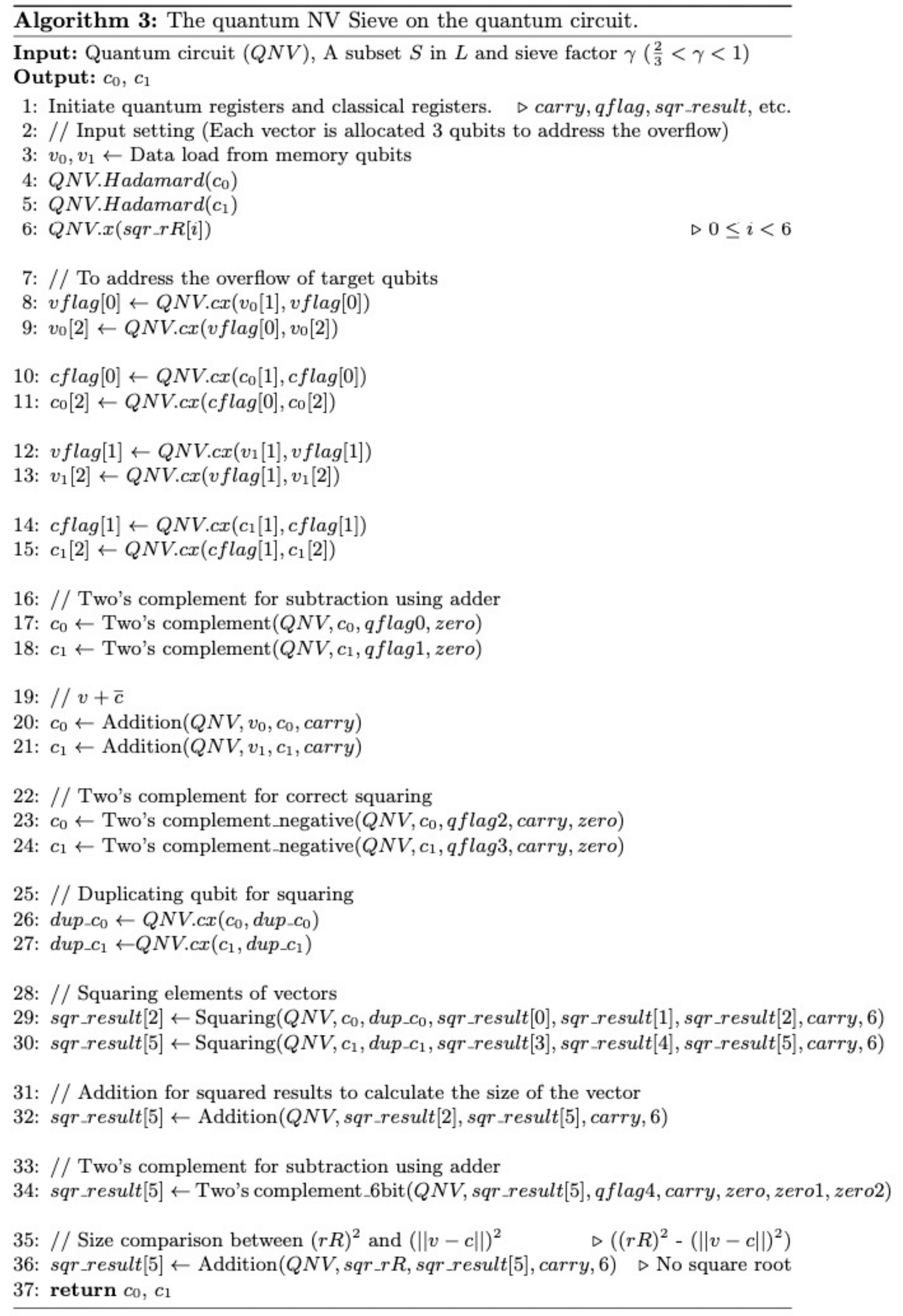

Lattice Cryptography
Based on the difficulty of solving SVP
SVP
To solve Lattice
How to solve SVP on Quantum Computer
Quantum Computer
For now, *NISQ era / QRAM
Quantum RAM (QRAM)
Data read/write
Assume QRAM exists and is available.
Reading Lattice Vector (v) using QRAM
Quantum Circuit
Execution on Quantum Computer
Exact algorithm
NV Sieve
To solve SVP (Find c)
+
Grover's Search
To reduce search complexity

