LWR-based Quantum-Safe Pseudo-Random Number Generator

Anupama Pandit[a], Atul Kumar[a], Arun Mishra*[a]

[a]*Department of Computer Science and Engineering, Defence Institute of Advanced Technology Girinagar Pune, 411025, Maharashtra, India*

Abstract

Learning with Rounding (LWR) is a lattice-based cryptographically hard problem which quantum computers find difficult to solve. LWR problem is a derandomized variant of Learning with Errors (LWE) where deterministic rounding is used to generate errors efficiently. LWR can construct a secure seed for Quantum-Safe Pseudo-Random Number Generator (QSPRNG). This study is an endeavor to construct a QSPRNG to generate a stream of pseudo-random bits through a Linear Feedback Shift Register (LFSR). The proposed QSPRNG uses secure seed as an input and a Homomorphic Function for preserving the security of the internal states of LFSRs. NIST statistical tests, ENT, and DIEHARD tests are performed on constructed PRNG for randomness analysis. Also, security and speed analysis for the proposed QSPRNG has been done to illustrate its possible future utilization in cryptography.

Keywords: Pseudo-Random Number Generator, Lattice-Based Cryptography, Learning with Rounding, Linear Feedback Shift Register, Homomorphic Function, NIST Statistical Test Suite

1. Introduction

A Pseudo-Random Number Generator (PRNG) is a computational function that generates a series of random numbers using a deterministic process. Most cryptographic protocols need to produce and use confidential values that must not be revealed to an intruder. Random numbers are required to generate nonces, challenges, blinding values, and padding bytes. These are also used to produce pri-

Email address: arunmishra@diat.ac.in (Arun Mishra*)

vate/public key pairs for asymmetric cryptographic algorithms like Diffie-Hellman [1], DSA [2], and RSA [3]. The crucial thing is that attackers/intruders and those familiar with the design must not be capable of making any fruitful prognosis about the random number generator output.

Security of various protocols [3, 2, 4] is dependent on the security of PRNGs. If a random number sequence is insecure, then the whole system based on that protocol would be insecure. Therefore, most of the secure systems use a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG).

Yarrow160 [5, 6], Fortuna [7], Counter-Based Random Number Generator [8] (CBRNG), Chaotic map based PRNG [9, 10], and Blum-Blum Shub [11] are a few examples of the CSPRNG for the classical computer. But problem arises with the advent of quantum algorithms. With the help of quantum computers, Grover's search algorithm [12] may be able to break Yarrow 160 and Fortuna RNGs which use SHA1 and SHA256 hash functions respectively. Similarly, Shor's [4] may be used for factorization and seed recovery in case of Blum-Blum Shub which is based on the hardness of factorization problem. Security of CBRNG depends on key (seed) value and encryption algorithm which is usually Advanced Encryption Standard (AES) [13] algorithm. Grover's algorithm can retrieve this seed value to breach the security of CBRNG. This makes CBRNG unsecured against quantum attacks. Therefore, a quantum safe PRNG is required which is secure against quantum attacks.

In this study, a scheme for Quantum-Safe PRNG (QSPRNG) is proposed to resolve the issues pertaining to post-quantum security. The proposed scheme comprises of following two major parts:
i) Secure Seed Construction: In this scheme, the seed is constructed using Quantum-Safe lattice based hard problem.
ii) Quantum Safe Random Bit Generation: In this scheme, homomorphic function is used for operations on encrypted data to conserve the secrecy of seed.
The main contribution of the proposed work is the development of a Quantum safe PRNG to generates random sequences which is statistically well-distributed, better random bit generation speed compared to existing PRNGs and have a very low correlation.

In 2005, a theoretical computer scientist Oded Regev proved that the lattice-based cryptographic scheme, Learning with Errors (LWE) [14] is secure against

Quantum attacks. Later Abhishek Banerjee et al. introduced a derandomized variant of LWE known as Learning with Rounding (LWR) [15]. Since the first objective is to generate the quantum-safe seed and generate the same sequence at the sender and receiver ends, which requires derandomization, the lattice-based derandomized problem Learning with Rounding is used. According to our knowledge, no algorithm exists in either classical or quantum theory that can break lattice-based LWR algorithm in polynomial time [15]. The constructed secure seed is used as an input to the quantum-safe approach that generates sequence of random bits. The proposed approach uses distributed Pseudo-Random Functions (PRFs) with arbitrary binary operations. The circuit (*FIGURE* 1) uses the secure seed as an input. It segments the input into three parts in such a manner that it does not change the lattice properties of LWR. The primary objective of the homomorphic function is to perform operations on encrypted information while maintaining seed secrecy. The proposed approach is quantum-safe because the homomorphic function is used to build circuits in such a way that the homomorphic function computes encrypted data (secure seed) to generate bitstream without losing lattice features.

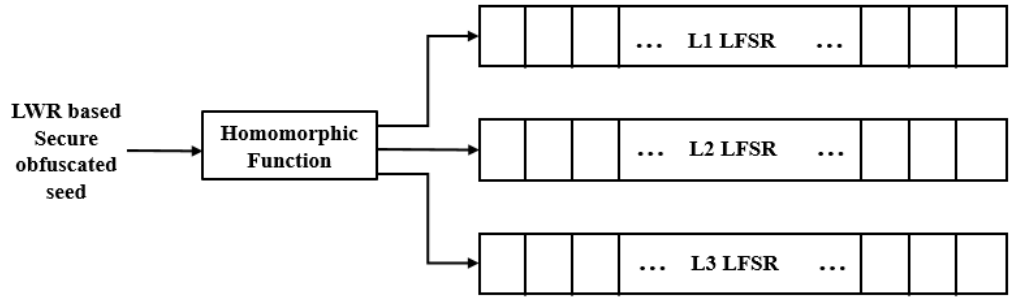


Figure 1: Initial Representation of LFSR Circuit

The rest of the paper is organized into the following sections: Section 2 discusses work related to PRNGs. Section 3 covers prerequisites for Pseudo-Random Generators, Lattice-Based Cryptography, Lattice-Based Hard Problems, LFSR and key distribution using quantum channel. Section 4 covers the proposed approach for developing a Quantum-Safe Pseudo-Random Number Generator in detail. Section 5 discusses the security of the proposed QSPRNG. Section 6 discuss results of randomness tests and security analysis. In section 7, a comparison of the proposed QSPRNG with other PRNG methods and speed analysis has been done. Section 8 concludes the proposed research work.

2. Related Work

PRF could be constructed by a one-way function and be the heart of symmetric key cryptography. Lattice makes PRF inefficient pertaining to high performance. Despite that, most of the lattice-based primitives like encryption, identification, and authentication are extremely parallelizable and comparatively efficient. This drives researchers to research how to construct the core object of symmetric key cryptography using lattices [16] that would be efficient and secure against quantum attacks.

The concept of PRFs and PRNGs was first formalized by Goldreich, Goldwasser, and Micali through GGM Construction [17]. Since then, the applications of PRFs have been realized in constructing various cryptographic primitives like Message Authentication Codes, Symmetric Encryption Schemes, Key Derivation functions, etc. Naor and Reingold (NR) [18] derived another basic way of constructing PRFs known as NR constructions. GGM and NR construction are theoretical building blocks of PRFs and PRNGs which are instantiated with specific hardness functions like Number Theoretic assumptions and Lattice-based hardness assumption [19], which result in efficient PRFs. The Number-Theoretic assumption was based on the Decision Diffie Hellman (DDH) [20] assumption. In contrast, Lattice-based instantiations are based on derandomized form of Learning with Errors Problem (LWE) is called Learning with Rounding Problem (LWR) [15]. Banerjee et al. instantiated GGM and NR with LWR assumption, but the construction did not turn out to be as efficient as Number Theoretic assumptions. Hence, they gave a direct construction of PRFs from LWR assumptions. GGM and NR constructions are generic methods of obtaining PRFs. In practice, efficient PRFs are not obtained through generic methods hence practical implementation is done through the use of notions of PRNGs and stream ciphers which define the building blocks of symmetric encryption schemes [20]. Various PRNGs are obtained from stream ciphers [21] like A5 [22], which is based on recursive relation obtained by primitive polynomial. These efficient PRNGs run indefinitely until the period of the polynomial is reached. Resembling this, PRNGs are constructed using Feistel structures, mainly by permutation and substitution techniques used in block ciphers. These constructions suffer from attacks related to seed – Differential cryptanalysis [23].

One of the paradigms in preserving seed comes from the contemporary construction of PRFs through homomorphic functions. Homomorphic function [24]

allows computation on encrypted data and is used in various applications of cloud storage and Zero-Knowledge privacy. Key – Homomorphic PRF [25] introduced by Naor, Pinkas, and Reingold allow efficient evaluations of $F_{s1+s2}(x)$ given the values of F_{s1} and F_{s2} and its applications in constructing distributed PRFs. Distributed PRFs [25] allow seed splitting into n parts so that they are reconstructed without reconstructing the key at one location.

Since PRFs are a core part of PRNGs, quantum algorithms could break them in polynomial time. Therefore, to protect PRFs from such attacks, Lattice-based LWR approach could help construct an efficient and deterministic PRF.

