Quantum Random Number Generators for NIST Post-Quantum Cryptography Standard Algorithms

Abel C. H. Chen

Information & Communications Security Laboratory, Chunghwa Telecom Laboratories, Taiwan
E-mail: chchen.scholar@gmail.com

Received 02 January 2026; Accepted 09 March 2026

Abstract

In recent years, the advancement of quantum computing technology has posed potential security threats to RSA cryptography and elliptic curve cryptography. In response, the National Institute of Standards and Technology (NIST) published several Federal Information Processing Standards (FIPS) of post-quantum cryptography (PQC) in August 2024, including the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM), Module-Lattice-Based Digital Signature Algorithm (ML-DSA), and Stateless Hash-Based Digital Signature Algorithm (SLH-DSA). Although these PQC algorithms are designed to resist quantum computing attacks, they may not provide adequate security in certain specialized application scenarios. To address this issue, this study proposes quantum random number generator (QRNG)-based PQC algorithms. These algorithms leverage quantum computing to generate random numbers, which serve as the foundation for key pair generation, key encapsulation, and digital signature generation. A generalized architecture of QRNG is proposed, along with the design of six QRNGs. Each generator is evaluated according to the statistical validation procedures outlined in

Journal of Web Engineering, Vol. 25_5, 861–888.
doi: 10.13052/jwe1540-9589.2555

NIST SP 800-90B, including tests for verification of entropy sources and independent and identically distributed (IID) outputs. Experimental results assess the computation time of the six QRNGs, as well as the performance of QRNG-based ML-KEM, QRNG-based ML-DSA, and QRNG-based SLH-DSA. These findings provide valuable reference data for future deployment of PQC-based Transport Layer Security in web systems.

Keywords: Quantum random number generators, true random number, ML-KEM, ML-DSA, SLH-DSA.

1 Introduction

In response to the security threats posed by quantum computing to RSA and elliptic curve cryptography [1], the National Institute of Standards and Technology (NIST) initiated a call for post-quantum cryptography (PQC) algorithms in 2016, making significant contributions toward the development of PQC standards. As part of these efforts, NIST formally standardized several Federal Information Processing Standards (FIPS) in August 2024, including the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) [2], the Module-Lattice-Based Digital Signature Algorithm (ML-DSA) [3], and the Stateless Hash-Based Digital Signature Algorithm (SLH-DSA) [4]. Additional standards are expected to be developed for algorithms such as FN-DSA (Fast-Fourier Transform over NTRU-Lattice-Based DSA) [5] and HQC (Hamming Quasi-Cyclic) [6]. Furthermore, in November 2024, NIST published a draft for the transition to PQC standards [7], outlining a clear transition timeline. This initiative highlights the growing importance of integrating PQC into contemporary cybersecurity practices.

In 2024, Yi-Kai Liu and Dustin Moody (major contributors to NIST's PQC project) published a paper in *Physical Review Applied* titled "Post-quantum Cryptography and the Quantum Future of Cybersecurity" [8]. The paper reviews recent developments in both PQC and quantum computing, discussing their respective strengths and limitations. It also emphasizes the potential for complementary integration between these two domains, suggesting that their combination could further enhance overall security. Motivated by this perspective, this study proposes a generalized architecture for quantum random number generator (QRNG), designing six QRNG implementations. Based on these generators, QRNG-based ML-KEM, QRNG-based ML-DSA, and QRNG-based SLH-DSA are developed. These systems

leverage quantum-generated randomness for key pair generation, key encapsulation, and digital signature generation, aiming to explore and enhance the synergy between quantum computing and PQC algorithms.

The contributions of this study are summarized as the follows.

- This study proposes a generalized architecture for QRNG, utilizing six different combinations of quantum logic gates to generate uniform superposition states, which are then measured to produce random bit values.
- The six QRNGs are validated using the testing procedures described in NIST SP 800-90B [9], confirming compliance with entropy validation requirements and the independent and identically distributed (IID) conditions for random bit sequences.
- The proposed QRNGs are applied to a key encapsulation mechanism (KEM), using ML-KEM as a case study for key pair generation and key encapsulation.
- The proposed QRNGs are applied to digital signature algorithm (DSA), using ML-DSA and SLH-DSA as case studies for key pair generation and digital signature generation.

This paper is organized into seven sections. Section 2 presents the architecture of the QRNG proposed in this study, along with six specific QRNG implementations. Section 3 through Section 5 provide detailed designs of the QRNG-Based ML-KEM, QRNG-Based ML-DSA, and QRNG-Based SLH-DSA, respectively. Section 6 evaluates the randomness and computational efficiency of the proposed QRNGs, as well as the performance of the QRNG-Based ML-KEM, QRNG-Based ML-DSA, and QRNG-Based SLH-DSA. Finally, Section 7 summarizes the contributions of this study and discusses its limitations and potential directions for future research.

2 Quantum Random Number Generator

This section provides a detailed description of the QRNGs designed in this study. Subsection 2.1 proposes a generalized architecture for QRNG. Subsections 2.2 through 2.7 present six QRNGs, each constructed using different quantum logic gates to generate uniform superposition states.

2.1 Proposed Generalized Architecture for QRNG

The core principle of the proposed generalized architecture for QRNG is the use of a uniform superposition state generator comprising one or

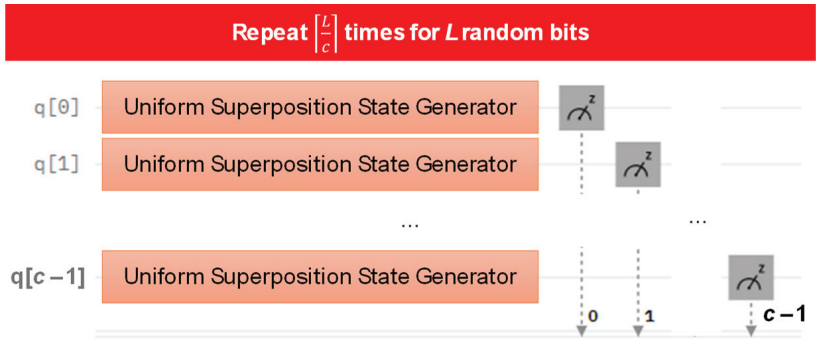

Figure 1 Proposed generalized architecture for QRNG.

more quantum logic gates to prepare each qubit in a uniform superposition state. Measurement is then performed on the superpositioned qubits to yield classical random bit values. Since a qubit in a uniform superposition state has an equal probability (i.e., 0.5) of being measured as either 0 or 1, the resulting bitstream exhibits high entropy.

The proposed QRNG is structured as a quantum circuit composed of c qubits. Each qubit undergoes operations from the uniform superposition state generator, followed by measurement to extract the random bit values. The overall architecture is illustrated in Figure 1. To generate a random bit sequence of length L, the quantum circuit is executed $\lceil \frac{L}{c} \rceil$ times. For instance, to minimize quantum hardware requirements, a circuit with a single qubit (i.e., $c = 1$) may be employed. In such a case, generating a 256-bit random number (i.e., $L = 256$) would require 256 repetitions of the quantum circuit execution.