- **Through Grover's search, we can simultaneously search for c that satisfies the condition.**
  - However, **there may be multiple short vectors that satisfy the condition.**
  - Therefore, since it is a **task with multiple solutions**, **the number of Grover's iterations needs to be considered**.

Overview of Proposed Method

- **Overall Process of proposed method**

1. Input setting
2. Upscaling
3. Two's complement for the positive number
4. Addition
5. Two's complement for the negative number
6. Duplicate qubit for squaring
7. Squaring
8. Addition for squared results
9. Two's complement for the positive number
10. Addition for size comparison between ($(\boldsymbol{rR})^2$) and ($\|v - c\|^2$)



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

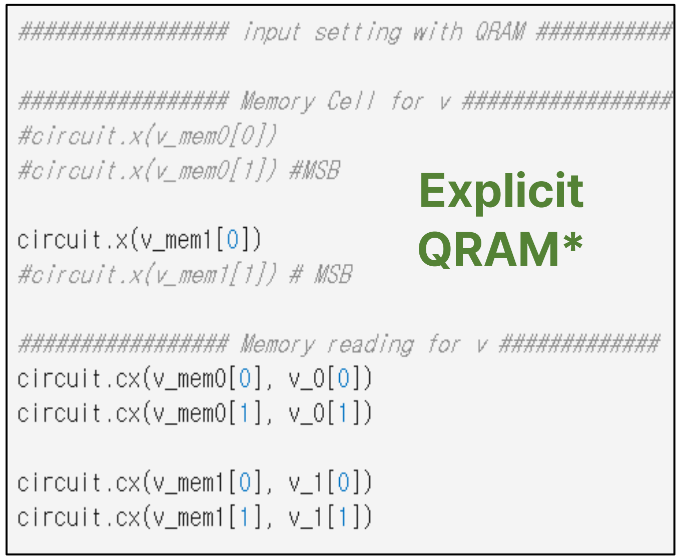
```
Initiate quantum registers and classical registers.   ▷ $carry, qflag, sqr\_result$, etc.
// Input setting (Each vector is allocated 3 qubits to address the overflow)
$v_0, v_1 \leftarrow$ Data load from memory qubits
$QNV.Hadamard(c_0)$
$QNV.Hadamard(c_1)$
$QNV.x(sqr\_rR[i])$   ▷ $0 \leq i < 6$

// To address the overflow of target qubits
$vflag[0] \leftarrow QNV.cx(v_0[1], vflag[0])$
$v_0[2] \leftarrow QNV.cx(vflag[0], v_0[2])$

$cflag[0] \leftarrow QNV.cx(c_0[1], cflag[0])$
$c_0[2] \leftarrow QNV.cx(cflag[0], c_0[2])$

$vflag[1] \leftarrow QNV.cx(v_1[1], vflag[1])$
$v_1[2] \leftarrow QNV.cx(vflag[1], v_1[2])$

$cflag[1] \leftarrow QNV.cx(c_1[1], cflag[1])$
$c_1[2] \leftarrow QNV.cx(cflag[1], c_1[2])$

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

// $v + \bar{c}$
$c_0 \leftarrow$ 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
$sqr\_result[2] \leftarrow$ Squaring$(QNV, c_0, dup\_c_0, sqr\_result[0], sqr\_result[1], sqr\_result[2], carry, 6)$
$sqr\_result[5] \leftarrow$ Squaring$(QNV, c_1, dup\_c_1, sqr\_result[3], sqr\_result[4], sqr\_result[5], carry, 6)$

// Addition for squared results to calculate the size of the vector
$sqr\_result[5] \leftarrow$ Addition$(QNV, sqr\_result[2], sqr\_result[5], carry, 6)$

// Two's complement for subtraction using adder
$sqr\_result[5] \leftarrow$ Two's complement_6bit$(QNV, sqr\_result[5], qflag4, carry, zero, zero1, zero2)$

// Size comparison between $(rR)^2$ and $(||v - c||)^2$   ▷ $((rR)^2$ - $(||v - c||)^2)$
$sqr\_result[5] \leftarrow$ Addition$(QNV, sqr\_rR, sqr\_result[5], carry, 6)$   ▷ No square root
return $c_0$, $c_1$
```