3. Preliminaries

3.1. Random Number Generator [26]

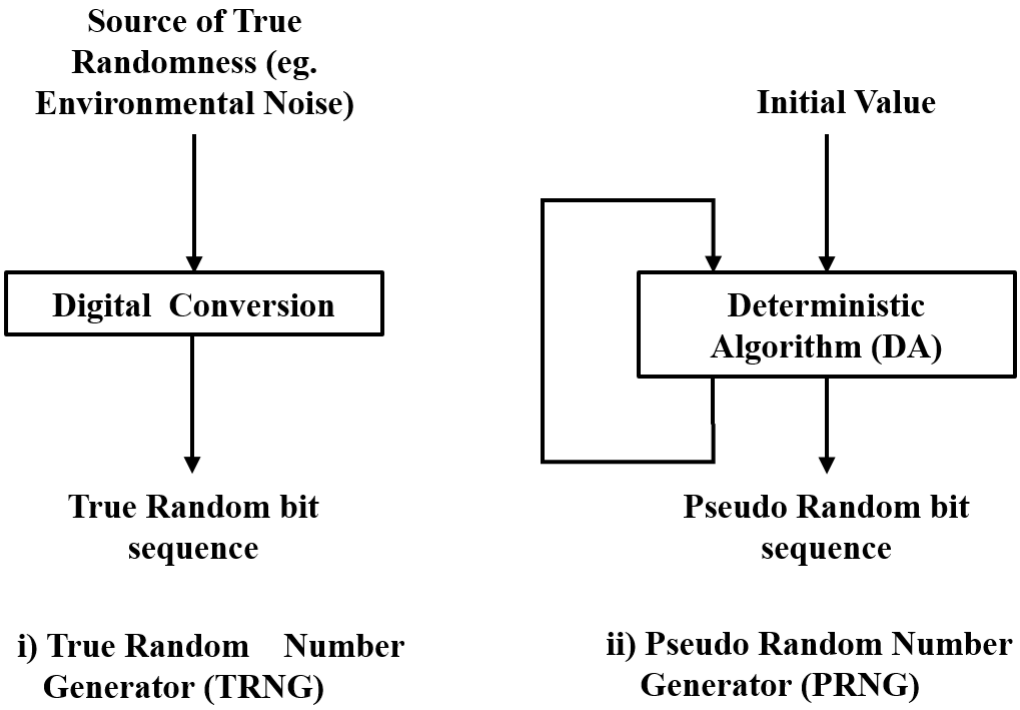


Figure 2: Random Number Generators

There are two kinds of RNGs. Pseudo Random Number Generators (PRNGs) and True Random Number Generators (TRNGs). TRNG takes input from the ex-

ternal natural sources in an analog format through sensors and then its conversion tasks place into the digital format and gives output in the form of sequence of random number streams. Whereas PRNG takes seed value or initial value from user or an external entity such as another process then applies mathematical operations over that and produces a stream of random numbers that can be served as a seed for next iteration. This procedure is iterated to produce randomness (Figure 2).

3.2. Lattice-based Cryptography [27]

The most popular classical computer algorithm is RSA algorithm, whose security depends on the large integer factoring problem. It is hard to factor a large composite number (with exactly two large prime factors) in polynomial time with a classical computer. But a quantum computer can find the solution of this problem in polynomial time by using Shor's algorithm [4]. Therefore, prime factorization is not a hard problem for a quantum computer. However, as soon as quantum algorithms are developed, a difficulty appears. Since Grover's [12] search method employs quantum computers, the underlying structure of Fortuna, and Yarrow 160 PRNG, which use SHA-256 and SHA-1 hash functions, respectively, can be broken [28]. In spite of that, some cryptography methods are safe in presence of quantum computer, such as lattice-based cryptography [19]. This type of cryptography is called Post-Quantum Cryptography (PQC) [29] .

Lattice-based cryptography is among the major competitors in the field of PQC that defines the framework of cryptographic primitives that entail lattices. A lattice [19] is a bunch of well-organized points evenly distributed in an $n-$dimensional space. Points could be visualized as a grid on graph paper. The security of cryptosystems depends on the hardness of the lattice problems and most of the cryptosystems are based on their hardness only. Shortest Vector Problem (SVP) [27] and Closest Vector Problem (CVP) [30] are hard lattice problems. SVP – Find the shortest vector from origin for a given lattice. CVP – Find closest point in a given lattice from a given target point that is not in the lattice. The security of lattice-based cryptography derives from its worst-case hardness assumption. It is also secured against quantum computers [31]. Lattices are used in many cryptographic algorithms, for example, digital signatures, asymmetric encryption, identity-based encryption, encryption defiant to key leakage attacks, and homomorphic encryption [32].

LWE and LWR are two crucial algorithms used in Lattice-based cryptography.

3.2.1. Learning with Errors [14]

It is assumed that the LWE problem is as difficult as the worst-case lattice problem [31]. Therefore, construction of such kind of cryptographic system would be secured against quantum attacks. In this problem, it is asked to find out the secret vector from an arbitrary number of noisy linear equations.

$$a_{11}\,s_1 + a_{12}\,s_2 + \ldots\, a_{1n}\,s_n + e_1 = b_1 \; mod \; q$$
$$a_{21}\,s_1 + a_{22}\,s_2 + \ldots\, a_{2n}\,s_n + e_2 = b_2 \; mod \; q$$
$$\ldots$$
$$\ldots$$
$$\ldots$$
$$a_{m1}\,s_1 + a_{m2}\,s_2 + \ldots\, a_{mn}\,s_n + e_m = b_m \; mod \; q$$

$(a_{11}, a_{12}, ..., a_{mn})$ are the elements of matrix A whereas $(s_1, s_2, ..., s_n)$, $(e_1, e_2, ..., e_m)$ and $(b_1, b_2, ..., b_m)$ are the elements of vectors s, e and b respectively. In short, the above linear equations could be written as $A \cdot s + e = b \; mod \; q$. Where $s \in \mathbb{Z}_q^m$, $A \in \mathbb{Z}_q^{n \times m}$, $e(error) \in \chi^n$, $q \in$ prime number, m and n represent rows and columns of the matrix respectively which contains uniformly distributed sampled values. The matrix A, secret vector s, error e and $b \; mod \; q$ could be represented in matrix/vector form as below:

$$A = \begin{bmatrix} a_{11} & a_{12} & & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{m1} & a_{m2} & & a_{mn} \end{bmatrix}, \; s = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}, \; e = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{bmatrix}, \; b \; mod \; q = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \; mod \; q$$

The error is randomly selected from Gaussian error distribution with mean zero and low standard deviation.

As errors are augmented, it is hard to find the secret s; otherwise, it is possible to solve these equations in polynomial time using the Gaussian Elimination method. The LWE algorithm could be divided into three parts:

i) Key Generation:

$$pubKey = \{A, b = A \cdot s + e\} \quad and \quad secKey = s$$

ii) Encryption:

$$Enc(pubKey, bit) = \{cipher\ text\ preamble\ (u) = A \cdot x,$$
$$cipher\ text\ (u') = b \cdot x + bit \cdot \lfloor \tfrac{q}{2} \rfloor\}, where\ x \in \{0,1\}^n$$

iii) Decryption:

$$Dec(secKey, (u, u')) = \begin{cases} 0 & if\ u' - s\ \cdot u\ mod\ q \in [\frac{-q}{4}, \frac{q}{4}] \\ 1 & if\ u' - s\ \cdot u\ mod\ q \in (\frac{q}{4}, \frac{3q}{4}] \end{cases}$$

SVP $\leq$ CVP $\leq$ LWE [33, 34] describes that Learning with Errors problem is indirectly reducible from the SVP. As SVP has not been solved yet by the quantum algorithm, therefore LWE algorithm would also be secured against a quantum attacker. This algorithm also provides non-determinism as the error is augmented explicitly in the generation of public key $b = A \cdot s + e$.

3.2.2. Learning with Rounding [15]

Alternatively, a small value is added with $A \cdot s \in \mathbb{Z}_q$ to hide their correct value. This value is multiplied by p/q for some $p < q$. LWR problem would be hard if $q > p$ and it is a deterministic rounded version of $A \cdot s$. Therefore, in this case $b = \lfloor A \cdot s \rceil_p = \lfloor \frac{p}{q} \cdot A \cdot s \rceil_p$. For implementation, usually the floor of this value is taken. Here we are reducing the result from mod q to *mod* p. The other processes, like encryption and decryption, would remain same. For pertinent parameters, LWR is at least as hard as LWE and offers a worst-case assurance for LWR [35]. The upper hand of LWR is the elimination of an error sampling process which leads to improved efficiency. LWR-based cryptosystems could be used in symmetric cryptography because of their deterministic nature.

3.3. Quantum Cryptography [36]

Cryptography in the classical computer is a technique of secure communication and protecting information between two parties in a public domain through mathematical concepts and algorithms. In contrast, in a Quantum computer, cryptography is a technique that employs laws of quantum mechanics to secure communication and protection of information, Even intruders have their quantum computing.

3.4. Linear Feedback Shift Register (LFSR) [37]

A stream cipher method key and algorithm are applied to each bit to encrypt and decrypt the data. The LFSR could produce the key used in symmetric cipher.

An LFSR is a combination of shift registers in a sequential manner of bits with combinational logic in which output from a shift register is smartly orchestrated and fed back into its input to cause the function to limitlessly cycle through an array of patterns. Basically, the feedback shift register [38] could be divided into a shift register and a feedback function. Shift register is a series of bits. A shift register is called n bit shift register if it is n bit long and it is considered to be its length. All of them are shifted one bit to the right and the least significant bit, before shifting, become the output bit. The next leftmost bit is calculated as a function of other bits in the register. Generally, the least significant one-bit is the output of the shift register. The length of the output sequence before it begins repeating is called the period of a shift register.

An LFSR is the simplest feedback shift register that uses a linear function, usually XOR, as an input. The bits that change the state of LFSR are called tap bits. Tap bits could be connected in two ways namely Fibonacci method and Galois method. In Fibonacci method, taps are cascaded and fed into the leftmost shift register of LFSR. In Galois method each tap is XOR-ed with the output stream. LFSRs could be used in cryptography to generate nonces, pseudo-random numbers, etc.

