

Security and privacy in post-quantum blockchain using hybrid approach of subsets in SVP within lattice-based cryptography

Ram Kumar [†]

Monica Sankat ^{*}

Nyasha Ojha [§]

School of Computer Science and Engineering

VIT Bhopal University

Kothrikalan

Sehore 466114

Madhya Pradesh

India

Abstract

This article addresses the challenges in improving the efficiency of solving the lattice-based cryptographic, Shortest Vector Problem (SVP), which is a crucial element to improve the security against the potential quantum attacks. We present a novel algorithm that integrates the concept of subsets generation with the SVP to enhance the efficiency of the lattice-based cryptographic schemes. This research delivers advancement by providing a more efficient method of solving the SVP. The method in this paper contributes towards the development of the security and privacy in cryptographic systems in the post-quantum era, surfacing the way for a vigorous encryption protocol.

Subject Classification: 94A60, 68Q12, 11H06, 68M01, 14G50.

Keywords: Blockchain, Post-quantum cryptography, Lattice-based cryptography, Post-quantum cryptographic algorithms.

1. Introduction

Within the past few decades, the technological advancement in network security and privacy domain have been a subject of sudden rise. Blockchain technology helps develop a secure transaction medium, which

[†] E-mail: drramkumar.research@gmail.com

^{*} E-mail: monicasankat.research@gmail.com (Corresponding Author)

[§] E-mail: nyashao2910@gmail.com

ensures full transparency. In the recent times, unlike systems like cloud computing- a form of centralized system is widely used, which consist of a single party holding all the power within a network, blockchain- a decentralized and distributed ledger system [1], transactions can be made without any influence by the middlemen. Blockchain is a decentralized ledger as well as a data structure where we can store information in the forms of blocks that are connected to the corresponding block with the prior block's hash value [2]. Blockchain stores immutable and time-stamped chain of blocks that are not under the control of a single entity, rather managed by the members within the network[3].

Blockchain lays the foundation of decentralized consensus mechanisms for the transaction sequence in a large network where members need not to trust or acknowledge anyone [4]. The concept of smart contracts, consensus algorithms (such as PoW- Proof of Work and PoS- Proof of Stake) along with IPFS (Inter Planetary File System) and peer-to-peer transactions, makes it well-suited for sensitive data usage and management[5].

Blockchain was first introduced and used by Satoshi Nakamoto (presumed pseudonymous person or group of people) developed Bitcoin in 2008 [6]. After that the concept flourished and marked the deployment of cryptocurrencies such as Ethereum and Dogecoin in the financial sector involving the idea of cashless i.e., digital payment through various optimal cryptographic methods including asymmetric cryptographically methods, RAS and ECC(Elliptic Curve Cryptography), hash functions such as SHA-256(Secure Hash Algorithm 256-bit) and RIPEMD-160, zero-knowledge proofs includes ZK- STARKs(Zero-Knowledge Scalable Transparent Arguments of Knowledge) as well as ZK-SNARKs(Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge), Merkle's Root Trees, along with digital signatures like ECDSA(Elliptic Curve Digital Signature Algorithm) and EdDSA(Edwards-Curve Digital Signature Algorithm) together with the consensus algorithms described above, ensures it is impermeable. However, by the development of methods such as Shor's algorithm, Grover's algorithm coupled with various attacks including MITM(Man-in-the-Middle) attack, Cryptographic Hash functions attack, Chosen- Ciphertext Attacks (CCA) and many more, proves its supposed impenetrability to be flawed because of quantum computers.

This paper dives into quantum computing and post-quantum cryptographic methods along with a hybrid approach we have developed to improve security in a world post quantum computing.

2. Quantum computers and its computing capabilities

Quantum computers are super machines that uses the attributes and parameters of non-deterministic theories of quantum mechanics to store data and perform computations on them. Unlike, classical computers which uses bits to represent data, quantum computers have qubits as the smallest unit. The qubits are either 0 or 1 or a combination of both simultaneously, hence covering all the possible aspects of any context. This allows quantum computers to perform many calculations at a time, hence, concluding the high-speed computation power of quantum computers when compared to classic computers.