Data load on Quantum Circuit

- **Data load with QRAM**
  - **Load the input vector (v)**

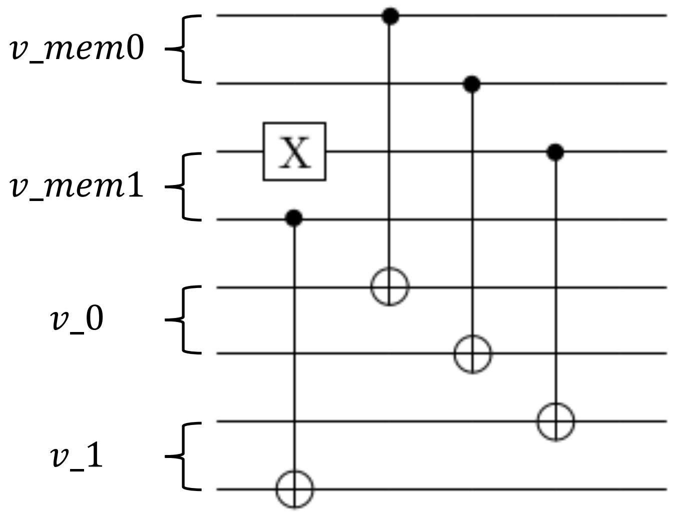
    Allocate qubits for quantum memory cell
    → $v = (00_{(2)}, 01_{(2)})$

    Copy data stored in quantum memory cell **to input qubit** (using cx gate)



```
################ input setting with QRAM ###########

################ Memory Cell for v ################
#circuit.x(v_mem0[0])
#circuit.x(v_mem0[1]) #MSB

circuit.x(v_mem1[0])
#circuit.x(v_mem1[1]) # MSB

################ Memory reading for v #############
circuit.cx(v_mem0[0], v_0[0])
circuit.cx(v_mem0[1], v_0[1])

circuit.cx(v_mem1[0], v_1[0])
circuit.cx(v_mem1[1], v_1[1])
```

Explicit QRAM*



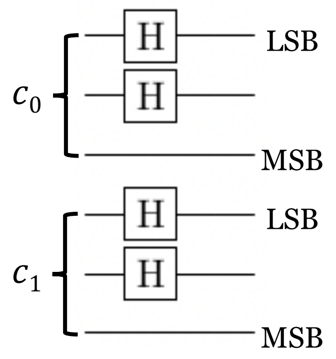
  - **Apply Grover's search repeatedly for each v (v is not a superposition)**

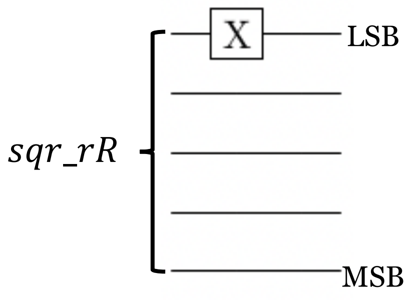
- **It is difficult to actually access QRAM.**
  - Therefore, we implement **a simple explicit QRAM on a quantum circuit.**
  - This is actually close to QROM (Quantum Read-only Memory) because it can only read data to be used.