3.5. Key Distribution using Quantum Channel

Quantum key distribution (QKD) [39] is a strategy for secure communication that allows the transfer of encryption keys known only to sharing parties. The QKD technique uses quantum physics-based characteristics to communicate a secret key in a verifiable and safe manner. QKD sends millions of polarized photons(light particles) from one end to another over a fiber optic channel. The quantum state of each photon is chosen at random, and all of the photons together make a stream of zeros and ones. When photons reach the terminal, the receiver uses horizontal, vertical, and diagonal beam splitters to ”read” their polarization. The receiver needs to predict which beam splitter to apply because it doesn't know which beam splitter was used for each photon at the sender side. The receiver then informs the sender which beam splitter he utilized for measurement and in what order. The sender then compares this information to the polarizer order used to send the photons. To produce an optical key that could be utilized for data encryption, the photons that the inaccurate beam splitter read are discarded.

4. Proposed Methodology

This study presents an approach towards generating pseudo-random numbers by taking advantage of the versatility of lattice-based cryptography. The proposed scheme chooses LBC (Lightweight Block Cipher) implementation, specifically Learning with Rounding Problem (LWR), for securing the seed against cryptanalysis. Figure 3 depicts structural scheme of proposed Pseudo-Random Number Generator. LWR obfuscates seed bits into a vector in a lattice, which is then fed to Linear Feedback Shift Registers (LFSR). A circuit (combination of three LFSRs) is computed to derive the random sequence. The most challenging aspect of this scheme is ensuring that the lattice vector remains in the lattice after the circuit generates the feedback sequence for itself. In addition, it needs to be ensured that the original seed/secret used in LWR encryption is not leaked. To tackle this, the present scheme uses a homomorphic function to compute the value for the circuit. Homomorphic functions are computed on encrypted data without disclosing the information of a secret seed.

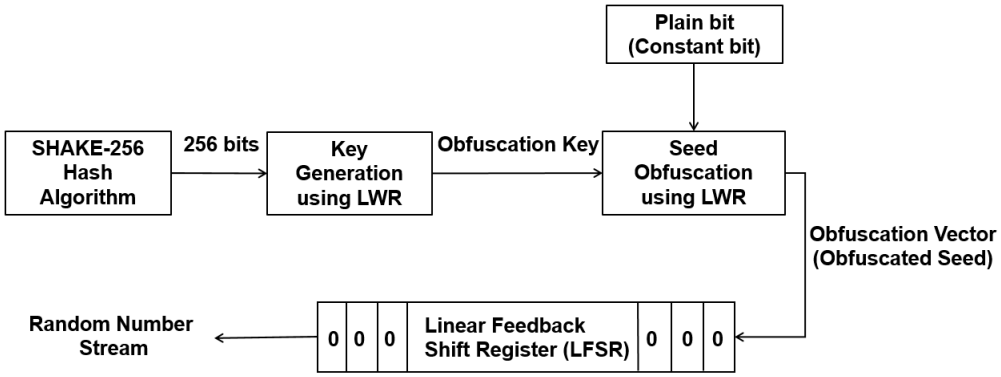


Figure 3: Structural Scheme of Proposed Pseudo-Random Number Generator

The scheme is implemented in three parts:

a) Seed Obfuscation using LWR

b) Homomorphic Function

c) Sequence Generation through LFSRs

a) Seed Obfuscation using LWR: Lattice-based cryptography has attracted a great deal of attention because of its provable worst-case hardness guarantees, resistance to quantum attacks unlike factoring and Discrete logarithm problems, easy implementation, and efficiency comparable to the current schemes.
Banerjee et al. [15] gave derandomized version of LWE which is used for determinism in lattice-based PRFs. The derandomized version of LWE is adopted as one of the central idea in LWR encryption to obfuscate the seed. Using the common attributes and seed, the other party involved in secure communication can generate the same pseudo-random sequence.
In this study an LWR obfuscation algorithm is designed for obfuscating an input seed. This has two stages, namely, Key Generation and LWR Obfuscation.

i) Key Generation (input_seed): According to the post-quantum cryptographic assumptions and guidelines, in the present work it has been assumed that a initial secret seed, which was used to launch the scheme, is exchanged through a quantum-safe channel (refer section 3.5) between two authorised parties without other parties learning its value.
Algorithm takes an input seed to generate the matrix $A \in R_q^{k\times k}$ and a secret $s \in R_q^{k\times 1}$ to generate obfuscation key $obs\_key = \lfloor A \cdot s \rceil_p$ where $\lfloor \cdot \rceil$ is rounding operation defined by taking all the coefficients generated by $A \cdot s$ and multiplying them to $\frac{p}{q}$ to introduce the rounding error. For the same, a function '$KeyGeneration(\alpha)$' is designed. Pseudocode is as follows:

1) $\alpha \leftarrow \{0,1\}^{256}$
2) $s \leftarrow S_\eta^l \in R_q^{k\times 1}$ where, k is rank of the module and l is the dimension of the lattice
3) $A \in R_q^{k\times k} := ExpandA(\alpha)$
4) $b^T = \lfloor A \cdot s \cdot \frac{p}{q} \rceil_p \in R_p^{k\times 1}$ where, prime $p <$ prime q
5) return $(obs\_key = b^T)$

The key generation function takes the input seed (α) from the SHAKE256 [40] hash function. SHAKE256 is the extendable output function (XOF), an upgrade of a hash function in cryptography. Rather than providing a fixed-length hash (e.g., 32 bytes like SHA256), it may create outputs of any length. The suffix "256" refers to the XOF's

security strength, not its output length. The seed is utilized to produce the secret key vector in the range of -2 to 2, controlled by η and to generate the matrix A. Initially, to generate matrix A, function squeezes many output blocks of SHAKE256 and SHAKE128 to provide adequate randomness and a high probability. By using A and s obfuscated key is generated by using above key generation function, whose value is restricted to $mod\ p$.

ii) Seed Obfuscation (obs_key, const_bit): The generated key is passed as an input to the Seed Obfuscation function to generate the obfuscation vector c. To do this, it first generates a random vector $x \in Z_q^{k\times 1}$. Next it generates $c = b^T \cdot x + const\_bit \cdot q/2$. The $const\_bit$ is used to add extra entropy to the obfuscated seed. The algorithm returns c, an obfuscated seed as a vector.

This completes the LWR obfuscation of a seed, where the scheme generates an obfuscated vector by using an original seed and extra randomness. This obfuscated seed, $'c'$, will be passed to the LFSR circuit to apply the homomorphic function. Pseudocode is as follows -

1) $A \in R_q^{k\times k} := ExpandA(\alpha)$, where α is the seed vector
2) $x \leftarrow \{0,1\}^{256}$
3) $c = [b^T \cdot x + const\_bit \cdot \lfloor q/2 \rfloor]$
4) return (c)

b) Homomorphic Function: Homomorphic encryption allows operations on encrypted data with the goal of preserving secrecy after evaluating them. Homomorphic encryption has proven to be among most significant applications of lattice-based cryptography. This study uses a homomorphic function to design circuits to compute on encrypted data in order to generate sequences without losing the lattice properties.

The proposed QSPRNG generator uses distributed PRFs on $\mathbb{Z}_q$ vector along with $\circ$ 'a binary' operation. The circuit takes input from the LWR obfuscation algorithm $'c'$ and divides it into three parts such that the operations obey the following circuit equality:

$$Homo\_LWR(c) = Homo\_LWR(c_1) \circ Homo\_LWR(c_2) \circ Homo\_LWR(c_3)$$

where $Homo\_LWR(c)$ is the function that LFSR obeys and does not hamper the lattice properties of LWR, and $\circ$ is any arbitrary binary operation.

We would prove the following theorem to ensure that the circuit preserves the structure of LWR after homomorphic transformation.

Theorem 4.1. *Given a circuit $Homo\_LWR$ with operation $\circ$ the evaluation on the LWR obfuscated vector c follows the equality below:*

$$Homo\_LWR(c) = Homo\_LWR(c_1) \circ Homo\_LWR(c_2) \circ Homo\_LWR(c_3)$$

Where c_1, c_2, c_3 are distributed seeds obtained from the LWR Obfuscation algorithm.

Proof: *The LWR function is $c = b \cdot x + const\_bit \cdot q/2$. Let the divided output c be c_1, c_2, c_3 where c_1, c_2,and c_3 acts as inputs to each individual circuit function $Homo\_LWR(c)$ with $\circ$ operation.*

Dividing c results into three parts. We get the following equations:

$$c_1 = b_1 \cdot x_1 + const\_bit_1 \cdot q/2 \tag{1}$$

$$c_2 = b_2 \cdot x_2 + const\_bit_2 \cdot q/2 \tag{2}$$

$$c_3 = b_3 \cdot x_3 + const\_bit_3 \cdot q/2 \tag{3}$$

Since $b = \lfloor A \cdot s \rfloor_p$, equations (1), (2), and (3) are:

$$c_1 = b_1 \cdot x_1 + const\_bit_1 \cdot \tfrac{q}{2} = \lfloor A_1 \cdot s_1 \rfloor_p \cdot x_1 + const\_bit_1 \cdot \tfrac{q}{2}$$

$$c_2 = b_2 \cdot x_2 + const\_bit_2 \cdot \frac{q}{2} = \lfloor A_2 \cdot s_2 \rfloor_p \cdot x_2 + const\_bit_2 \cdot \frac{q}{2}$$

$$c_3 = b_3 \cdot x_3 + const\_bit_3 \cdot \frac{q}{2} = \lfloor A_3 \cdot s_3 \rfloor_p \cdot x_3 + const\_bit_3 \cdot \frac{q}{2}$$