Subsections 2.2 through 2.7 utilize combinations of the Hadamard Gate (H-Gate), Square-root of X Gate (SX-Gate), Rotation-X Gate (RX-Gate), Rotation-Y Gate (RY-Gate), Phase Gate (P-Gate), and the Universal Single-Qubit Gate (U-Gate) [10] to construct various QRNGs. For the sake of clarity, each design is illustrated using a single-qubit quantum circuit as an example. It is assumed that the initial state of the qubit is $|0\rangle$, which can be represented by the vector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

2.2 Method (1): H-Gate-Based QRNG

This subsection presents the most used method, which generates a uniform superposition state by applying a H-Gate, whose matrix representation is

shown in Equation (1). As shown in Equation (2), the application of the H-Gate results in a uniform superposition state. Consequently, the probability of measuring the qubit in state $|0\rangle$ is $\left|\frac{1}{\sqrt{2}}\right|^2 = 0.5$, and the probability of measuring it in state $|1\rangle$ is also $|\frac{1}{\sqrt{2}}|^2 = 0.5$.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{1}$$

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{2}$$

2.3 Method (2): SX-Gate-Based QRNG

This subsection describes the generation of a uniform superposition state by applying the SX-Gate, whose matrix representation is provided in Equation (3). As shown in Equation (4), the application of the SX-Gate results in a uniform superposition state. Consequently, the probability of measuring the qubit in state $|0\rangle$ is $|\frac{1+i}{2}|^2 = 0.5$, and the probability of measuring it in state $|1\rangle$ is $|\frac{1-i}{2}|^2 = 0.5$.

$$SX = \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}. \tag{3}$$

$$SX|0\rangle = \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1+i \\ 1-i \end{bmatrix}. \tag{4}$$

2.4 Method (3): RX-Gate-Based QRNG

This subsection explains the generation of a uniform superposition state by applying the RX-Gate, whose matrix representation is given in Equation (5), with a rotation angle of $\frac{\pi}{2}$ (i.e., $\theta = \frac{\pi}{2}$) around the X-axis. As shown in Equation (6), this operation results in a uniform superposition state. Consequently, the probability of measuring the qubit in state $|0\rangle$ is $|\frac{1}{\sqrt{2}}|^2 = 0.5$, and the probability of measuring it in state $|1\rangle$ is $|-\frac{i}{\sqrt{2}}|^2 = 0.5$.

$$RX(\theta) = \begin{bmatrix} cos\left(\frac{\theta}{2}\right) & -i\ sin\left(\frac{\theta}{2}\right) \\ -i\ sin\left(\frac{\theta}{2}\right) & cos\left(\frac{\theta}{2}\right) \end{bmatrix}. \tag{5}$$

$$RX\left(\frac{\pi}{2}\right)|0\rangle = \begin{bmatrix} cos\left(\frac{\pi}{4}\right) & -i\ sin\left(\frac{\pi}{4}\right) \\ -i\ sin\left(\frac{\pi}{4}\right) & cos\left(\frac{\pi}{4}\right) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} cos\left(\frac{\pi}{4}\right) \\ -i\ sin\left(\frac{\pi}{4}\right) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{i}{\sqrt{2}} \end{bmatrix}. \tag{6}$$

2.5 Method (4): RY-Gate-Based QRNG

This subsection describes the generation of a uniform superposition state by applying the RY-Gate, whose matrix representation is shown in Equation (7), with a rotation angle of $\frac{\pi}{2}$ (i.e., $\theta = \frac{\pi}{2}$) about the Y-axis. As demonstrated in Equation (8), this operation produces a uniform superposition state. As a result, the probability of measuring the qubit in state $|0\rangle$ is $|\frac{1}{\sqrt{2}}|^2 = 0.5$, and the probability of measuring it in state $|1\rangle$ is also $|\frac{1}{\sqrt{2}}|^2 = 0.5$.

$$RY(\theta) = \begin{bmatrix} cos\left(\frac{\theta}{2}\right) & -sin\left(\frac{\theta}{2}\right) \\ sin\left(\frac{\theta}{2}\right) & cos\left(\frac{\theta}{2}\right) \end{bmatrix}. \tag{7}$$

$$RY\left(\frac{\pi}{2}\right)|0\rangle = \begin{bmatrix} cos\left(\frac{\pi}{4}\right) & -sin\left(\frac{\pi}{4}\right) \\ sin\left(\frac{\pi}{4}\right) & cos\left(\frac{\pi}{4}\right) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} cos\left(\frac{\pi}{4}\right) \\ sin\left(\frac{\pi}{4}\right) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}. \tag{8}$$

2.6 Method (5): (P-Gate & H-Gate)-Based QRNG

This subsection presents the generation of a uniform superposition state by applying a P-Gate with a rotation angle of $\frac{\pi}{2}$ (i.e., $\theta = \frac{\pi}{2}$), followed by a H-Gate. The matrix representation of the P-Gate is shown in Equation (9), and the resulting quantum state is illustrated in Equation (10). As a result,

the probability of measuring the qubit in state $|0\rangle$ is $|\frac{1}{\sqrt{2}}|^2 = 0.5$, and the probability of measuring it in state $|1\rangle$ is also $|\frac{1}{\sqrt{2}}|^2 = 0.5$.

$$P(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}. \tag{9}$$

$$HP(\theta)|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{10}$$

2.7 Method (6): U-Gate-Based QRNG

This subsection describes the generation of a uniform superposition state by applying a U-Gate, whose matrix representation is provided in Equation (11). The operation involves sequential rotations: a rotation of $\frac{\pi}{2}$ about the Z-axis, followed by a rotation of $\frac{\pi}{2}$ about the Y-axis, and another rotation of $\frac{\pi}{2}$ about the Z-axis. The resulting quantum state is shown in Equation (12). Consequently, the probability of measuring the qubit in state $|0\rangle$ is $|\frac{1}{\sqrt{2}}|^2 = 0.5$, and the probability of measuring it in state $|1\rangle$ is $|\frac{\mathrm{i}}{\sqrt{2}}|^2 = 0.5$.

$$U\left(\frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}\right) = \begin{bmatrix} cos\left(\frac{\pi}{4}\right) & -e^{i\frac{\pi}{2}} sin\left(\frac{\pi}{4}\right) \\ e^{i\frac{\pi}{2}} sin\left(\frac{\pi}{4}\right) & e^{i\pi} cos\left(\frac{\pi}{4}\right) \end{bmatrix}. \tag{11}$$

$$\begin{aligned} U\left(\frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}\right)|0\rangle &= \begin{bmatrix} cos\left(\frac{\pi}{4}\right) & -e^{i\frac{\pi}{2}} sin\left(\frac{\pi}{4}\right) \\ e^{i\frac{\pi}{2}} sin\left(\frac{\pi}{4}\right) & e^{i\pi} cos\left(\frac{\pi}{4}\right) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} cos\left(\frac{\pi}{4}\right) \\ e^{i\frac{\pi}{2}} sin\left(\frac{\pi}{4}\right) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{bmatrix}. \end{aligned} \tag{12}$$

3 QRNG-based ML-KEM

FIPS 203 specifies the parameters and detailed algorithms for the ML-KEM. This section focuses on identifying components within the design that can be modified to incorporate QRNGs in order to enhance security.