As a result of this high computational power of these computers, the Shor's algorithm is able to solve the prime factorization of large integers [7] and ECC in a minimal amount of time, whereas, the Grover's algorithm provides quadratic speed as compared to classic computers for brute-force search problems over symmetric-key cryptography like 3DES and AES [8] by reducing the key length of symmetric encryption to half, hence making it less secure for the classic cryptographic security methods.

Therefore, modern-day technology requires cryptographic methods that is shielded from quantum computers and its computing abilities and such cryptographic methods are known as post quantum cryptographic methods used over post-quantum blockchain (PCB). PCBs are designed to secure the blockchain systems by potential attacks and threats posed by quantum computers.

3. Post quantum cryptography

Post quantum cryptographic methods are designed and leveraged to protect and secure data against the potential threats by quantum computers. Quantum computers, when fully scalable and extensible, will be able to solve many cryptographic problems fast a very fast pace than the classical computers. This poses a high risk on the cryptographic methods used today, and hence the requirement of post quantum cryptographic methods comes to face.

4. Post quantum cryptographic algorithms

4.1 Code-based cryptography

This form of quantum cryptography depends upon coding theories for its security, contrasting with the traditional mathematical computations

based upon number theories such as the discrete logarithm problem or the factorization of large integers. Code-based cryptography leverages the core of decoding random linear code lines for its security foundation.

The basics of this method consist of linear codes and the decoding problems. Linear codes defined as set of vectors (codewords), are the structural mathematics used to encode data in a vector space in a finite field space. Obtaining the codewords with the received vector (may contain errors), the problems involve locating the nearest set of vectors, codewords, to the received vector.

McEliece cryptographic system is the most famous example of code-based cryptography [7].

4.2 Hash-based cryptography

Merkle Signature Scheme (MSS) and SPHINCS+ digital signatures and SHA-256 and SHA-3 hash functions are the most widely used hash-based cryptographic schemes.

These schemes are the cryptographic methods which depend upon the hash function for the method security. Hash-based cryptography is widely used in public key infrastructures and digitally secured signatures [9].

The concept of hash-based function comprises of the hash function and OTS (One-Time Signature). Hash function takes the “message” in the form of input and produces a random byte size string. These are impracticable to reverse, in other words reconstruction of the original input from the hash function is infeasible. Whereas the OTS, signature schemes, creates a unique signature for each and every message. These have inherent security effects, hence are usually used only once.

4.3 Multivariate Quadratic Cryptography

Multivariate Quadratic cryptographic methods rely on the complicated solutions of multivariate quadratic equations, which forms the basis of this cryptographic method. The solution is needed to be found within a finite field space [7].

The quadratic polynomial equations comprise of multiple variables and the security of this method is employed by cryptographic protocols such as public-key encryption with digital signatures.

The mathematical polynomial equation with n variables has the general form:

$f(x_1, x_2, x_3, \dots, x_n) = \sum_{(i,j)} a_{ij} x_i x_j + \sum_i b_i x_i + c$; where a_{ij} , b_i and c are coefficients and x_i and x_j are the variables.

The multivariate quadratic equation is considered to be a NP-hard problem, thus being computationally infeasible and therefore, enhancing the security system.

4.4 *Isogeny-based cryptography*

Derived from the mathematical theory of isogenies between elliptic curves, this method of cryptography is well known as isogeny-based cryptography [7].

The term isogeny refers to the function that conserve the elliptic curve structure.

Mapping an elliptical curve onto another and they have a finite kernel.

Super singular Isogeny Diffie-Hellman (SIDH) and Super singular Isogeny Key Exchange (SIKE) are two of the secure key exchange protocols and Isogeny-based Signature Schemes are the cryptographic scheme used to generate digital signatures which often depends on the isogenies computational difficulties between the super singular elliptical curves.