The function $Homo\_LWR(c)$ is as follows:

$$Homo\_LWR(c) = Homo\_LWR(c_1) \circ Homo\_LWR(c_2) \circ Homo\_LWR(c_3)$$

$$\begin{aligned} b \cdot x + const\_bit \cdot \frac{q}{2} = {} & b_1 \cdot x_1 + const\_bit_1 \cdot \frac{q}{2} + \\ & b_2 \cdot x_2 + const\_bit_2 \cdot \frac{q}{2} + \\ & b_3 \cdot x_3 + const\_bit_3 \cdot \frac{q}{2} \end{aligned}$$

$$\begin{aligned} \lfloor A \cdot s \rfloor_p \cdot x + const\_bit \cdot \frac{q}{2} = {} & \lfloor A_1 \cdot s_1 \rfloor_p \cdot x_1 + const\_bit_1 \cdot \frac{q}{2} + \\ & \lfloor A_2 \cdot s_2 \rfloor_p \cdot x_2 + const\_bit_2 \cdot \frac{q}{2} + \\ & \lfloor A_3 \cdot s_3 \rfloor_p \cdot x_3 + const\_bit_3 \cdot \frac{q}{2} \end{aligned}$$

$$\begin{aligned} \lfloor A \cdot s \rfloor_p \cdot x + const\_bit \cdot \frac{q}{2} = {} & \lfloor A_1 \cdot s_1 \rfloor_p \cdot x_1 + \lfloor A_2 \cdot s_2 \rfloor_p \cdot x_2 + \lfloor A_3 \cdot s_3 \rfloor_p \cdot x_3 + \\ & const\_bit_1 \cdot \frac{q}{2} + const\_bit_2 \cdot \frac{q}{2} + const\_bit_3 \cdot \frac{q}{2} \end{aligned}$$

Since the value of $A_1 \cdot s_1 \cdot x_1$, $A_2 \cdot s_2 \cdot x_2$, $A_3 \cdot s_3 \cdot x_3$ are single vectors, and when all of these vectors are concatenated, they form the same vector $A \cdot s \cdot x$. Similar is the case for $const\_bit_1 \cdot q/2$, $const\_bit_2 \cdot q/2$, and $const\_bit_3 \cdot q/2$

Hence, it is proved that the circuit preserves the structure of LWR after homomorphic transformation.

c) Sequence Generation through LFSRs: As discussed above, the pseudo-random sequences are generated by utilising the LWR-based homomorphic function. The proposed scheme takes three LFSRs and implements the distributed Homomorphic circuit using the obfuscated seed.
The LFSR sequence generator comprises three stages:

i) Initialization of LFSRs

ii) Bit output sequence Generation

iii) Feedback Generation

i) Initialization of LFSRs:

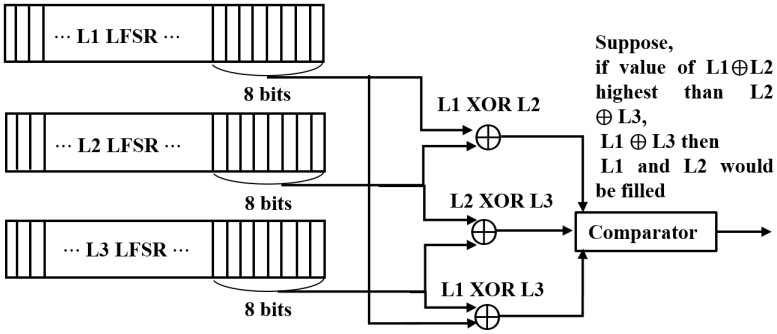


Figure 4: Instance of Initialization of LFSR

Three LFSRs L1, L2 and L3 having 2000 bits each are considered during design. The LWR obfuscation function generates obfuscated seed of 6000 bits to initialize all bits. Each LFSR is initialized with the obfuscated seed in a non-sequential fashion. For the initialization of LFSRs, an iterative process is used. In the first iteration, eight bits are filled in each LFSR sequentially. After that eight bits of each LFSRs are XOR-ed with each other like L1⊕L2, L2⊕L3, L3⊕L1, the pair of LFSR with highest XOR-ed value is found out. The next iteration fills a pair with the highest XOR-ed value. The same process is repeated until two LFSRs are completely filled. This is completely a non-sequential procedure (as shown in Figure 4). Finally, the remaining third LFSR is filled in a sequential manner with the remaining obfuscated seed bits.

Pseudo Code for initialization of LFSR -

a) I/P: Three LFSRs (L1, L2 and L3 having 2000 bits each) would be initialized
b) Obfuscated seed $[c]_{6000}$
c) Fill the first eight bits of all three LFSRs using $[c]$
d) Do
   - L1 $\oplus$ L2 $\rightarrow$ R1
     L2 $\oplus$ L3 $\rightarrow$ R2
     L1 $\oplus$ L3 $\rightarrow$ R3

Compare R1, R2, R3.
○ The LFSR pair which would have the highest value would be filled

e) Repeat step (d) until two LFSRs are completely filled
f) Fill the pending third LFSR with the remaining bits of $[c]$

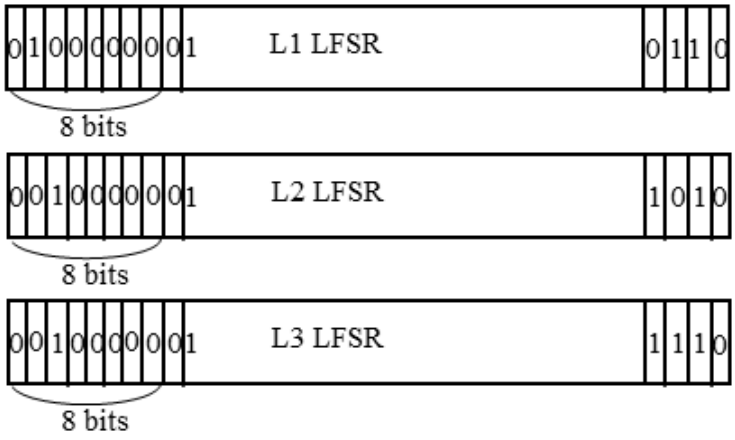
In the above pseudo code recently filled eight bits of LFSR L1, L2, and L3 are XOR-ed with each other. These XOR-ed bits are compared with each other, and the LFSR pair with the highest value is filled.

ii) Bit output sequence Generation:

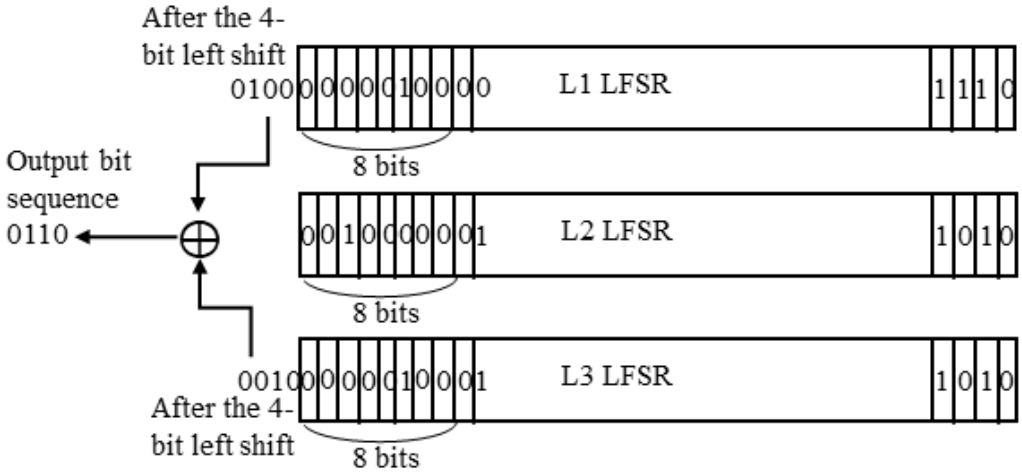
After initializing all three LFSRs, the output sequence is generated according to the state Master-Slave LFSRs. The scheme initially chooses one LFSR, L2, as Master LFSR and the other two say L1 and L3 as Slave LFSR. Initially, all the bits in every LFSR are divided into eight-bit blocks. In a first iteration, the first block of Master LFSR L2 is selected and location of the ones in that block is found. According to the location of ones in a selected block of L2, the left shift for slave LFSR L1 and L3 is performed. For example, if one is found on location two in the selected block of Master LFSR L2, then a number of shifting bits is computed as $2^2 = 4$ bits. Therefore, L1 and L3 is shifted four bits left. These four bits of LFSR L1 and L3 would be XOR-ed further and the resulting four bits would be the output as shown in Figure 5b.

Pseudo Code for Bit Output Sequence Generation -

a) Divide the 2000 bits of each LFSR into eight-bit blocks
b) Choose L2 LFSR as a master and L1 and L3 as a slave by default for first iteration
c) Find out the location of all ones in the selected block of Master LFSR. Shifting would be performed according to the location of one
d) For every $2^{i^{th}}$ shifting, XOR the $2^{i^{th}}$ output bits of slaves L1 and L3. Resultant XOR-ed bits would be the output
e) Repeat this process for every $2^{i^{th}}$ shift
f) Repeat the steps (c), (d) and (e) for all the eight-bit blocks of the



(a) Initial instance of Bit Output Sequence Generation of LFSR



(b) 4-bit left shift operation is performed as 1 is at 2^{nd} location as indexing starts from 0

Figure 5: Bit Output Sequence Generation

respective Master LFSR

Master Node Selection –
When all the bits of a selected block for the present LFSR are consumed, the selection of Master LFSR for the next iteration is required.

a) Compute the value of left-shifted bits for each slave LFSR
b) If value for the shifted bits is higher than the other, then the corresponding LFSR would become Master
c) Else the present LFSR would remain master for the next iteration

As Figure 5b is showing left shifted bits for slave LFSR L1 is 0100 (4 in decimal) and left shifted bits for slave LFSR L3 is 0010 (2 in decimal). If we compare these shifted values, the highest value corresponds to LFSR L1; therefore, L1 would be the master for the next iteration.

iii) Feedback Generation: To generate outputs for arbitrary number of bits, the LFSR sequence must be fed continuously to prevent it from exhausting. This scheme uses the output bits and the current byte to provide feedback to the LFSRs. The left-shifted bits from the two slave LFSRs produced in bit output sequence generation are XOR-ed with the same number of bits present in the selected block of master LFSR. These XOR-ed bits are fed into the slave LFSRs in a cross manner. For example, if L2 is master and L1, L3 are slave LFSRs then XOR-ed bits of L1and L2 would be fed in to L3 and XOR-ed bits of L2 and L3 would be fed into L1 as shown in Figure 6.