3.1 Key Pair Generation

The key pair generation procedure for ML-KEM is primarily defined in Algorithm 19 of FIPS 203 [2]. In this algorithm, two 32-byte random values, denoted as d and z, are generated and used as inputs to the function ML-KEM.KeyGen_internal(d, z), which outputs an encapsulation key ek (i.e., a public key) and a decapsulation key dk (i.e., a private key) [2]. In this study, the algorithm is modified to replace the random number generation with quantum random number generation. Specifically, two 256-bit random values generated by a QRNG are used in place of d and z. The revised version of the algorithm is presented as Algorithm 1 in this study.

Algorithm 1 ML-KEM.KeyGen()

Output: encapsulation key ek.
Output: decapsulation key dk.
1: $d \leftarrow$ **256 random bits from QRNG**
2: z $\leftarrow$ **256 random bits from QRNG**
3: **if** d == NULL **or** z == NULL **then**
4: **return** $\perp$
5: **end if**
6: (ek, dk) $\leftarrow$ ML-KEM.KeyGen_internal(d, z)
7: **return** (ek, dk)

3.2 Key Encapsulation

To achieve security under the Indistinguishability under Chosen-Ciphertext Attack (IND-CCA) model, a different random value is used each time a ciphertext is generated. The key encapsulation procedure in the ML-KEM is primarily defined in Algorithm 20 of FIPS 203 [2]. In this procedure, an encapsulation key ek and a random value m are provided as inputs to ML-KEM.Encaps_internal(ek, m), which produces a shared secret key K and a pseudorandom value r. The ciphertext c is then derived using K-PKE.Encrypt(ek, m, r) [2]. In this study, the algorithm is modified by replacing the generated random value m with a 256-bit random value produced by a QRNG. The revised version of the algorithm is presented as Algorithm 2 in this study.

Algorithm 2 ML-KEM.Encaps(*ek*)

Input: encapsulation key *ek*.
Output: shared secret key *K*.
Output: ciphertext *c*.

1: $m \leftarrow$ **256 random bits from QRNG**
2: **if** m == NULL **then**
3: **return** $\perp$
4: **end if**
5: (K, c) ML-KEM.Encaps_internal(ek, m)
6: **return** (K, c)

3.3 Generating Multiple Random Values By a QRNG

When multiple random values are required, the QRNG can be invoked either multiple times or once, depending on the implementation strategy. For example, in the key pair generation procedure of the ML-KEM, two random values of 256 bits each are needed. This can be achieved by invoking the QRNG twice to generate two separate 256-bit random values. Alternatively, the QRNG can be invoked once to produce a single 512-bit random value, from which the first 256 bits are used as d and the remaining 256 bits as z. If the random bits generated by the QRNG satisfy the condition of being IID, both approaches (i.e., multiple invocations and a single invocation) are considered equivalent in terms of randomness quality.

4 QRNG-based ML-DSA

FIPS 204 defines the relevant parameters and specific algorithms for the ML-DSA. This section focuses on identifying components within the design that can be modified to incorporate QRNGs in order to enhance security.

4.1 Key Pair Generation

The key pair generation algorithm of the ML-DSA is primarily defined in Algorithm 1 of FIPS 204 [3]. In this algorithm, a 32-byte random value ξ is generated and used as input to ML-DSA.KeyGen_internal(ξ) to derive the public key *pk* and the private key *sk* [3]. In this study, the algorithm is modified such that a 256-bit random value is generated by a QRNG and used as ξ. The modified version of the algorithm is presented as Algorithm 3 in this study.

Algorithm 3 ML-DSA.KeyGen()
Output: public key *pk*.
Output: private key *sk*.
1: $\xi \leftarrow$ **256 random bits from QRNG**
2: **if** ξ == NULL **then**
3: **return** $\perp$
4: **end if**
5: **return** ML-DSA.KeyGen_internal(ξ)

4.2 Signature Generation Based on ML-DSA

To achieve the security level of Existential Unforgeability under Chosen Message Attack (EUF-CMA), each signature generation must incorporate a unique random value. The signature generation procedure for the ML-DSA is defined in Algorithm 2 of FIPS 204 [3]. In this process, the private key *sk*, the message to be signed *M'*, and a 32-byte random value *rnd* are used as inputs to ML-DSA.Sign_internal(*sk*, *M'*, *rnd*) to produce the signature ξ [3]. In this study, the algorithm is modified such that a 256-bit random value is generated by a QRNG and used as *rnd*. The modified version of the algorithm is presented as Algorithm 4 in this study.

Algorithm 4 ML-DSA.Sign(*sk*, *M*, *ctx*)
Input: private key *sk*, message *M*, context string *ctx*.
Output: signature ξ.
1: **if** |*ctx*|> 255 **then**
2: **return** $\perp$
3: **end if**
4:
5: *rnd* $\leftarrow$ **256 random bits from QRNG**
6: **if** *rnd* == NULL **then**
7: **return** $\perp$
8: **end if**
9:
10: *M'* $\leftarrow$ BytesToBits(IntegerToBytes(0, 1) $\|$
IntegerToBytes(|*ctx*|, 1) $\|$ *ctx*) $\|$ *M*
11: $\xi \leftarrow$ ML-DSA.Sign_internal(*sk*, *M'*, *rnd*)
12: **return** ξ

4.3 Signature Generation Based on Pre-Hash ML-DSA

Algorithm 4 of FIPS 204 defines the Pre-Hash ML-DSA, in which a specified pre-hash function *PH* is applied to the message *M* to compute PH_M. The object identifier (OID) and PH_M are then used to derive the message M' to be signed. All subsequent computations follow the same procedure as described in Algorithm 2 of FIPS 204 [3]. In this study, the algorithm is modified by employing a QRNG to produce a 256-bit random value used as *rnd*. The modified version of the algorithm is presented as Algorithm 5 in this study.

Algorithm 5 HashML-DSA.Sign(*sk*, *M*, *ctx*, *PH*)

Input: private key *sk*, message *M*, context string *ctx*,
pre-hash function *PH*.
Output: signature ξ.