The existing quantum algorithms are not quite sufficient to solve this problem which makes it a better choice for security protocol against quantum computers.

4.5 *Lattice-based cryptography*

Lattice-based cryptography pivots on the mathematical structures of lattice for security. Lattices are a sequence of points in a n-dimensional space, formed from the linear combinations of a set of basis vector [7].

The basis of lattice-based cryptography roots from various mathematical problems that are quite hard to solve such as Closest Vector Problem (CVP), in which the point closes to the target vector is to be found. Other such lattice-based problem is LWE (Learning with Errors), in which one has to recover the secret vector underneath a vector with errors [9]. Additionally, the Shortest Vector Problem (SVP) encourages to find the shortest non-zero vector within the lattice.

This article presents a hybrid approach to the Shortest Vector Problem (SVP), which can increase the computational complexity, hence ensuring the increased security of data by classical as well as quantum computers. Techniques like Lenstra-Lenstra-Lovász (LLL) algorithm can reduce the basis vector of a lattice [7], hence reducing its complexity.

In this approach, the concept of subsets selection is adjoined with the concept of basis generation to enhance the computational complexity of this cryptographic method.

This algorithm generates all possible subsets of the given set of integers, hence providing various basis vector combinations.

The identified subsets are further used to build a lattice basis, B_n . B_n further is employed to generate the candidate vectors. The candidate vectors can be generated by the linear combination of the selected basis vector. The candidate vectors are later subjected to fulfil required constraints. The constraints include the boundary condition (the shortest vector should be found within a specified range) and the modulo constraint. Afterwards, the algorithm calculates the length of the candidate vector using the Euclidean method and the vector with the smallest non-zero length is concluded to be the shortest vector.

5. Enhanced Algorithm for Solving the Subset Shortest Vector Problem in Lattice-Based Cryptography

By integrating a systematic approach, the suggested algorithm improves the efficiency of lattice-based cryptographic schemes by solving the Subset Shortest Vector Problem (SVP) with extra restrictions. To set the stage for later calculations, the procedure starts by initializing the shortest vector and its length to infinity. Under modular and summation constraints, it creates a collection of candidate vectors based on a given lattice basis defined by integer combinations of the basis vectors. The building of lattice bases, which act as a basis for the generation of candidate vectors, is made possible by the identification of subsets from the created set. Every potential vector is examined under strict limitations, such as modulo conditions and restricted sum limits, to make sure it satisfies the requirements for additional assessment.

Algorithm 1: Algorithm for the SVP problem with additional constraints of the Subset

- 1: **Step 1: - Initializing the shortest vector;**
- 2: Start with no vector and the length set to infinity;
- 3: ShortestVector = null
- 4: shortestLength = infinity
- 5: **Step 2: - Generate a set $S_1, S_2, \dots, S_n$ that represents the elements from which subsets will be derived**

```

6:      Sn = {s1, s2, ..., sn}
7: Step 3: - Subset identification
8:      From the generated Sn, identify all possible subsets, which will then be used to
      generate the lattice basis.
9:      subset = generatingSubset(Sn)
10: Step 4: - Generation of the lattice basis
11:      Use the identified subsets to generate the basis vectors for the lattice
12:      Bn = generatingBasis(subSets)
13: Step 5: - Generating the candidate vectors:
14:      Using the lattice basis, generate all possible candidate vectors that fit the
      constraints. Constraints include modulo conditions, sum boundaries, and
      subset restrictions.
15:      candidateVectors = generateCandidateVector(Bn, modulo, k, SumBound)
16: Step 6: - Iterating over the candidate vectors:
17:      Iterate through the candidate vectors, calculate the length of each, and compare it
      with the current shortest vector. Update the shortest vector if a smaller, valid one is
      found.
18: for each candidate in candidateVectors:
19:     length = calculateLength(candidate)
20:     if length < shortestLength and length > 0:
21:         shortestVector = candidate
22:         shortestLength = length
23: Step 7: - Return the analysed shortest Vector:
24:     return shortestVector