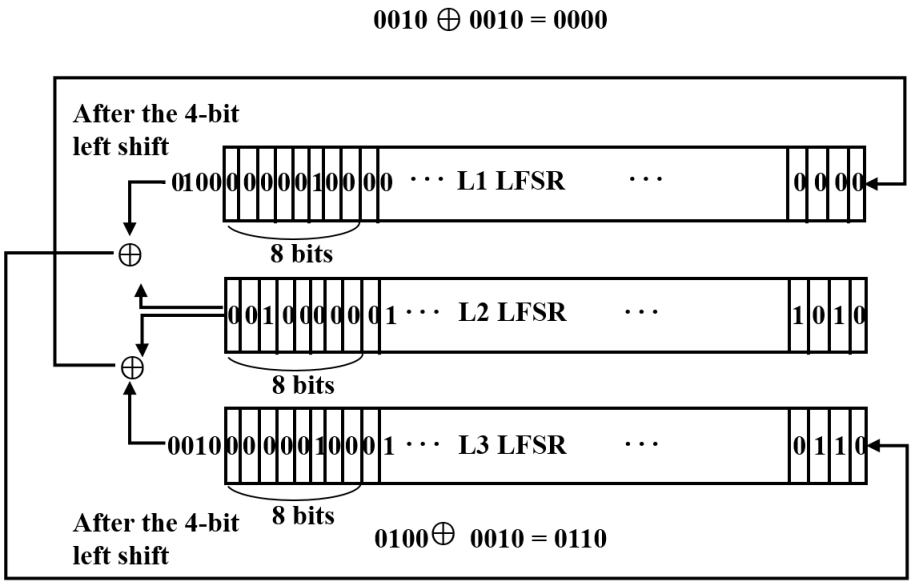


Figure 6: Instance of Feedback Generation

Pseudo Code for Feedback Generation -

a) Consider L2 LFSR as the Master
For every 2^i shift XOR-ed the 2^i shifted bits of L1 and L3 with first 2^i bits of L2
   - $L1\_feedback = L2\_2^i \oplus L1\_shifted$
   - $L3\_feedback = L2\_2^i \oplus L3\_shifted$
   - $L1\_feedback\_bits = L3\_feedback$
   - $L3\_feedback\_bits = L1\_feedback$

b) Repeat this process for every $2^{i^{th}}$ shift

c) Repeat steps (a) and (b) for all the eight-bit blocks of the respective Master LFSR

5. Security

The lattice-based cryptosystem is among the most promising quantum secure candidates [41]. The present scheme is focused on securing the seed by obfuscating it through a Lattice-based function, specifically LWR. The LWR problem is a derandomized form of LWE that has been proven to be as secure as LWE. Therefore, current scheme also relies on the security of the LWE problem and is as

hard as worst-case lattice problems [35]. However, for the construction of cryptographic primitives like PRNG, major challenge is to keep the function deterministic, and direct construction of PRNGs through noisy schemes like LWE results in weak PRFs and PRNGs. This work uses the direct construction by Banerjee et al. [15], a deterministic lattice-based PRF through the LWR problem. Subsequently, with appropriate parameters, it turns out that the decision LWR problem is at least as hard as the decision LWE.

In the proposed scheme, the original seed for PRNG is obfuscated by generating LWR parameters A and s through another seed. The resulting key $b = \lfloor A \cdot s \rfloor_p$ is used to generate the obfuscated seed $c = b \cdot x + bit \cdot q/2$ where x is randomly generated vector and bit is a constant $(0/1)$. There is a high probability that b is not close to some values in Z_q having a different rounding value [15]. That means the occurrence of such bad instances is negligible. This proves that the adversary can distinguish between LWR sequences with a very small probability.

The security with LWR construction is maintained even after the circuit evaluation in LFSR. We have proved above that the homomorphic function used in the circuit retains the lattice structure; hence, the security would lie under LWR.

6. Result and Analysis

6.1. Randomness Analysis

Specific tests are used to compare the characteristics of an obtained binary sequence to those of a truly random sequence to determine its randomness. DIEHARD test [42], ENT test [43], NIST test [41], Donald Knuth's tests [44], and CryptXS suite [44] are among the most popular and certified tests. Each test suite, as mentioned above, includes its own set of tests that identify a specific type of non-randomness for a given input. NIST SP 800-22 suite, ENT test suite, and DIEHARD test suite are used to test the randomness of the bit sequences generated by the proposed QSPRNG. Test results and analysis of the proposed PRNG are discussed in the subsequent sections.

6.1.1. NIST SP 800-22

NIST SP 800-22 test suite has been used to check if the generated bit sequence from the QSPRNG adhered to the appropriate statistical features.

We have performed NIST's 15 tests on the proposed Pseudo-Random Number $(0/1)$ Generator. These tests are performed in a Secure Systems Lab, CSE Department, DIAT, Pune, on CDAC-PARAM Shavak machine. We buffered the 1.1 GB output bits in a file, and then these buffered bits are used as an input to NIST test to provide 10^6 bits for every 50 sample to test the randomness produced by the generator. The parameters we have chosen for specific tests are depicted in Table 1, and the results are presented in Table 2. The overall result of the sequences in each test category was found to be above the minimum probability value, and all statistical characteristics of randomness tests were successfully passed.

Table 1: Parameters chosen for the respective tests

Sr. No.	**Test**	**Block length (bits)**
1	Block Frequency Test	128
2	Non-Overlapping Template Test	8
3	Overlapping Template Test	8
4	Approximate Entropy Test	7
5	Serial Test	10
6	Linear Complexity Test	500

Table 2: NIST test results for the proposed PRNG

Sr. No.	**Test**	**Statistic and Reference Distribution**	**Proportion**	**Accepted/ Rejected**
1	Frequency (Mono-bit) Test	Half normal distribution	48/50	Accepted
2	Frequency Test within a Block	χ^2 distribution	47/50	Accepted
3	Runs Test	χ^2 distribution	48/50	Accepted
4	Tests for the Longest Run of Ones in a Block	χ^2 distribution	47/50	Accepted

5	Binary Matrix Rank Test	χ^2 distribution	49/50	Accepted
6	Discrete Fourier Transform (Spectral) Test	Normal distribution	50/50	Accepted
7	Non-overlapping Template Matching Test	χ^2 distribution	48/50	Accepted
8	Overlapping Template Matching Test	χ^2 distribution	48/50	Accepted
9	Maurer's "Universal Statistical" Test	Half normal distribution	47/50	Accepted
10	Linear Complexity Test	χ^2 distribution	50/50	Accepted
11	Serial Test	χ^2 distribution	47/50	Accepted
12	Approximate Entropy Test	χ^2 distribution	47/50	Accepted
13	Cumulative Sums Test	Normal distribution	48/50	Accepted
14	Random Excursions Test	χ^2 distribution	23/23	Accepted
15	Random Excursions Variant Test	Half normal distribution	23/23	Accepted

The minimum pass rate for each statistical test except for the random excursion (variant) test, is approximately 47 for a sample size of 50 binary sequences.

The minimum pass rate for random excursion (variant) test is approximately 21 for a sample size of 23 binary sequences.

Table 2 explicitly shows that the proposed QSPRNG has passed all NIST tests as the proportion value for each statistical test meets the minimum criteria limit to pass statistical tests.