*Explicit QRAM : Write data to a quantum circuit, read from the corresponding qubit, and query logic is required for this.

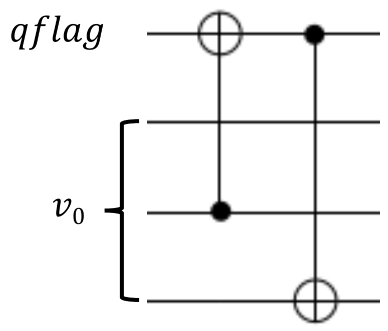
Input setting on Quantum Circuit

- **Input setting**
  - Apply **Hadamard gate** on c (**The target of Grover's search**)
  - Since $\boldsymbol{rR}$ is a classical value, we set it to its squared value ($(\boldsymbol{rR})^2$).
  - $sqr\_rR = (\boldsymbol{rR})^2 = \left(0001_{(2)}\right) = 1_{(10)}$





- **To handle overflow:**
  - We allocate **1 additional qubit for upscaling**.
  - If the dimension of v is 2:
    - Data can be represented by 2-qubits
    - But, we allocate 3-qubits for $\boldsymbol{v}, \boldsymbol{c}$ **and** $\boldsymbol{rR}$.
      - **For $\boldsymbol{c}$, Hadamard gate is applied only to 2-qubits** among 3-qubits (Search range is the same)
    - The value of **the second qubit is copied to the $qflag$.**
    - **Afterward, upscaling is completed by copying the $qflag$ to the highest qubit.**
    - Through this, **the value expressed through 3-qubits** can be expressed equally with 2 qubits.



Complement on Quantum Circuit

- **Two’s complement**
  - **After 1’s complement (bit flip), LSB+1**
  - **Since the input vector has a sign, a sign bit is required.**
  - **Since a sign bit exists, in order to perform NV Sieve Logic, complement is required for positive/negative numbers.**
    - Since $\mathbf{10_{(2)}}$**is -2, not** $\mathbf{2}$, complement operation is required to perform correct logic.
    - Complement for **positive numbers**

      → Used for subtraction through an adder ($v_0 - c_0$, $v_1 - c_1$ → $v_0 + \bar{c}_0$, $v_1 + \bar{c}_1$)
    - Complement for **negative numbers**

      → Used for integer squaring ($11_{(2)}$is -1, not 3, so the squared result is different without complement operation.)

  - **Conditional statements (e.g. if) cannot be used in quantum circuits.**
    - We utilize **control qubits** to implement a function that **calculates the two's complement** only if the number is positive or negative.
    - For this, **ancilla qubit** is used. ($qflag$)