1: **if** |*ctx*|> 255 **then**
2: **return** $\perp$
3: **end if**
4:
5: *rnd* $\leftarrow$ **256 random bits from QRNG**
6: **if** *rnd* == NULL **then**
7: **return** $\perp$
8: **end if**
9: execute the flows from **Line 9** to **Line 22** in **Algorithm 4** of FIPS 204 to get OID and generate the hashed message PH_M based on *PH*.
...
23: *M'* $\leftarrow$ BytesToBits(IntegerToBytes(1, 1) $\|$ IntegerToBytes(|*ctx*|, 1) $\|$ *ctx* $\|$ OID $\|$ PH_M)
24: $\xi \leftarrow$ ML-DSA.Sign_internal(*sk*, M', *rnd*)
25: **return** ξ

5 QRNG-based SLH-DSA

FIPS 205 defines the parameters and detailed algorithms of the SLH-DSA. This section focuses on identifying the components within the design that can be modified to incorporate QRNGs in order to enhance security.

5.1 Key Pair Generation

The key pair generation algorithm for the SLH-DSA is primarily defined in Algorithm 21 of FIPS 205 [4]. This process involves generating three random values of n bytes each, namely SK.seed, SK.prf, and PK.seed. These values are then input into the function slh_keygen_internal(SK.seed, SK.prf, PK.seed) to compute the public key PK and the private key SK [4]. In this study, the algorithm is modified such that three $8n$-bit random numbers are generated using a QRNG to serve as SK.seed, SK.prf, and PK.seed, respectively. The modified algorithm is presented as Algorithm 6 in this study.

Algorithm 6 slh_keygen()

Output: SLH-DSA key pair (SK, PK).

1: SK.seed ← **$8n$ random bits from QRNG**
2: SK.prf ← **$8n$ random bits from QRNG**
3: PK.seed ← **$8n$ random bits from QRNG**
4: **if** SK.seed == NULL **or** SK.prf == NULL **or** PK.seed == NULL **then**
5: **return** ⊥
6: **end if**
7: **return** slh_keygen_internal(SK.seed, SK.prf, PK.seed)

The SLH-DSA supports various parameter sets, including SLH-DSA-SHA2-128s and SLH-DSA-SHAKE-128f, each associated with a different value of n, as shown in Table 1 [4]. For instance, when using SLH-DSA-SHAKE-128f, the corresponding n value is 16, indicating that three 128-bit random numbers are required during key pair generation.

Table 1 The value of n for SLH-DSA

	n	$8n$	Security Level
SLH-DSA-SHA2-128s			
SLH-DSA-SHAKE-128s	16	128	1
SLH-DSA-SHA2-128f			
SLH-DSA-SHAKE-128f	16	128	1
SLH-DSA-SHA2-192s			
SLH-DSA-SHAKE-192s	24	192	3
SLH-DSA-SHA2-192f			
SLH-DSA-SHAKE-192f	24	192	3
SLH-DSA-SHA2-256s			
SLH-DSA-SHAKE-256s	32	256	5
SLH-DSA-SHA2-256f			
SLH-DSA-SHAKE-256f	32	256	5

As discussed in Subsection 3.3, when the random bits generated by the QRNG are IID, invoking the generator once or multiple times is functionally equivalent. Therefore, in the key pair generation process for SLH-DSA-SHAKE-128f, the QRNG can be invoked once to produce 384 bits of random data. The first 128 bits are used as *SK*.seed, the next 128 bits as *SK*.prf, and the final 128 bits as *PK*.seed.

5.2 Signature Generation Based on SLH-DSA

To achieve EUF-CMA, a unique random value is required for each signature generation. The signature generation process of SLH-DSA is primarily defined in FIPS 205, Algorithm 22 [4]. In this process, the message to be signed M', the private key *SK*, and a random value *addrnd* of *n* bytes are input into the function slh_sign_internal(*M'*, SK, *addrnd*) to produce the signature *SIG* [4]. Accordingly, this study modifies the algorithm by generating an $8n$-bit random value for *addrnd* using a QRNG. The modified procedure is detailed in Algorithm 7 of this study. For instance, when generating a signature using SLH-DSA-SHAKE-128f, a 128-bit random value is produced by the QRNG for use as *addrnd*.

Algorithm 7 slh_sign(*M*, *ctx*, *SK*)

Input: message *M*, context string *ctx*, private key *SK*.
Output: SLH-DSA signature *SIG*.

1: **if** |*ctx*|> 255 **then**
2: **return** $\perp$
3: **end if**
4: *addrnd* $\leftarrow$ **8*n* random bits from QRNG**
5: **if** *addrnd* == NULL **then**
6: **return** $\perp$
7: **end if**
8: $M' \leftarrow$ toByte(0, 1) $\|$ toByte(|*ctx*|, 1) $\|$ *ctx* $\|$ *M*
9: *SIG* $\leftarrow$ slh_sign_internal(*M'*, *SK*, *addrnd*)
10: **return** *SIG*

5.3 Signature Generation Based on Pre-Hash SLH-DSA

FIPS 205, Algorithm 23 also defines the Pre-Hash SLH-DSA. In this algorithm, a specified pre-hash function *PH* is applied to the message *M* to generate the pre-hashed message PH_M. Based on the OID and PH_M, the message to be signed M' is derived. The remaining steps of the algorithm are consistent with those described in FIPS 205, Algorithm 22 [4]. Accordingly,

this study modifies the algorithm by generating an $8n$-bit random value using a QRNG for use as *addrnd*. The modified procedure is presented as Algorithm 8 in this study. For example, when generating a signature using SLH-DSA-SHAKE-128f, a 128-bit random value is generated by the QRNG and used as *addrnd*.

6 Experimental Results and Discussion

This section provides a performance evaluation of the six QRNGs developed in this study, along with the QRNG-Based ML-KEM, QRNG-Based ML-DSA, and QRNG-Based SLH-DSA implementations. Subsection 6.1 outlines the experimental environment and configuration settings. Subsections 6.2 and 6.3 assess the randomness of the output bits generated by the QRNGs, while Subsection 6.4 analyzes the computational overhead associated with the six QRNG designs. Lastly, Subsection 6.5 examines the computation time required when integrating the QRNGs into the NIST post-quantum cryptographic standard algorithms.

Algorithm 8 hash_slh_sign(*M*, *ctx*, *PH*, *SK*)

Input: message *M*, context string *ctx*, pre-hash function *PH*, private key *SK*.
Output: SLH-DSA signature *SIG*.

1: **if** |*ctx*|> 255 **then**
2: **return** $\perp$
3: **end if**
4: *addrnd* $\leftarrow$ **8*n* random bits from QRNG**
5: **if** *addrnd* == NULL **then**
6: **return** $\perp$
7: **end if**
8: execute the flows from **Line 8** to **Line 23** in **Algorithm 23** of FIPS 205 to get OID and generate the hashed message PH_M based on *PH*.
...
24: M' $\leftarrow$ toByte(1, 1) $\|$ toByte(|*ctx*|, 1) $\|$ *ctx*) $\|$ OID $\|$ PH_M
25: *SIG* $\leftarrow$ slh_sign_internal(M', *SK*, *addrnd*)
26: **return** *SIG*

6.1 Experimental Environment

In late 2024, the Groupe Speciale Mobile Association (GSMA) published the white paper "IG.18 Opportunities and Challenges for Hybrid (QKD and PQC) Scenarios," which highlighted the potential integration of QRNGs with PQC algorithms in both core network and end entities [11]. In response to this, this study focuses on the end entity scenario, implementing QRNGs and PQC algorithms on Raspberry Pi 4 hardware.