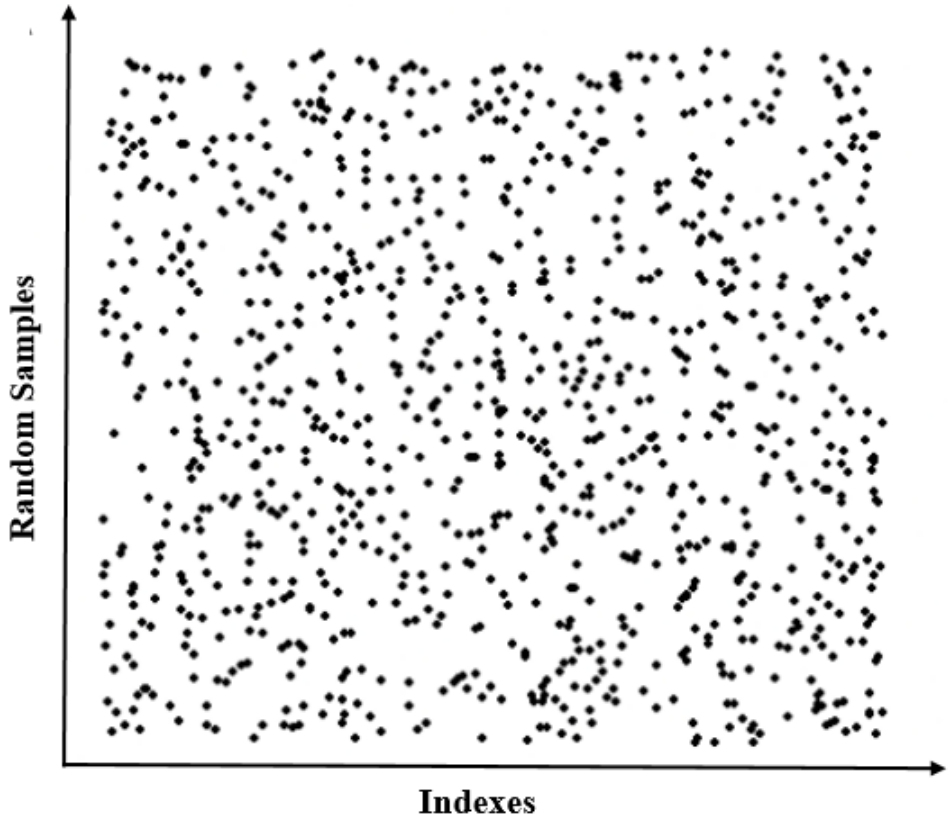


Figure 7: Scattered Graph for proposed PRNG

The Pseudo-Random Number Generator's scattered graph in Figure 7 is an

embedding function that transforms the random numbers produced by the PRNG to 3-bit indexes 0 to 7. The graph demonstrates that the produced random numbers are spread uniformly over this space without cluster formation.

6.1.2. DIEHARD test suite

Diehard tests include 16 statistical tests. It is a complete, detailed, and comprehensive collection of statistical tests for PRNG. This set of statistical tests represents a form of litmus test for verification and eventual certification of PRNG. The majority of DIEHARD's tests give a p-value that ought to be consistent within the domain [0,1), if such input data contain a completely unbiased sequence of random bits.

The proposed PRNG was evaluated against a Die-Harder Test suite to gain insight into the entropy it generated with a sample size of 10000. The diehard test result for the proposed QSPRNG is shown in Figure 8

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
        mt19937|/home/shavak/Testing/LFSR_stream.txt|  8.53e+07  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.83675154|  PASSED
      diehard_operm5|   0|   1000000|     100|0.75072354|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.43911957|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.98619761|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.18829639|  PASSED
        diehard_opso|   0|   2097152|     100|0.80800721|  PASSED
        diehard_oqso|   0|   2097152|     100|0.19678399|  PASSED
         diehard_dna|   0|   2097152|     100|0.68956457|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.77747400|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.79021841|  PASSED  |
 diehard_parking_lot|   0|     12000|     100|0.72169377|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.54837032|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.92768451|  PASSED
     diehard_squeeze|   0|    100000|     100|0.80269320|  PASSED
        diehard_sums|   0|       100|     100|0.10427555|  PASSED
        diehard_runs|   0|    100000|     100|0.23565737|  PASSED
        diehard_runs|   0|    100000|     100|0.94278102|  PASSED
       diehard_craps|   0|    200000|     100|0.03939797|  PASSED
       diehard_craps|   0|    200000|     100|0.25053056|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.98164326|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.16171353|  PASSED
         sts_monobit|   1|    100000|     100|0.37386094|  PASSED
            sts_runs|   2|    100000|     100|0.66288541|  PASSED
          sts_serial|   1|    100000|     100|0.09689069|  PASSED
         rgb_bitdist|   1|    100000|     100|0.04381975|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.57560095|  PASSED
    rgb_permutations|   5|    100000|     100|0.28117934|  PASSED
      rgb_lagged_sum|  32|   1000000|     100|0.32349249|  PASSED
     rgb_kstest_test|   0|     10000|    1000|0.88261871|  PASSED
     dab_bytedistrib|   0|  51200000|       1|0.64368890|  PASSED
             dab_dct| 256|     50000|       1|0.67078071|  PASSED
        dab_filltree|  32|  15000000|       1|0.61501088|  PASSED
       dab_filltree2|   1|   5000000|       1|0.89464634|  PASSED
        dab_monobit2|  12|  65000000|       1|0.92036656|  PASSED
```

Figure 8: Diehard Test Results for Proposed PRNG

As per achieved results, the proposed QSPRNG passed all 16 test and has appropriate statistical behavior for generating pseudo-random numbers.

6.1.3. ENT test suite

ENT test examines the PRNGs using five tests. The test suite runs on byte sequences and provides distinct findings. The entropy test of this suite assesses

the information density of a data file, which should be 8. The chi-square test verifies data randomness. This test is error-sensitive. File bytes minus file length is the arithmetic mean test. The optimal score for this test is 127.5. The Monte Carlo test calculates a square-enclosed circle using pseudo-random integers. The Serial correlation coefficient (SCC) test compares the current byte to the previous byte. Zero is the finest. The result of the ENT test suite for the proposed QSPRNG is shown in Table 3.

Table 3: ENT test results

Sr. No.	Test Name	Value	Result
1	Entropy	7.899999 bits/byte	Pass
2	Serial-Correlation	Correlation coefficient = -0.000044	Pass
3	Chi-square distribution	490541141.899390	Pass
4	Monte-Carlo-Pi	3.1325	Pass
5	Arithmetic mean	127.4998	Pass

6.1.4. Frequency Test

Computing the percentage of 0s and 1s throughout the whole sequence is the purpose of the frequency test [41]. This test aims to see if a sequence's count of 0s and 1s is close to what would be required in a truly random sequence. In this test, 1s and 0s in the input sequence are translated to $+1$ and -1 values, respectively, then summed together to obtain S_n. Then the test statistic S_{obs} is calculated as $S_{obs} = |S_n| \div \sqrt{Bit\ \ string\ \ length}$. The complementary function of error erft() is used to calculate $p - value = erft(s_{obs}) \div \sqrt{2}$. If the calculated $p - value \geq 0.01$, the sequence is considered as random else, the sequence is considered as non-random.
The frequency test result for the proposed QSPRNG is shown in Figure 9.

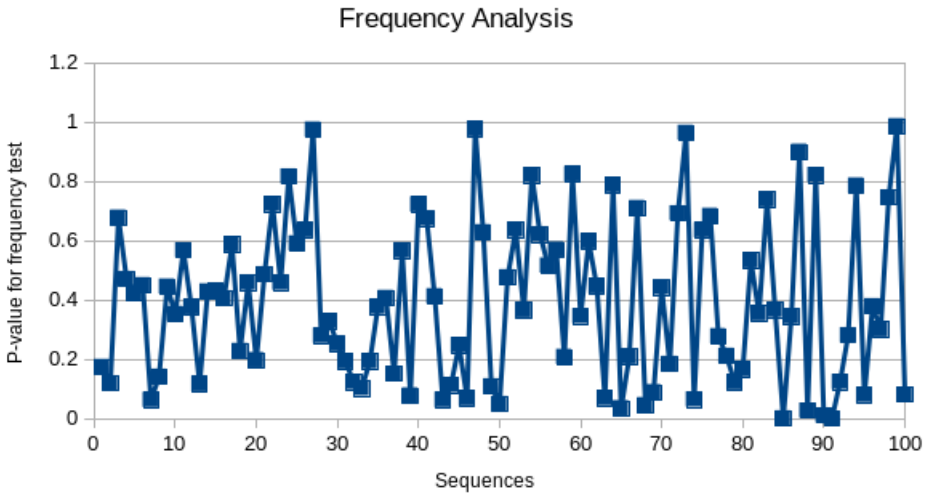


Figure 9: Frequency Test for Sequences Generated using Proposed PRNG

Since the P-value obtained for the generated 100 sequences is ≥ 0.01, the proposed QSPRNG has passed the frequency test.

6.2. Security Analysis

6.2.1. Key Space Analysis

The key-space of a PRNG technique is the set of possible keys that can be used to produce random bits by utilizing particular PRNG approach. The total number of bits in key, also known as the key length, is used to calculate the size of the key space. The size of the key space is equal to the key length multiplied by the number of possible bits for each place on the key. In the proposed QSPRNG, the initial seed of the LFSR is generated using lattice-based hard problem LWR, which is computed under n-dimensional euclidean space in modulus p. The input for the LWR encryption has been taken from Shake256. The output generated from Shake256 has key space is equal to 2^{256}. Hence the key space for proposed generator is 2^{256}, which is large number.

6.2.2. Key Sensitivity Analysis

The key sensitivity analysis determines how much a change in the PRNG's key bits, changes the output sequence and generates a totally distinct sequence. As per the Kerckhoff's principle [45], a cryptography system's security should

depend only on its keys. Bad choice of key can make the whole encryption system unsafe.
In the proposed QSPRNG, the key bits are diffused uniformly by using LWR. As per the LWR assumption, it generates uniformly distributed LWR samples that are computationally indistinguishable from original key values. In order to develop simple and effective pseudo-random generators and functions, the LWR principle has been applied. So the very slight change in the input key will affect the generated sequence. We test the proposed QSPRNG algorithm's high sensitivity to secret key by employing slightly different keys. The bit change rate (BCR) test [9] is used to analyze the key sensitivity of the pseudo-random number generator. It is defined as follows: For two sequences, X and Y,

$$Bit\ Change\ Rate(X,Y) = \frac{Number\ of\ distinct\ bits\ in\ X\ and\ Y\ sequences}{length\ of\ a\ longest\ sequence} \times 100 \tag{4}$$

The output of the BCR equation 4 should ideally be close to fifty percent [9]; if it is fifty it shows that the two sequences are different. Figure 10 shows the BCR for the 500 output sequences generated using the proposed PRNG with different 500 keys (with only one bit different from the first key).

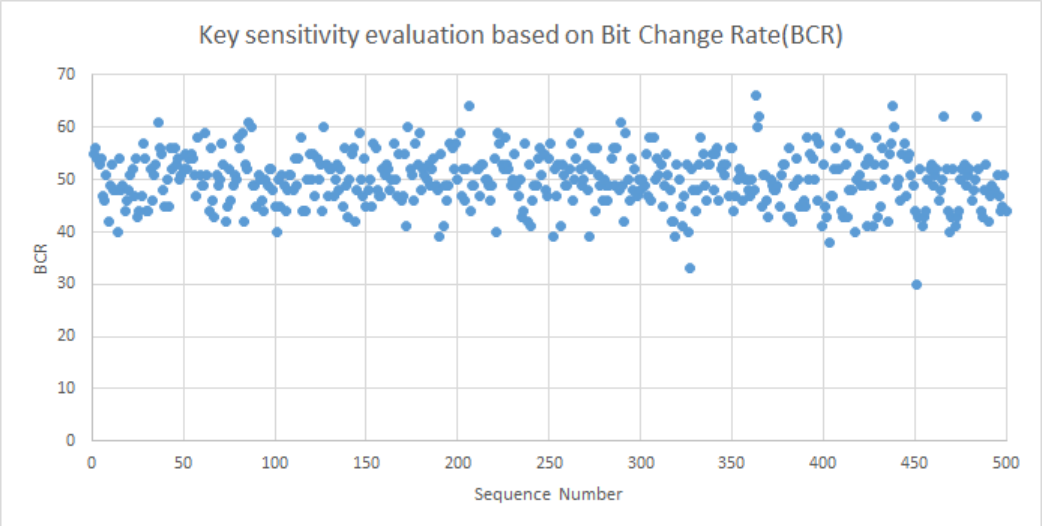


Figure 10: Key Sensitivity Evaluation based on Bit Change Rate

The scatter graph in Figure 10 shows that the BCR of the sequences changes between 40 to 60. The average BCR value is equal to 50.012 which shows that the proposed PRNG is very sensitive to small changes in a key bit.

6.2.3. Correlation Analysis

The test of lagged correlations [42] checks the possibility of the random number generator having a bit-level correlation after a certain number of bits. As a result, the lagged sums test is extremely simple. Add up the uniform deviates sampled from the random numbers, ignoring the lag samples between each rand used. The mean of the tsamples samples should be 0.5*tsamples. sqrt(tsamples/12) should be the standard deviation. The experimental sum values are thus converted into p-values (through the use of erf()), and a ks-test is applied to psamples of them. The correlation analysis for the proposed QSPRNG is shown in Figure 11.