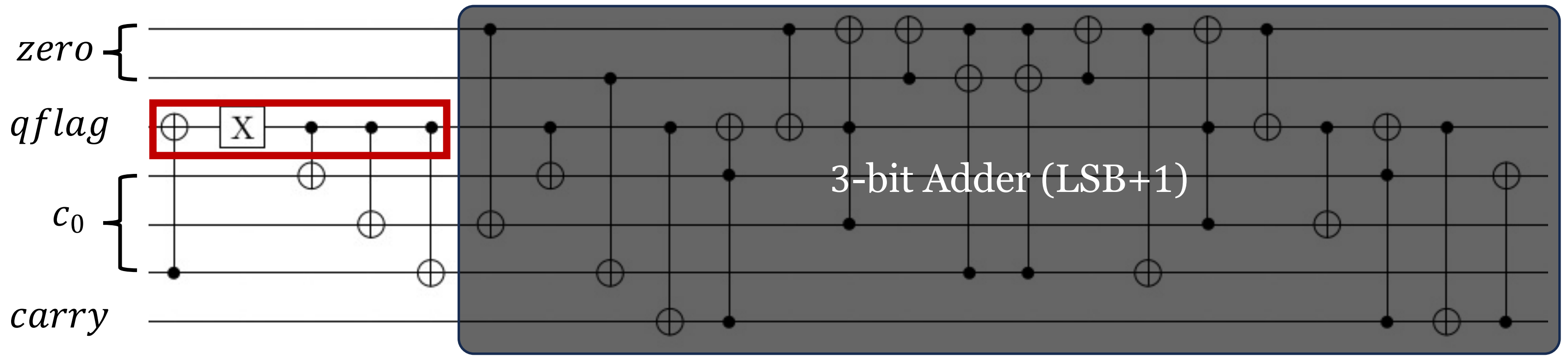
Complement on Quantum Circuit

- **2′s complement only if positive (→ MSB = 0)**
  1. MSB on qubits for c → **Copy to control qubit ($qflag$)**
  2. Since the cx gate is inverted **when the control qubit is 1**, **x gate is applied to $qflag$**. → $\boldsymbol{qflag = 1}$
  3. Perform all bit inversions using cx (**1's complement**)
  4. Create new qubit for **LSB+1** ($\boldsymbol{a}$**=**$\boldsymbol{[0, 0, qflag]}$ → a=(0,0,1))
  5. LSB+1 is performed using quantum adder.

     e.g. $\mathbf{001_{(2)}} = +1_{(10)}$ → $110_{(2)}$ (**inverted**) → $110_{(2)} + \overset{(\boldsymbol{a})}{001_{(2)}} = \mathbf{111_{(2)}} = \mathbf{-1_{(10)}}$

- **The 2′s complement, which applies only when negative, is not applied to the x gate on $qflag$.**
- Except for this part, the rest of the logic is the same

Ripple carry adder is used*.

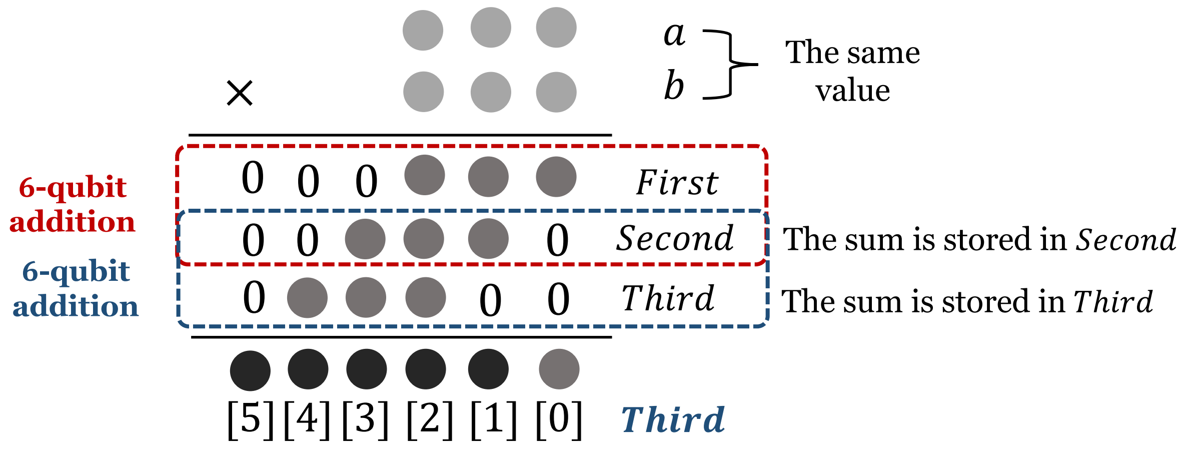


*Cuccaro, Steven A., et al. "A new quantum ripple-carry addition circuit." arXiv preprint quant-ph/0410184 (2004).