Given that this research emphasizes proof-of-concept (PoC) validation, it does not employ physical quantum chips. Instead, quantum computation is simulated using IBM Qiskit SDK version 1.1.1. It is important to note that real quantum hardware will be necessary in the future to achieve true quantum physical properties and security assurances.

For the implementation of PQC algorithms, the BouncyCastle Open Source API version 1.81 is used, with modifications to its internal functions to enable the use of QRNG-generated random bits. It should also be emphasized that future implementations must incorporate secure communication channels and mechanisms to ensure the integrity and confidentiality of such operations.

6.2 Entropy Validation of Random Bits from QRNGs

This study adopts the validation methodology outlined in NIST SP 800-90B [9] and constructs two datasets for analysis.

(1) Sequential Dataset: One million random bits are generated to analyze the distribution of 0s and 1s.
(2) Restart Test Dataset: The system is restarted 1,000 times, and in each instance, 1,000 random bits are generated, forming a 1000×1000 restart matrix.

For the restart matrix, the number of occurrences of 0s and 1s is calculated for each row and each column. The maximum frequency among these values is defined as the Most Common Value (MCV). The MCV is then used to compute the minimum entropy and evaluate the randomness using a binomial distribution test to derive the corresponding p-value.

The validation results for the six QRNG designs developed in this study are presented in Table 2 and Figure 3. Since each row and column in the restart matrix consists of 1,000 bits, the expected frequency of 0 s and 1 s under a uniform distribution is 500. Thus, the closer the MCV is to 500, the higher the resulting minimum entropy and p-value. If the p-value falls below the threshold defined by NIST SP 800-90B (i.e., 0.000005), it indicates insufficient randomness and a lack of adequate entropy.

Table 2 Entropy validation of random bits from QRNGs

QRNG	MCV	mini_entropy	p-value
H-Gate-Based	555	0.99125	0.00056
SX-Gate-Based	552	0.99218	0.00112
RX-Gate-Based	559	0.98993	0.00021
RY-Gate-Based	560	0.98959	0.00017
Based on P-Gate & H-Gate	551	0.99248	0.00139
U-Gate-Based	558	0.99027	0.00027

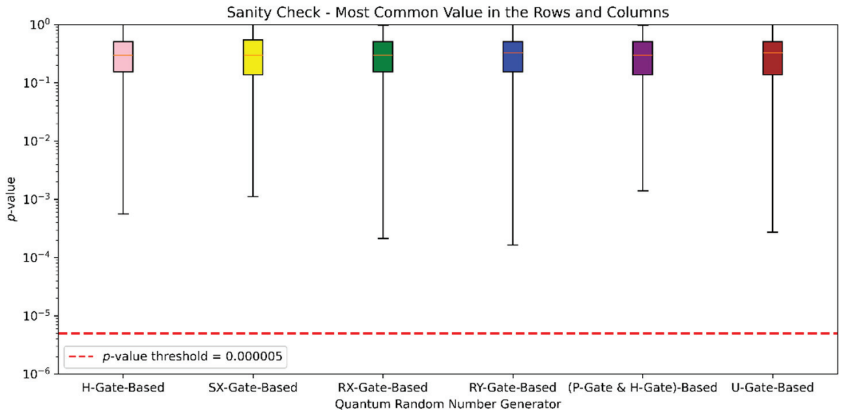

Figure 2 Sanity check results.

The use of the MCV serves to identify the dataset with the lowest entropy (i.e., the worst-case scenario) among the 2,000 evaluated groups. If the worst-case data sample passes the validation, it implies that the remaining 1,999 data samples would also pass.

Figure 2 presents the distribution of p-values for the six QRNGs using box plots across 2,000 data samples per QRNG. All p-values exceed the 0.000005 threshold, indicating that each of the six QRNGs produces sufficiently random and unpredictable bitstreams, thereby passing the NIST SP 800-90B randomness validation.

6.3 IID Validation of Random Bits from QRNGs

In addition to verifying that the QRNGs produce sufficiently random bitstreams, it is also essential to ensure that the output bits are IID. To this end, NIST SP 800-90B [9] defines three statistical tests for IID verification: the Independence Test (Ind. Test), the Goodness-of-Fit Test (GF Test), and the

Length of the Longest Repeated Substring Test (Length of the LRS Test). These tests are described as follows.

(1) Independence Test (Ind. Test): As described in Subsection 6.2, each row and column of the restart matrix consists of 1,000 random bits. Each sample is divided into 10 equal segments, with each segment containing 100 bits. Under the assumption of a uniform distribution, the expected frequency of both 0s and 1s is 50 per segment. A chi-squared test is used to assess the deviation between observed and expected frequencies, producing a chi-squared statistic for each dataset. With 9 degrees of freedom, the corresponding p-value is computed. According to NIST SP 800-90B [9], the threshold p-value is 0.001, corresponding to a chi-squared critical value of 27.877.

(2) Goodness-of-Fit Test (GF Test): Each sample in the restart matrix is segmented into 250 groups of 4 bits each, resulting in one of 16 possible 4-bit combinations (from 0000 to 1111). Under uniform distribution, each combination is expected to occur 15.625 times per dataset. The chi-squared test is applied to measure the deviation between observed and expected frequencies, and the resulting chi-squared statistic is calculated with 14 degrees of freedom. The corresponding p-value is evaluated, with a threshold of 0.001, equivalent to a critical chi-squared value of 36.123, as defined in NIST SP 800-90B [9].

(3) Length of the Longest Repeated Substring Test (Length of the LRS Test): Each sample, consisting of 1,000 bits, is analyzed to determine the length of its longest repeated substring. A binomial distribution test is then applied to compute the p-value. The threshold p-value, as specified in NIST SP 800-90B [9], is 0.001. For sequences of length 1,000, this corresponds to an approximate maximum repeated substring length threshold of 28.