```
#=============================================================================#
#                          Histogram of test p-values                         #
#=============================================================================#
# Bin scale = 0.100000
#     20|    |    |    |    |    |    |    |    |    |    |
#       |****|    |    |    |    |    |    |    |    |    |
#     18|****|    |    |    |    |    |    |    |    |    |
#       |****|    |    |    |    |    |    |    |    |    |
#     16|****|    |    |    |    |    |    |    |    |    |
#       |****|    |    |    |    |    |    |    |    |    |
#     14|****|****|    |    |    |    |    |    |    |    |
#       |****|****|    |    |    |    |    |    |    |    |
#     12|****|****|    |    |    |****|    |    |    |    |
#       |****|****|    |    |    |****|    |    |    |    |
#     10|****|****|    |    |****|****|    |    |****|    |
#       |****|****|    |    |****|****|****|    |****|    |
#      8|****|****|    |    |****|****|****|****|****|    |
#       |****|****|    |****|****|****|****|****|****|    |
#      6|****|****|    |****|****|****|****|****|****|****|
#       |****|****|****|****|****|****|****|****|****|****|
#      4|****|****|****|****|****|****|****|****|****|****|
#       |****|****|****|****|****|****|****|****|****|****|
#      2|****|****|****|****|****|****|****|****|****|****|
#       |****|****|****|****|****|****|****|****|****|****|
#       |-------------------------------------------------
#       | 0.1| 0.2| 0.3| 0.4| 0.5| 0.6| 0.7| 0.8| 0.9| 1.0|
#=============================================================================#
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
      rgb lagged sum|   0|   1000000|     100|0.03170292|  PASSED
```

Figure 11: Correlation Analysis

According to the preceding histogram, the random sequence generated by the proposed PRNG has an auto correlation coefficient near to zero since it passes the lagged sum test.

To compute the correlation coefficient, we build pseudo-random sequences using

different keys. The following equation is used to compute the Pearson's correlation coefficient.

$$Correlation_{(X,Y)} = \frac{N\sum XY - (\sum X \sum Y)}{\sqrt{[N\sum x^2 - (\sum x)^2][N\sum y^2 - (\sum y)^2]}} \tag{5}$$

When the correlation value is +1, all data points lie on a line where Y increases as X increases, and vice versa when the correlation value is -1. A value is 0 means that the two sequences do not depend on each other in a linear way. A correlation shown in the Figure 12 is between a pseudorandom sequence created by the key "4d69fe35f9fcfa998d7c9a3c747317227516d785d264c38f41de4f494c8c7" & 500 pseudorandom sequences generated by 500 random keys. The average correlation coefficient is 0.00576673. The uniform 500 test results are approximately equal to zero, confirming no correlation among the pseudorandom sequences generated by the proposed technique.

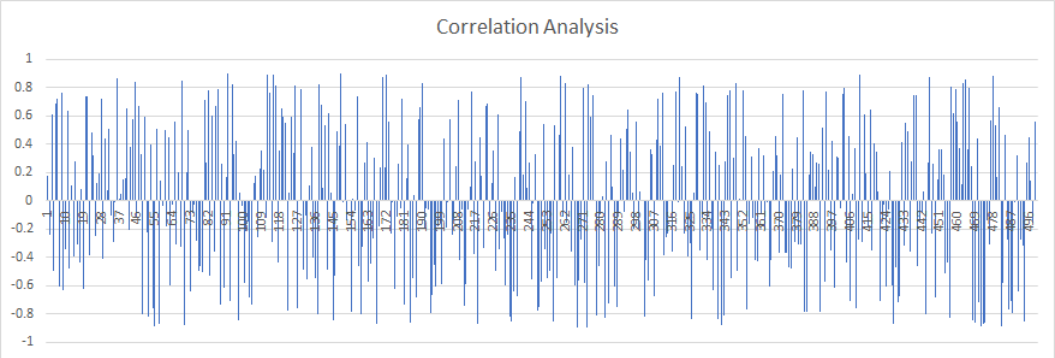


Figure 12: Correlation Analysis for 500 Sequences

7. Comparison with other PRNGs

This Section compares the proposed PRNG to alternative approaches regarding randomness, security tests, and performance. The approach proposed in this research is based on a lattice-based hard problem and LFSR, which can generate a random bit sequence. The advantage of the current scheme over the other PRNG schemes is that the proposed generator has acceptable statistical features and its behavior is entirely random. Also, the additional statistical and security analysis of the proposed PRNG scheme has been reported here. The important advantageous feature of the proposed PRNG scheme is that it is based on PQC primitive that provides post-quantum security. Table 4 compares the suggested strategy to

other comparable approaches. The table compares the proposed PRNG to other ways using various criteria and features of various approaches for examining the proposed method. Table uses criteria such as statistical testing suits, key sensitivity analysis, histogram, correlation analysis, frequency test, speed analysis, and quantum safety to analyze PRNG. Table illustrates the assessed features of several approaches, and the number of characteristics examined in every approach is recorded. According to the data, it is evident that the proposed PRNG analyzed the most tests.

Table 4: Comparison of the proposed PRNG with other PRNG schemes

Features	[46]	[47]	[48]	[10]	Proposed PRNG
Implementation based on	Memristor + LFSR	LFSR	Avalanche noise	Chaotic map	Lattice based hard problem
NIST test suite	✓	✓	✓	✓	✓
DIEHARD test suite	-	-	-	-	✓
ENT test suite	-	-	✓	-	✓
Key space analysis	-	-	-	✓	✓
Key sensitivity analysis	-	-	-	✓	✓
Histogram	-	-	-	✓	✓
Correlation Analysis	-	-	✓	✓	✓
Frequency Test	✓	-	✓	✓	✓
Speed Analysis	-	✓	-	✓	✓
Quantum Safe	-	-	-	-	✓

Note: ✓means "achieved" and - means there is no result reported.

7.1. Speed

The proposed QSPRNG's speed was tested on an Intel(R) Core(TM) i7-9700 hardware system with 8 GB of RAM and implemented in c language. The results

are shown in Table 5. The proposed PRNG generates 35.089 Mbit/second. To test the speed of current PRNG a software-based approach, time stamping method [49, 50], has been used.

Table 5: Speed comparison of proposed PRNG with other PRNG schemes

Sr. No.	PRNG	Running Speed in Mbit/s
1	[46]	1.5
2	[47]	17.066
3	[48]	0.0081
4	[10]	1.7
5	Proposed PRNG	35.089

8. Conclusion

This scheme attempts to secure the seed using lattice-based primitives, precisely a LWR problem. It generates an indefinite random sequence through LFSR and homomorphic function, which guarantees the theoretical security requirement of the LWR problem. The proposed QSPRNG's randomness was successfully tested by NIST, DIEHARD, and ENT test suites. For security aspects for cryptographic purposes, security analysis is reported, such as key space, key sensitivity, and correlation coefficients analysis. The performance analysis of pseudo-random numbers generation is done using speed and frequency test. It validates and justifies our QSPRNG's capabilities. The proposed approach was compared against existing PRNG methods. The assessment findings show that the proposed QSPRNG has high performance, acceptable statistical features, pseudo-random sequence production unpredictability, and the capacity to offer security in post-quantum period as it is based on lattice-based problems.
In future, the present work can be extended by shifting the lattice using a short random vector with uniform distribution. A uniformly random shifting will help to generate more random sequences. Also, parallelization of the code can be done to increase the bit generation rate.

Competing interests

The authors declare that they have no known competing interests.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Data Availability Statement

Implemented code and data related to results are available with authors.

Author's contributions

Anupama Arjun Pandit: Conception and design, Implementation of the supporting computer code, Analysis and interpretation of the data, Writing-original draft, Review & editing. **Atul Kumar:** Programming, Review & editing. **Dr. Arun Mishra:** Supervised, Review & editing.

Acknowledgments

This paper and the research behind it would not have been possible without the exceptional support of Dr. Ajay Misra, Dr. Odelu Ojjela, and Mr. Nilesh Kolhalkar. Their enthusiasm, knowledge, and exacting attention to detail have been an inspiration and kept our work on track. We would like to thank all of them. They are the ultimate role models.

References

[1] W. Diffie, M. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (6) (1976) 644–654. doi:10.1109/TIT.1976.1055638.

[2] P. Gallagher, Digital signature standard (dss), Federal Information Processing Standards Publications, volume FIPS 186 (2013).