NV Sieve Implementation on Quantum Circuit

- **Squaring**
  - **Integer squaring** for values converted to positive numbers through complement.
  - Multiplication operation between the same values
    - **Cannot operate on same qubits** → **Copy the same value to other qubits** (cx gate)
  - **Allocate qubits to store the squared result** → Add the multiplication result for each qubit and store it
    - **Using a Toffoli gate, the target qubit will be 1 if and only if the target values being multiplied are both 1.**
    - Each calculation result is stored in an array of 6 qubits.

    → ex: **In the *First* array, values are stored only in the lower 3 qubits and the upper 3 qubits are set to 0.**



NV Sieve Oracle for Extended Rank/Dimension

- **Extended Rank version**
  - **Rank + 1**
    - $v = (2\ qubit, 2\ qubit, 2\ qubit)$
    - $c = (2\ qubit, 2\ qubit, 2\ qubit)$
    - $(rR)^2 = (6\ qubit)$
    - **After scaling, perform the Quantum NV Sieve logic**
    - The operation **scale is the same**, and only **calculation on additional 1 vector** is performed additionally.

- **Extended Dimension version**
  - **Dimension x 2**
    - $v = (4\ qubit, 4\ qubit)$
    - $c = (4\ qubit, 4\ qubit)$
    - $(rR)^2 = (10\ qubit)$
    - **After scaling, perform the Quantum NV Sieve logic**
    - The **calculation scale becomes larger,** and the **number of calculations remains the same**.

These extended implementations ensure scalability when the size of the lattice vector increases.

Environments

- **Simulation on Qiskit (IBM Quantum Lab)**

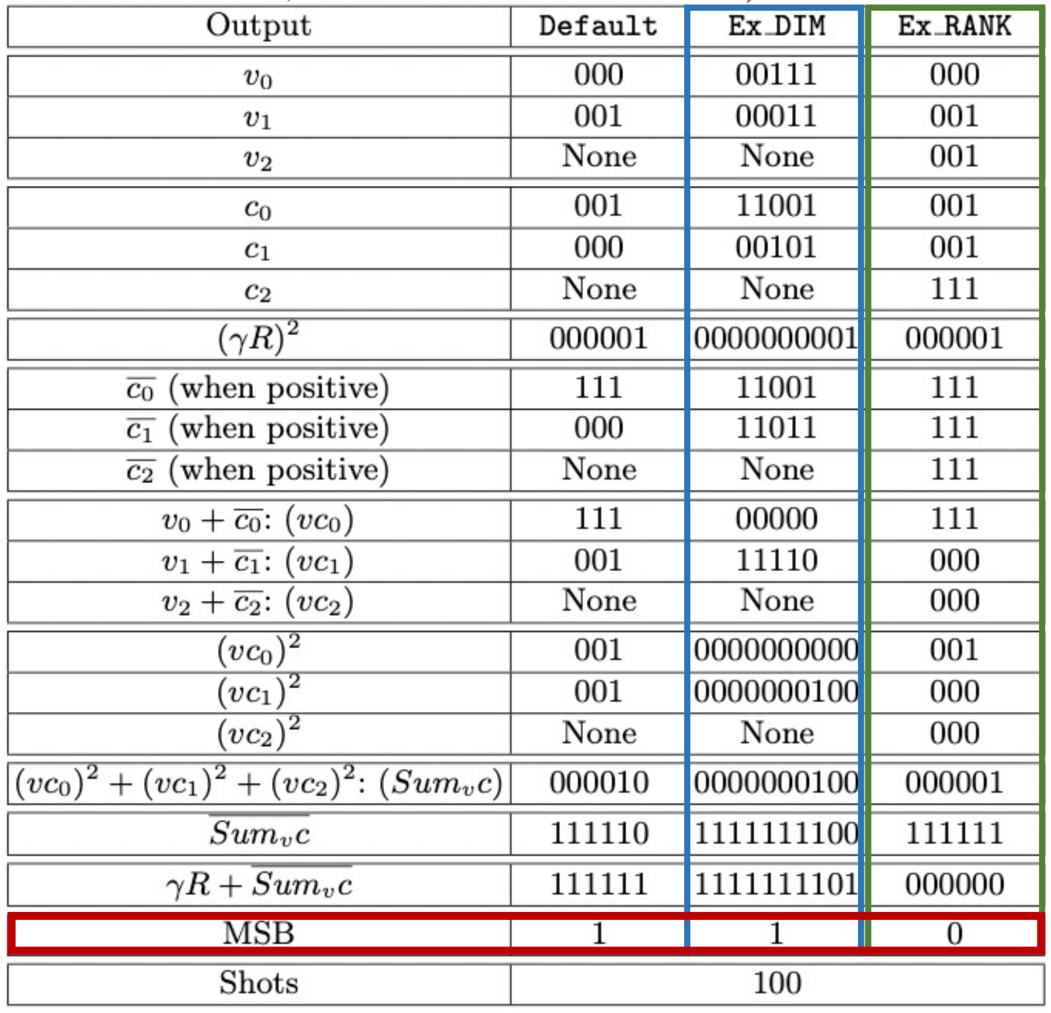
- **3 types of Oracle**
  - Default Model **(*Def*)**
  - Extended Rank Model **(*ExRank*)**
  - Extended Dimension Model **(*ExDim*)**