The IID verification results for the six QRNGs developed in this study are presented in Tables 3 and 4. Table 3 provides the median p-values across

Table 3 The median p-values for iid validation

QRNG	Ind. Test	GF Test	Length of the LRS Test
H-Gate-Based	0.41184	0.42615	1.00000
SX-Gate-Based	0.42264	0.42615	1.00000
RX-Gate-Based	0.38726	0.41690	1.00000
RY-Gate-Based	0.39072	0.47381	1.00000
Based on P-Gate & H-Gate	0.41542	0.40774	1.00000
U-Gate-Based	0.40296	0.42615	1.00000

Table 4 The minimum p-values for IID validation

QRNG	Ind. Test	GF Test	Length of the LRS Test
H-Gate-Based	0.00111	0.00116	0.36772
SX-Gate-Based	0.00106	0.00138	0.10787
RX-Gate-Based	0.00126	0.00111	0.60097
RY-Gate-Based	0.00181	0.00106	0.36772
Based on P-Gate & H-Gate	0.00142	0.00157	0.05536
U-Gate-Based	0.00111	0.00187	0.05536

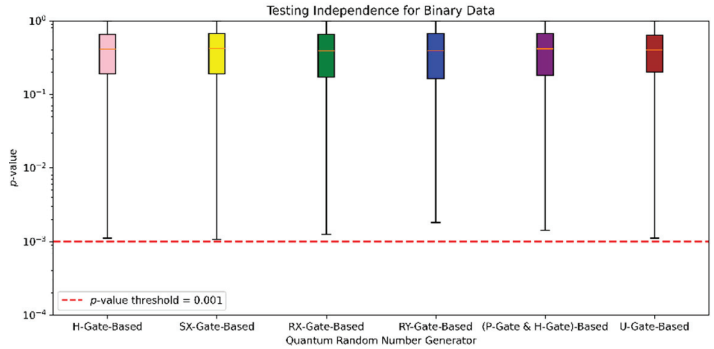

Figure 3 Independence test results.

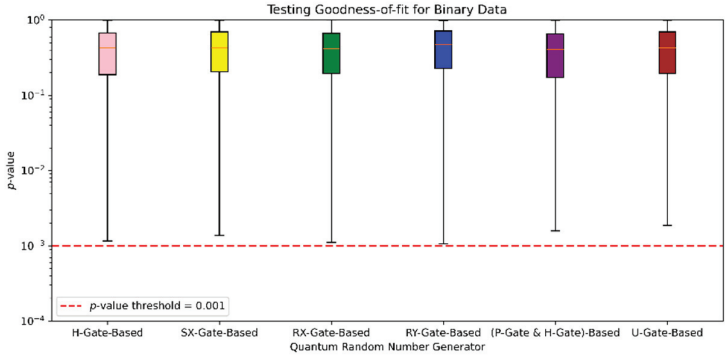

Figure 4 Goodness-of-fit test results.

all datasets for each of the three tests, while Table 4 reports the minimum p-values observed (i.e., the worst-case scenarios). Box plots in Figures 3 and 4 illustrate the distribution of p-values for the Ind. Test and GF Test, respectively. Figure 5 presents a box plot of the longest repeated substring lengths observed in the Length of the LRS Test.

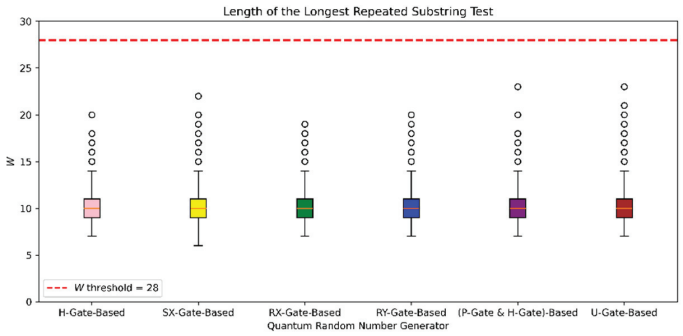

Figure 5 Length of the longest repeated substring test results.

The results show that, even in the worst-case scenarios, all p-values exceed the 0.001 threshold, indicating that all six QRNGs pass the tests for independence and identical distribution as defined by NIST SP 800-90B.

6.4 Computation Time of QRNGs

This section evaluates the computational performance of the six QRNGs designed in this study under different combinations of random bit lengths and qubit counts, denoted as (the length of random bits L, the number of qubits c). Given that the cryptographic algorithms ML-KEM, ML-DSA, and SLH-DSA typically require random numbers of 256 bits, the experiments primarily focus on scenarios with a random bit length of 256.

The number of qubits is varied across four settings: 1, 2, 4, and 8. A smaller number of qubits reduces the cost of constructing quantum circuits but requires a higher number of repeated executions, leading to increased computation time. Conversely, a larger number of qubits increases circuit complexity but reduces the number of repetitions required, thus decreasing computation time.

Each configuration was executed 1,000 times. The median computation times for each configuration are summarized in Table 5. To comprehensively illustrate the experimental results, box plots are presented in Figure 6. It can be observed that the (256, 1) configuration incurs the longest computation time, while the (256, 8) configuration achieves the shortest. These results indicate that further increasing the number of qubits could improve computational efficiency.

Among the QRNG designs, the H-Gate-Based QRNG and SX-Gate-Based QRNG exhibit the highest efficiency due to their relatively simple

Table 5 Computation time of QRNGs (unit: millisecond)

QRNG	(256, 1)	(256, 2)	(256, 4)	(256, 8)
H-Gate-Based	403.80	228.83	146.50	87.00
SX-Gate-Based	408.38	232.95	139.99	98.34
RX-Gate-Based	457.67	255.79	152.32	98.45
RY-Gate-Based	459.49	255.43	151.87	99.84
Based on P-Gate & H-Gate	480.30	274.20	169.87	117.35
U-Gate-Based	520.50	310.82	203.37	148.19

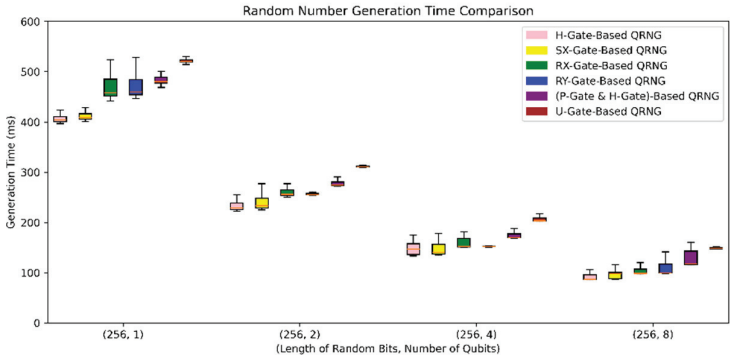

Figure 6 Random number generation time comparison.

computation. In contrast, the (P-Gate & H-Gate)-Based QRNG and U-Gate-Based QRNG involve more complex operations, resulting in longer computation times.

6.5 Performance Evaluation of QRNG-Based PQC Algorithms

This section focuses on evaluating the computation time of QRNGs when applied to NIST PQC standard algorithms. The performance of QRNG-Based ML-KEM, QRNG-Based ML-DSA, and QRNG-Based SLH-DSA is sequentially assessed. Additionally, Java's built-in SecureRandom class is employed as a pseudorandom number generator (PRNG) benchmark for comparison purposes. Each experiment is conducted 1,000 times, and the median results are used for comparative analysis.