[3] R. L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM 21 (2) (1978) 120–126. doi:10.1145/359340.359342.
URL https://doi.org/10.1145/359340.359342

[4] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comput. 26 (5) (1997) 1484–1509. doi:10.1137/S0097539795293172.
URL https://doi.org/10.1137/S0097539795293172

[5] J. Kelsey, B. Schneier, N. Ferguson, Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator, in: International Workshop on Selected Areas in Cryptography, Springer, 1999, pp. 13–33.

[6] M. R. Murray, An implementation of the yarrow {PRNG} for {FreeBSD}, in: BSDCon 2002 (BSDCon 2002), 2002.

[7] N. Ferguson, B. Schneier, T. Kohno, Generating randomness, Cryptography Engineering: Design Principles and Practical Applications (2015) 135–161.

[8] J. K. Salmon, M. A. Moraes, R. O. Dror, D. E. Shaw, Parallel random numbers: as easy as 1, 2, 3, in: Proceedings of 2011 international conference for high performance computing, networking, storage and analysis, 2011, pp. 1–12.

[9] M. Jafari Barani, P. Ayubi, M. Yousefi Valandar, B. Y. Irani, A new pseudo random number generator based on generalized newton complex map with dynamic key, Journal of Information Security and Applications 53 (2020) 102509. doi:https://doi.org/10.1016/j.jisa.2020.102509.
URL https://www.sciencedirect.com/science/article/pii/S2214212619309512

[10] M. Murillo-Escobar, C. Cruz-Hernández, L. Cardoza-Avendaño, R. Méndez-Ramírez, A novel pseudorandom number generator based on pseudorandomly enhanced logistic map, Nonlinear Dynamics 87 (1) (2017) 407–425. doi:doi:10.1007/S11071-016-3051-3.

[11] L. Blum, M. Blum, M. Shub, A simple unpredictable pseudo-random number generator, SIAM Journal on computing 15 (2) (1986) 364–383.

[12] L. K. Grover, A fast quantum mechanical algorithm for database search, in: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, 1996, pp. 212–219.

[13] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, J. F. Dray Jr, et al., Advanced encryption standard (aes) (2001).

[14] O. Regev, On lattices, learning with errors, random linear codes, and cryptography, Journal of the ACM (JACM) 56 (6) (2009) 1–40.

[15] A. Banerjee, C. Peikert, A. Rosen, Pseudorandom functions and lattices, Cryptology ePrint Archive, Paper 2011/401, https://eprint.iacr.org/2011/401 (2011).
URL https://eprint.iacr.org/2011/401

[16] J. Ding, R. Lindner, Identifying ideal lattices, Cryptology ePrint Archive (2007).

[17] M. Blum, S. Micali, How to generate cryptographically strong sequences of pseudo random bits, in: 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), 1982, pp. 112–117. doi:10.1109/SFCS.1982.72.

[18] M. Naor, O. Reingold, On the construction of pseudo-random permutations: Luby-rackoff revisited, in: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, 1997, pp. 189–199.

[19] D. Micciancio, O. Regev, Lattice-based cryptography, in: Post-quantum cryptography, Springer, 2009, pp. 147–191.

[20] Y. Lindell, Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich, Springer, 2017.

[21] R. A. Rueppel, Stream ciphers, in: Analysis and Design of Stream Ciphers, Springer, 1986, pp. 5–16.

[22] O. D. Jensen, K. A. Andersen, A5 encryption in gsm (2017).

[23] E. Biham, A. Shamir, Differential cryptanalysis of des-like cryptosystems, Journal of CRYPTOLOGY 4 (1) (1991) 3–72.

[24] X. Yi, R. Paulet, E. Bertino, Homomorphic encryption, in: Homomorphic encryption and applications, Springer, 2014, pp. 27–46.

[25] D. Boneh, K. Lewi, H. Montgomery, A. Raghunathan, Key homomorphic prfs and their applications, in: Annual Cryptology Conference, Springer, 2013, pp. 410–428.

[26] B. Jun, P. Kocher, The intel random number generator, Cryptography Research Inc. white paper 27 (1999) 1–8.

[27] C. Peikert, A decade of lattice cryptography, Found. Trends Theor. Comput. Sci. 10 (4) (2016) 283–424. doi:10.1561/0400000074.
URL https://doi.org/10.1561/0400000074

[28] A. Hülsing, J. Rijneveld, F. Song, Mitigating multi-target attacks in hash-based signatures, in: Public-Key Cryptography–PKC 2016, Springer, 2016, pp. 387–416. doi:doi.org/10.1007/978-3-662-49384-7_15.

[29] J. Buchmann, K. Lauter, M. Mosca, Postquantum cryptography, part 2, IEEE Security & Privacy 16 (05) (2018) 12–13. doi:https://doi.org/10.1109/MSP.2018.3761714.

[30] D. Micciancio, S. Goldwasser, Closest vector problem, in: Complexity of Lattice Problems, Springer, 2002, pp. 45–68.

[31] O. Regev, The learning with errors problem, Invited survey in CCC 7 (30) (2010) 11.

[32] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, et al., Homomorphic encryption standard, in: Protecting Privacy through Homomorphic Encryption, Springer, 2021, pp. 31–62.

[33] A. Becker, N. Gama, A. Joux, Solving shortest and closest vector problems: The decomposition approach, Cryptology ePrint Archive (2013).

[34] T. Laarhoven, J. van de Pol, B. de Weger, Solving hard lattice problems and the security of lattice-based cryptosystems, Cryptology ePrint Archive (2012).

[35] J. Alwen, S. Krenn, K. Pietrzak, D. Wichs, Learning with rounding, revisited, in: Annual Cryptology Conference, Springer, 2013, pp. 57–74.

[36] C. H. Bennett, G. Brassard, Quantum cryptography: Public key distribution and con tos5 (2014).

[37] S. Hassan, M. U. Bokhari, Design of pseudo random number generator using linear feedback shift register, International Journal of Engineering and Advanced Technology 9 (2) (2019) 1956–1965.

[38] B. Schneier, Applied cryptography: protocols, algorithms, and source code in C, john wiley & sons, 2007.

[39] V. Scarani, H. Bechmann-Pasquinucci, N. J. Cerf, M. Dušek, N. Lütkenhaus, M. Peev, The security of practical quantum key distribution, Rev. Mod. Phys. 81 (2009) 1301–1350. doi:10.1103/RevModPhys.81.1301.
URL https://link.aps.org/doi/10.1103/RevModPhys.81.1301

[40] J. L. G. Pardo, C. Gómez-Rodríguez, The sha-3 family of cryptographic hash functions and extendable-output functions, in: Maple Document, 2015.

[41] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, A statistical test suite for random and pseudorandom number generators for cryptographic applications, Tech. rep., Booz-allen and hamilton inc mclean va (2001).

[42] G. Marsaglia, The marsaglia random number cdrom including the diehard battery of tests of randomness, http://www. stat. fsu. edu/pub/diehard/ (2008).

[43] J. Walker, Ent: a pseudorandom number sequence test program, Software and documentation available at/www. fourmilab. ch/random/S (2008).

[44] J. Soto, Statistical testing of random number generators, in: Proceedings of the 22nd national information systems security conference, Vol. 10, NIST Gaithersburg, MD, 1999, p. 12.

[45] F. A. P. Petitcolas, Kerckhoffs' Principle, Springer US, Boston, MA, 2011, pp. 675–675. doi:10.1007/978-1-4419-5906-5_487.
URL https://doi.org/10.1007/978-1-4419-5906-5_487

[46] V. K. Rai, S. Tripathy, J. Mathew, Memristor based random number generator: Architectures and evaluation, Procedia Computer Science 125 (2018) 576–583. doi:doi:10.1016/J.PROCS.2017.12.074.

[47] M. A. S. AL-khatib, A. H. Lone, Acoustic lightweight pseudo random number generator based on cryptographically secure lfsr, International Journal of Computer Network and Information Security 11 (2) (2018) 38. doi:doi:10.5815/ijcnis.2018.02.05.

[48] B. Lampert, R. S. Wahby, S. Leonard, P. Levis, Robust, low-cost, auditable random number generation for embedded system security, in: Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM, 2016, pp. 16–27.

[49] Timestamp - wikipedia.
URL https://en.wikipedia.org/wiki/Timestamp

[50] Windows api gettickcount64 used to retrieve system uptime on windows with 64-bit resolution (5000+ years) · github.
URL https://gist.github.com/Iristyle/a6cb43d3d6d9f3756b29ef1706120e8b