- **Used iteration and shots = 1**

Result of NV Sieve on Quantum Circuit

- **Calculation results for each step mentioned previously** (Example case to confirm implementation suitability)

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



Output	`Default`	`Ex_DIM`	`Ex_RANK`
v_0	000	00111	000
v_1	001	00011	001
v_2	None	None	001
c_0	001	11001	001
c_1	000	00101	001
c_2	None	None	111
$(\gamma R)^2$	000001	0000000001	000001
$\overline{c_0}$ (when positive)	111	11001	111
$\overline{c_1}$ (when positive)	000	11011	111
$\overline{c_2}$ (when positive)	None	None	111
$v_0 + \overline{c_0}$: (vc_0)	111	00000	111
$v_1 + \overline{c_1}$: (vc_1)	001	11110	000
$v_2 + \overline{c_2}$: (vc_2)	None	None	000
$(vc_0)^2$	001	0000000000	001
$(vc_1)^2$	001	0000000100	000
$(vc_2)^2$	None	None	000
$(vc_0)^2 + (vc_1)^2 + (vc_2)^2$: $(Sum_v c)$	000010	0000000100	000001
$\overline{Sum_v c}$	111110	1111111100	111111
$\gamma R + \overline{Sum_v c}$	111111	1111111101	000000
MSB	1	1	0
Shots	100		

If MSB=1 → wrong c
If MSB=0 → correct c

Grover's search of NV Sieve

- **When performing Grover's search, only the correct solution should be derived.**
  - This requires **accurate implementation of the oracle** and **the proper number of iterations.**
  - The correct iteration when multiple solutions exist is $\left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rfloor$ *Search range (N), the number of solutions (M)
  - Currently, **iteration 1** is used in the 3 types of implementations.

Result of NV Sieve on Quantum Circuit

- **Resource estimation of NV Sieve (Oracle) (Grover X, Only oracle)**

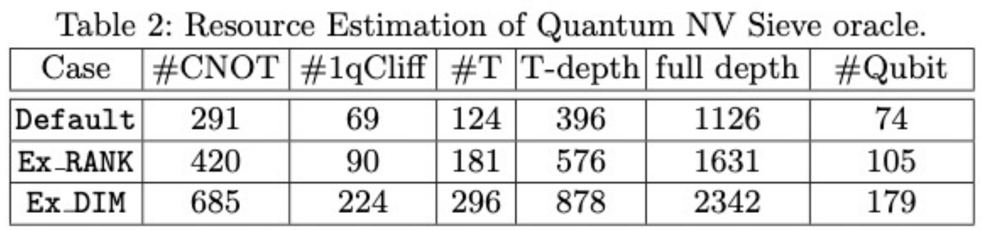


Table 2: Resource Estimation of Quantum NV Sieve oracle.