6.5.1 Evaluation of QRNG-Based ML-KEM

This study evaluates the computation time for key pair generation using the QRNG-Based ML-KEM, with the results presented in Table 6. It is observed that, even under the ML-KEM-1024 parameter set, the benchmark completes computation within 1 millisecond, demonstrating the high efficiency of the

Table 6 Key pair generation time of QRNG-based ML-KEM (unit: millisecond)

RNG	ML-KEM-512	ML-KEM-768	ML-KEM-1024
SecureRandom PRNG (benchmark)	0.614	0.872	0.979
H-Gate-Based QRNG	173.967	174.518	174.427
SX-Gate-Based QRNG	196.834	196.966	196.854
RX-Gate-Based QRNG	197.693	196.996	197.347
RY-Gate-Based QRNG	199.306	200.978	200.046
QRNG Based on P-Gate & H-Gate	234.547	235.056	248.954
U-Gate-Based QRNG	296.398	296.698	297.025

Table 7 Key encapsulation time of QRNG-based ML-KEM (unit: millisecond)

RNG	ML-KEM-512	ML-KEM-768	ML-KEM-1024
SecureRandom PRNG (benchmark)	0.413	0.595	0.697
H-Gate-Based QRNG	87.131	87.425	87.390
SX-Gate-Based QRNG	98.562	98.648	98.624
RX-Gate-Based QRNG	98.983	98.734	98.853
RY-Gate-Based QRNG	99.876	100.599	100.219
QRNG Based on P-Gate & H-Gate	117.423	117.668	124.542
U-Gate-Based QRNG	148.309	148.506	148.703

ML-KEM algorithms selected and designed by NIST. However, since the QRNGs in this study are primarily implemented on simulators, a longer time is required to generate random numbers. Additionally, the ML-KEM key pair generation requires a total of 512 bits of random numbers, which further increases the computation time. Therefore, the computation time for QRNG-Based ML-KEM key pair generation mainly depends on the time needed for the QRNGs designed in this study to produce 512 bits of random data.

This study also evaluates the computation time for key encapsulation using the QRNG-Based ML-KEM, with the results shown in Table 7. It is observed that the benchmark computation time ranges approximately from 0.413 milliseconds to 0.697 milliseconds. However, since the QRNGs in this study require more time to produce random numbers, and the ML-KEM key encapsulation requires a total of 256 bits of random data, the computation time for QRNG-Based ML-KEM key encapsulation mainly depends on the time needed for the quantum random number generators designed in this study to generate 256 bits of random numbers.

6.5.2 Evaluation of QRNG-Based ML-DSA

This study evaluates the computation time for key pair generation using the QRNG-Based ML-DSA, with the experimental results presented in Table 8.

Table 8 Key pair generation time of QRNG-based ML-DSA (unit: millisecond)

RNG	ML-DSA-44	ML-DSA-65	ML-DSA-87
SecureRandom PRNG (benchmark)	0.905	1.070	1.355
H-Gate-Based QRNG	87.702	88.575	88.375
SX-Gate-Based QRNG	98.806	99.545	99.369
RX-Gate-Based QRNG	99.332	99.540	99.465
RY-Gate-Based QRNG	100.369	100.619	101.089
QRNG Based on P-Gate & H-Gate	117.522	118.806	117.829
U-Gate-Based QRNG	149.015	149.176	149.625

Table 9 Signature generation time of QRNG-based ML-DSA (unit: millisecond)

RNG	ML-DSA-44	ML-DSA-65	ML-DSA-87
SecureRandom PRNG (benchmark)	1.390	2.371	2.794
H-Gate-Based QRNG	88.665	91.308	90.826
SX-Gate-Based QRNG	99.377	101.092	100.545
RX-Gate-Based QRNG	100.276	101.798	101.653
RY-Gate-Based QRNG	101.554	104.424	104.534
QRNG Based on P-Gate & H-Gate	118.910	123.106	120.780
U-Gate-Based QRNG	149.915	150.914	151.777

It can be observed that, even under the ML-DSA-87 parameter set, the benchmark computation time is approximately 1.355 milliseconds. The ML-DSA, selected and designed by NIST, demonstrates high efficiency. However, since the QRNGs in this study are primarily implemented on a simulator, a longer time is required to generate random numbers. Although ML-DSA key pair generation only requires 256 bits of random data, the time consumed exceeds that of the benchmark. Therefore, the computation time for QRNG-Based ML-DSA key pair generation, combining the six QRNGs designed in this study, is mainly determined by the time required for generating 256 bits of random numbers by the QRNGs.

This study also evaluates the computation time for signature generation using the QRNG-Based ML-DSA, with the experimental results shown in Table 9. It can be observed that the benchmark computation time ranges from approximately 1.390 milliseconds to 2.794 milliseconds. Since ML-DSA signature generation requires a total of 256 bits of random data, the computation time for the QRNG-Based ML-DSA signature generation, as designed in this study, is similar to that of key pair generation. This time is primarily dependent on the duration required by the QRNGs to produce 256 bits of random data.

6.5.3 Evaluation of QRNG-based SLH-DSA

Due to the large number of parameter combinations in SLH-DSA, this study primarily focuses on the SLH-DSA SHAKE-f series for comparison. The computation time for key pair generation using the QRNG-Based SLH-DSA was evaluated, with results presented in Table 10. The benchmark computation time was observed to range from approximately 6.445 milliseconds to 23.560 milliseconds. The computation time for QRNG-Based SLH-DSA key pair generation, as designed in this study using six different QRNGs, mainly depends on the time required to generate the random numbers. Specifically, generating the SLH-DSA SHAKE-128f key pair requires 384 bits of random data, resulting in shorter computation times. In contrast, the SLH-DSA SHAKE-192f and SHAKE-256f key pairs require 576 bits and 768 bits of random data, respectively, which leads to longer computation times.

This study also evaluated the computation time for signature generation using QRNG-Based SLH-DSA, with the experimental results summarized in Table 11. Due to the relatively long computation time required for SLH-DSA signature generation, the benchmark computation time ranged approximately from 148.651 milliseconds to 490.060 milliseconds. However, since the computation time for random number generation using the six QRNGs designed in this study remains higher than that of the SecureRandom PRNG, the overall computation time for QRNG-Based SLH-DSA signature generation is longer. Specifically, generating the SLH-DSA SHAKE-128f signature requires 128 bits of random data, resulting in shorter computation times. In contrast, generating SLH-DSA SHAKE-192f and SHAKE-256f signatures requires 192 bits and 256 bits of random data, respectively, which leads to longer computation times.