```

Subsidiary(helping) functions:

```

//Defining the candidate generating function
function generateCandidateVectors(B, modulo, k, sumBound):
    candidateVectors = []
    for i in range (r1, r2):
        for j in range (r3, r4):
            candidate = vectorSummation(i, j, B)
            if isCandidateValid(candidate, modulo, k, sumBound):
                candidateVectors.append(candidate)
    return candidateVectors

//Generating the set
function setGeneration(n, range):
    Sn= []
    for i from 0 to n+1:
        Sn.append(randomIntGeneration(range))

```

```
return Sn
```

```
//Identifying the subsets
```

```
function generatingSubset(Sn) :
```

```
    subSets = []
```

```
    numberOfSubsets = 2 ^ length (Sn)
```

```
    for i from 0 to numberOfSubsets - 1:
```

```
        subset = []
```

```
        for j from 0 to length (Sn) - 1:
```

```
            if (1== 1) & (i>>):
```

```
                subset.append(Sn[ j ])
            
```

```
        subSets.append(subset)
```

```
    return subSets
```

```
// generation of the lattice basis
```

```
function generatingSubset(Sn):
```

```
    Bn = []
```

```
    for subset in subSets:
```

```
        if subset! = [ ]:
```

```
            vectorBasis = subset + [0] * (length(subSets) - length(subset))
```

```
            Bn.append(vectorBasis)
```

```
    return Bn
```

```
//Computing i * b1 + j * b2
```

```
function vectorSummation(i, j, Bn):
```

```
    sumVector = []
```

```
    // Assuming a 2D lattice for simplification
```

```
    for k in range (len (B [0] ) ):
```

```
        summationComponents = i * Bn[0][k] + j * Bn[1][k]
```

```
        sumVector.append(summationComponents)
```

```
    return sumVector
```

```
//Checking whether the candidates meet the required constraints
```

```
//Checking the modulo and the sum constraint
```

```
function isCandidateValid(candidate, modulo, k, sumBound):
```

```
    for component in candidate:
```

```
        if component % modulo! = k:
```

```

        return false
    vectorSummation = sum(candidate)
    if vectorSummation < sumBound[0] or vectorSummation > sumBound[1]:
        return false
    return true

//Computing the Euclidean length of the vector
function calculateLength(vector):
    length = sqrt (sum [component^2 for component in vector]):
    return length

```

Example with integer basis vector: -

Consider a simple 2-dimensional lattice with basis vectors:

$b_1 = (4,2)$ and $b_2 = (1,3)$

Constraints are defined as:

- Modulo constraint: $x \bmod 3 = 1$
- Constraint for bounded sum: $10 \leq \sum x \leq 20$

STEP-BY-STEP IMPLEMENTATION

Step 1: - Initialising the shortest vector

```

shortestVector = null
shortestLength = infinity

```

Step 2: - Generating the candidate vectors

```

vectorCandidate = generateCandidateVectors([[4, 2], [1,
3]], 3, 1, [10, 20])

```

Step 3: - generateCandidateVectors

```

for i = -1 to 1 and j = -1 to 1
for i in range (-1, 2):
    for j in range (-1, 2):
        candidate = vectorSummation(i, j, [[4, 2], [1, 3]]):
        if isCandidateValid(candidate, 3, 1, [10, 20]):
            candidateVectors.append(candidate)

```

Step 4: - vectorSummation

```

vectorSummation(-1, -1, [[4, 2], [1, 3]]) = [-5, -5]
vectorSummation(-1, 0, [[4, 2], [1, 3]]) = [-4, -2]