Case	#CNOT	#1qCliff	#T	T-depth	full depth	#Qubit
`Default`	291	69	124	396	1126	74
`Ex_RANK`	420	90	181	576	1631	105
`Ex_DIM`	685	224	296	878	2342	179

- **Resource estimation of NV Sieve (Grover O)**
  - **With Reverse, Diffusion operator and Hadamard gate (iteration =1)**

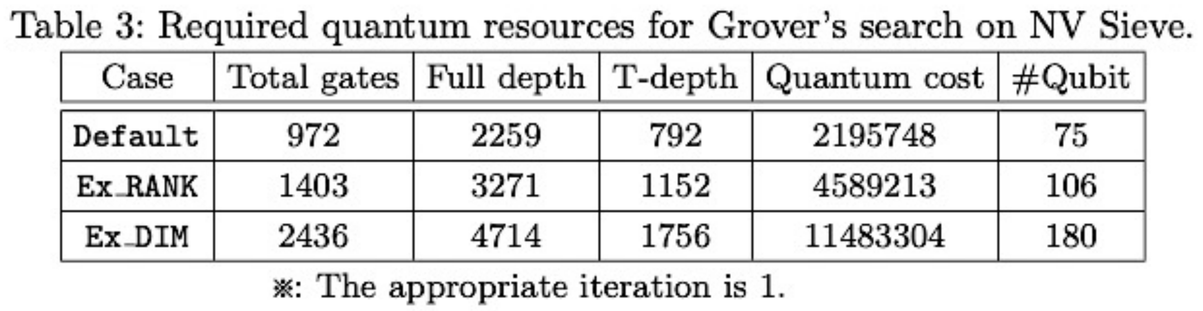


Table 3: Required quantum resources for Grover's search on NV Sieve.

Case	Total gates	Full depth	T-depth	Quantum cost	#Qubit
`Default`	972	2259	792	2195748	75
`Ex_RANK`	1403	3271	1152	4589213	106
`Ex_DIM`	2436	4714	1756	11483304	180

※: The appropriate iteration is 1.

Quantum Cost

$2^{21.06}$

$2^{22.13}$

$2^{23.45}$

Smaller amount of quantum resources than a 16-bit symmetric key (on Grover)

However, considering the very small problem scale, quite a lot of quantum resources are required.

Further Discussion

- **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.**
  - We get the proper iteration that ensures that only the correct answer is derived.

- **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.**

- **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.**

- **NISQ era**
  - In NISQ era, it is difficult to use many quantum resources.
  - As the resource estimation results indicate, **quite a bit of attack cost is required despite the small dimensions and rank.**
  - **It is thought that solving SVP for high dimensions (50~60 dimensions) such as classical is difficult for now.**

Conclusion and Future work

- **Conclusion**
  - To 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.
  - We implemented **Quantum NV Sieve using quantum circuit. (3 types of oracle)**
  - We **estimate the quantum resources** required for each case-specific oracle and the cost of Grover's attacks when applied with their Quantum NV Sieve.
  - **Despite the small problem scale, quite a bit of quantum cost is required.**
    $\rightarrow$ **Currently, it is difficult to solve SVP for lattice of approximately 50-rank/dimensions that are actually used.**

- **Future work**
  - **We have improved research results based on this work.**
    - More extended oracle implementation
    - Complexity comparison between Classical and Quantum
    - Discussion of each element of the Sieve algorithm, etc.
  - In future work, **by optimizing the oracle and implementing higher ranks and dimensions**, **we will increase the upper limit of SVP solution.**
  - Also, we will **find the correct iteration for those**, and **estimate the quantum cost of Grover's search.**

Thank you for your attention!

e-mail: khj1594012@gmail.com