Table 10 Key pair generation time of QRNG-based SLH-DSA (unit: millisecond)

RNG	SLH-DSASHAKE-128f	SLH-DSASHAKE-192f	SLH-DSASHAKE-256f
SecureRandom PRNG (benchmark)	6.445	9.079	23.560
H-Gate-Based QRNG	136.645	204.519	283.712
SX-Gate-Based QRNG	154.540	229.921	316.971
RX-Gate-Based QRNG	153.877	229.666	318.319
RY-Gate-Based QRNG	162.391	233.187	322.601
QRNG Based on P-Gate & H-Gate	182.025	275.916	375.705
U-Gate-Based QRNG	228.539	342.069	467.849

Table 11 Signature generation time of QRNG-based SLH-DSA (unit: millisecond)

RNG	SLH-DSASHAKE-128f	SLH-DSASHAKE-192f	SLH-DSASHAKE-256f
SecureRandom PRNG (benchmark)	148.651	234.374	490.060
H-Gate-Based QRNG	192.138	299.832	577.291
SX-Gate-Based QRNG	198.065	308.165	587.393
RX-Gate-Based QRNG	198.093	308.367	588.517
RY-Gate-Based QRNG	200.975	309.816	591.005
QRNG Based on P-Gate & H-Gate	208.266	323.529	608.306
U-Gate-Based QRNG	223.086	345.522	638.762

7 Conclusion and Future Work

To further enhance security, this study integrates QRNGs with the NIST PQC standard algorithms. A generalized QRNG architecture is proposed, from which six QRNGs are constructed. Furthermore, QRNG-based ML-KEM, QRNG-based ML-DSA, and QRNG-based SLH-DSA are developed, utilizing quantum-generated randomness for key pair generation, key encapsulation, and digital signature generation. In the experimental evaluation, the QRNGs designed in this study are validated using the methods described in NIST SP 800-90B [9], confirming compliance with both entropy and IID criteria for random bits. Furthermore, a comprehensive performance analysis of computation time is conducted to inform potential future deployments.

The primary objective of this research is to design QRNG-based ML-KEM, QRNG-based ML-DSA, and QRNG-based SLH-DSA, as well as a generalized QRNG architecture. However, the quantum computations in this study are implemented using IBM's Qiskit SDK in simulation mode. As such, the results represent simulated quantum behavior and do not embody the physical properties of actual quantum hardware. Future research is recommended to transition to real quantum chips, which can provide genuine quantum physical characteristics and generate true random numbers that satisfy the validation requirements outlined in NIST SP 800-90B.

This work serves as a PoC, demonstrated through software-based keys. For future development, it is suggested to design QRNG-based HSMs that meet practical security requirements. It is hoped that NIST will eventually provide security guidelines or specifications for QRNG-based HSMs.

Furthermore, the techniques of PQC-based Transport Layer Security could be developed for web systems.

Acknowledgments

The implementation of the QRNGs in this study was primarily conducted using the IBM Qiskit SDK, while the PQC algorithms were implemented using the BouncyCastle Open Source API. The author expresses his sincere appreciation to IBM and BouncyCastle for providing free access to these tools, which greatly facilitated the progress of this research. Some contributions of the manuscript have been published as a preprint on the web pages of arXiv (i.e. https://doi.org/10.48550/arXiv.2507.21151). This study also gratefully acknowledges the ETSI Technical Committee CYBER (Cybersecurity) for incorporating the author's related comments into ETSI TR 104 171, and the ETSI QKD Working Group for incorporating the author's related comments into ETSI GR QKD 007.

References

[1] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997, doi: 10.1137/S0097539795293172.

[2] "Module-Lattice-Based Key-Encapsulation Mechanism Standard," in Federal Information Processing Standards, FIPS 203, pp. 1–47, 13 August 2024, doi:10.6028/NIST.FIPS.203.

[3] "Module-Lattice-Based Digital Signature Standard," in Federal Information Processing Standards, FIPS 204, pp. 1–55, 13 August 2024, doi:10.6028/NIST.FIPS.204.

[4] "Stateless Hash-Based Digital Signature Standard," in Federal Information Processing Standards, FIPS 205, pp. 1–51, 13 August 2024, doi:10.6028/NIST.FIPS.205.

[5] G. Alagic et al., "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process," in NIST Interagency/Internal Report, NIST IR 8413-upd1, pp. 1–93, 5 July 2022, doi:10.6028/NIST.IR.8413-upd1.

[6] G. Alagic et al., "Status Report on the Fourth Round of the NIST Post-Quantum Cryptography Standardization Process," in NIST

Interagency/Internal Report, NIST IR 8545, pp. 1–27, March 2025, doi:10.6028/NIST.IR.8545.

[7] G. Alagic et al., "Transition to Post-Quantum Cryptography Standards," in NIST Interagency/Internal Report, NIST IR 8547, pp. 1–22, 12 November 2024, doi:10.6028/NIST.IR.8547.

[8] Y. K. Liu and D. Moody, "Post-quantum Cryptography and the Quantum Future of Cybersecurity," *Physical Review Applied*, vol. 21, no. 4, article no. 040501, pp. 1–9, 2024, doi:2331-7019/24/21(4)/040501(9).

[9] M. S. Turan et al., "Recommendation for the Entropy Sources Used for Random Bit Generation," in NIST Special Publications, NIST SP 800-90B, pp. 1–76, January 2018, doi:10.6028/NIST.SP.800-90B.

[10] P. K. Lala, *Quantum Computing: A Beginner's Introduction*, McGraw-Hill Education, New York, 2019, ISBN: 9781260123111.

[11] "IG.18 Opportunities and Challenges for Hybrid (QKD and PQC) Scenarios," Whitepaper of the GSMA, Version 1.0, pp. 1–23, 20 October 2024. [Online]. Available: https://www.gsma.com/newsroom/wp-content/uploads/IG.18-Hybrid-QKD-and-PQC-security-scenarios-and-use-cases-Whitepaper-v1.0-002.pdf

Biography

Abel C. H. Chen (Senior Member, IEEE) has published over 400 journal articles, conference papers, and patents. His contributions were published in *IEEE Transactions on Intelligent Transportation Systems, IEEE Transactions on Emerging Topics in Computational Intelligence, IEEE Internet of Things Journal, ACM Transactions on Sensor Networks, Information Science, IEEE Communications Letters*, and *Physica A: Statistical Mechanics and its Applications*. He has also submitted numerous contributions to more than 100 standards, including IEEE 1609.2, IEEE 1609.2.1, ETSI TS 102 941, ETSI

TR 104 171, and ETSI GR QKD 007. He served as the Chair for several conferences, such as AAAI-22 Workshop, WWW 2021 Workshop, IEEE BIBM 2021 Workshop, IEEE TrustCom 2021 Workshop, IEEE APNOMS 2020, and IEEE ICC 2020. He serves as an Editor for several journals, such as *Scientific Data, IEEE Open Journal of Intelligent Transportation Systems*, and *Network: Computation in Neural Systems*. He served as an Associate Editor or the Guest Editor for several journals, such as *IEEE Access, IEICE Transactions on Information and Systems, Journal of Applied Statistics*, and *ISPRS International Journal of Geo-Information*. He was listed among the Top 2% Scientists Worldwide, in 2022, 2023, 2024, and 2025 by Stanford University. Some of his publications have been recognized as highly cited papers on Web of Science using data from Essential Science Indicators (ESI).