```

```

vectorSummation(0, -1, [[4, 2], [1, 3]]) = [-1, -3]
vectorSummation(0, 0, [[4, 2], [1, 3]]) = [0, 0]
vectorSummation(1, 0, [[4, 2], [1, 3]]) = [4, 2]
vectorSummation(0, 1, [[4, 2], [1, 3]]) = [1, 3]
vectorSummation(1, 1, [[4, 2], [1, 3]]) = [5, 5]

```

Step 5: - isCandidateValid

```

isCandidateValid([-5, -5], 3, 1, [10, 20]) = false
isCandidateValid([-4, -2], 3, 1, [10, 20]) = false
isCandidateValid([-1, -3], 3, 1, [10, 20]) = false
isCandidateValid([0, 0], 3, 1, [10, 20]) = false
isCandidateValid([1, 3], 3, 1, [10, 20]) = true
isCandidateValid([4, 2], 3, 1, [10, 20]) = false
isCandidateValid([5, 5], 3, 1, [10, 20]) = false

```

Step 6: - Iterating over candidate vector

```

for candidate in vectorCandidates:
    length = calculateLength(candidate)
    if length < shortestLength and length > 0:
        shortestVector = candidate
        shortestLength = length

```

Step 7: - calculateLength

```

calculateLength([1, 3]) = sqrt(1 + 9) = sqrt(10) = 3.16

```

Step 8: - Shortest Vector

```

return [1, 3]

```

The example demonstrates the application of an algorithm to solve the Subset Shortest Vector Problem (SVP) in a 2-dimensional lattice. The lattice is defined by two basis vectors, $b_1 = (4, 2)$ $b_{-1} = (4, 2)$ $b_1 = (4, 2)$ and $b_2 = (1, 3)$ $b_{-2} = (1, 3)$ $b_2 = (1, 3)$. The goal is to find the shortest vector in this lattice that satisfies two constraints:

1. **Modulo constraint:** $x \bmod 3 = 1$ $x \bmod 3 = 1$ $x \bmod 3 = 1$
2. **Bounded sum constraint:** $10 \leq \sum x \leq 20$

Example with Tight Constraints and Higher Dimensional Lattice

Lattice Basis Vectors:

- $b_1 = (6, 2, 4)$ $b_{-1} = (6, 2, 4)$ $b_1 = (6, 2, 4)$
- $b_2 = (2, 8, 3)$ $b_{-2} = (2, 8, 3)$ $b_2 = (2, 8, 3)$
- $b_3 = (3, 1, 7)$ $b_{-3} = (3, 1, 7)$ $b_3 = (3, 1, 7)$
- $b_4 = (1, 5, 6)$ $b_{-4} = (1, 5, 6)$ $b_4 = (1, 5, 6)$

Constraints:

- Modulo constraint: $x \bmod 5 = 3$
- Bounded sum constraint: $25 \leq \sum x \leq 30$

Step-by-Step Implementation:

Step 1: Initializing the Shortest Vector

- Initialize the shortest vector as null.
- Set the shortest length to infinity.

Step 2: Generating Candidate Vectors

- Generate all possible candidate vectors using combinations of basis vectors b_1, b_2, b_3, b_4 .
- Iterate over integer values for coefficients i, j, k, l in the range $-2 \leq i, j, k, l \leq 2$.

Step 3: Vector Summation

- Compute the sum of the basis vectors multiplied by the coefficients i, j, k and l .

Example calculations for vector summation:

- $\text{vectorSummation}(1, -1, 1, 0, [[6, 2, 4], [2, 8, 3], [3, 1, 7], [1, 5, 6]]) = (7, -5, 8)$
- $\text{vectorSummation}(0, 1, 0, -1, [[6, 2, 4], [2, 8, 3], [3, 1, 7], [1, 5, 6]]) = (1, 3, -3)$

Step 4: Checking Constraints

- Apply the modulo and sum constraints to each candidate vector.

Example checks:

- $\text{isCandidateValid}([7, -5, 8], 5, 3, [25, 30]) \rightarrow \text{False}$ (Modulo constraint fails).
- $\text{isCandidateValid}([1, 3, -3], 5, 3, [25, 30]) \rightarrow \text{False}$ (Sum constraint fails).

Step 5: Finding Valid Candidates

- Valid candidates that satisfy both constraints are scarce due to tight conditions.
- **Valid candidate:** $[13, 3, 9]$ satisfies both the modulo and sum constraints:
 - o Modulo check: $13 \bmod 5 = 3$

- o Sum check: $\Sigma [13,3,9] = 25$

Step 6: Calculating Length and Finding the Shortest Vector

- For valid candidates, calculate their Euclidean lengths and compare them with the current shortest length.

Example length calculations:

- $\text{calculateLength}([13,3,9]) = 13^2 + 3^2 + 9^2 = 169 + 9 + 81 = 259 \approx 16.09$

Step 7: Return the Shortest Vector

- The shortest vector that satisfies the constraints is $[13,3,9]$, with a length of approximately 16.09.

Conclusion

In this study, a novel approach towards the Shortest Vector Problem (SVP) in the lattice cryptographic method with the incorporation of subsets generation algorithm. The improved algorithm enhances the efficiency of lattice-based cryptography systems and also builds resistance against the potential quantum threats. While the algorithm provides significant improvements, scalability in higher dimensions, still remains a limitation to be addressed in further researches.

References

- [1] D. Efanov and P. Roschin, "The all-pervasiveness of the blockchain technology," *Procedia Computer Science*, vol. 123, pp. 116-121 (2018), doi: 10.1016/j.procs.2018.01.019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050918300206>.
- [2] R. Chatterjee and R. Chatterjee, "An overview of the emerging technology: Blockchain," in *2017 3rd International Conference on Computational Intelligence and Networks (CINE)*, Odisha, India, pp. 126-127 (2017), doi: 10.1109/CINE.2017.33.
- [3] M. Abouaroek and K. Ahmad, "Performance analysis of NTRU algorithm with non-post-quantum algorithms," *J. Discrete Math. Sci. Cryptogr.*, vol. 24, no. 5, pp. 1349-1363 (2021), doi: 10.1080/09720529.2021.1932926.

- [4] I. R. Khan, N. Gupta, S. Hundekari, A. Bansal, and R. A. Mahajan, "Efficient key revocation in WSN with lattice-based cryptography," *J. Discrete Math. Sci. Cryptogr.*, vol. 27, no. 2-A, pp. 339–348 (2024), doi: 10.47974/JDMSC-1887.
- [5] Y.-L. Gao, X.-B. Chen, Y.-L. Chen, Y. Sun, X.-X. Niu, and Y.-X. Yang, "A secure cryptocurrency scheme based on post-quantum blockchain," *IEEE Access*, vol. 6, pp. 27205–27213 (2018), doi: 10.1109/ACCESS.2018.2827203.
- [6] G. Tripathi, M. A. Ahad, and G. Casalino, "A comprehensive review of blockchain technology: Underlying principles and historical background with future challenges," *Decision Analytics J.*, vol. 9, Art. no. 100344 (2023), doi: 10.1016/j.dajour.2023.100344. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2772662223001844>.
- [7] National Institute of Standards and Technology, *Report on Post-Quantum Cryptography* (2016). [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf>.
- [8] R. Bavdekar, E. J. Chopde, A. Agrawal, A. Bhatia, and K. Tiwari, "Post quantum cryptography: A review of techniques, challenges and standardizations," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Bangkok, Thailand, pp. 146–151 (2023), doi: 10.1109/ICOIN56518.2023.10048976.
- [9] J. Buchmann, E. Dahmen, and M. Szydło, "Hash-based digital signature schemes," in *Post-Quantum Cryptography*, D. J. Bernstein, J. Buchmann, and E. Dahmen, Eds. Berlin, Germany: Springer, pp. 35–60 (2009), doi: 10.1007/978-3-540-88702-7_3.

Received October, 2024