applied sciences

MDPI

*Article*

# An Intrusion Resistant SCADA Framework Based on Quantum and Post-Quantum Scheme

**Sagarika Ghosh [1], Marzia Zaman [2], Gary Sakauye [3] and Srinivas Sampalli [1,*]**

[1] Faculty of Computer Science, Dalhousie University, Halifax, NS B3H 4R2, Canada; Sagarika.Ghosh@dal.ca
[2] Research and Development, Cistel Technology, Ottawa, ON K2E 7V7, Canada; marzia@cistel.com
[3] Research and Development, Technologie Sanstream, Gatineau, QC J8Y 2V5, Canada; gsakauye@sanstream.ca
* Correspondence: srini@cs.dal.ca

**Abstract:** The rapid emergence of quantum computing threatens current Supervisory Control and Data Acquisition (SCADA) security standards, mainly, American Gas Association (AGA)-12. Therefore, researchers are developing various security schemes based on either quantum or post-quantum algorithms. However, the efficiency of quantum algorithms impacts the security of the post-quantum digital signature scheme. We propose an intrusion resistant algorithm exploiting and applying quantum principles in the post-quantum signature algorithm. We use the Bennett 1992 (B92) protocol, a quantum key distribution scheme, to obtain the cipher, and the practical Stateless Hash-based Signatures (SPHINCS)-256 protocol to obtain a post-quantum signature. However, instead of Chacha-12, a well-known cryptographically secure pseudo-random number generator, we apply a quantum random number generator to obtain a truly random Hash to Obtain Random Subset (HORS) signature with Tree (HORST) secret key used in SPHINCS-256. We have implemented the design in Python with the Quantum Information Toolkit. We have validated the proposed algorithm using the Probabilistic Model Checking for Performance and Reliability Analysis (PRISM) and Scyther tools. Moreover, the National Institute of Standards and Technology (NIST) statistical tests show that the proposed algorithm key pairs have randomness of 98% and RSA and ECDSA are below 96%.

**Keywords:** Network security; quantum cryptography; qubits; post-quantum cryptography



**Citation:** Ghosh, S.; Zaman, M.; Sakauye, G.; Sampalli, S. An Intrusion Resistant SCADA Framework Based on Quantum and Post-Quantum Scheme. *Appl. Sci.* **2021**, *11*, 2082. https://doi.org/10.3390/app11052082

Academic Editor: Nikola Paunković and André Souto
Received: 2 February 2021
Accepted: 18 February 2021
Published: 26 February 2021



## 1. Introduction

Many industries and organizations have adopted Supervisory Control and Data Acquisition (SCADA) systems to control and monitor industrial processes such as water management, oil and gas industry, traffic management, and nuclear power plants. The significance of SCADA systems is directly proportional to the required security level to protect them from cyber attacks [1,2]. Existing standards, including the American Gas Association (AGA)-12, used in SCADA systems are vulnerable to traditional and quantum attacks. The primary quantum algorithms to crack traditional cryptographic schemes are Shor's algorithm [3] and Grover's algorithm [4]. Shor's algorithm is an approach to solve the factoring problem by finding the order or period of the particular function in the problem. A classical computer can easily calculate the periods of a function. However, for large numbers, we need a quantum computer using superposition to solve the order-finding problem efficiently. Grover's approach is a quantum search algorithm to find a particular element, given an array of $m$ elements [5,6].

AGA-12 mainly adopts three algorithms; namely, Rivest–Shamir–Adleman (RSA) for key management, Advanced Encryption Standard (AES) for encryption and Elliptic Curve Digital Signature (ECDSA) for digital signature with the Secure Hash Algorithm (SHA-1) hash function [1].

The boom of quantum computing is both beneficial and detrimental to cyber-physical systems. Gidney et al. [7] practically proved that the Shor's algorithm is capable of cracking RSA-2048 within 8 hours with 20 million qubits. However, while a classical search

algorithm takes $O(N)$, an exhaustive search by Grover's algorithm takes $O(\sqrt{N})$. Thus, it weakens the key strength of AES-*256* to only *64* bits. A hash algorithm should be both preimage and collision-resistant. However, Google has cracked SHA-1 by launching a collision attack [8]. The SHA-2,3 remains safe at present. An attack on SHA-2,3 requires on the order of $2^{256}$ queries in a non-quantum setting and, $2^{128}$ queries in a quantum setting [8]. Thus, SHA-2,3 has *security level* on the order of $2^{256}$ in classical hardware and $2^{128}$ in quantum hardware.

Quantum computing endangers stable cyber-physical security, and researchers are developing solutions to mitigate the quantum threat. The solutions involve quantum and post-quantum cryptography to secure cyber-physical systems from future attacks by quantum computing [9]. Quantum cryptography involves quantum key distribution protocols that exploit the laws of quantum physics, superposition and entanglement principle to secure against Shor's algorithm [10,11]. Whereas, post-quantum cryptography uses algorithms based on mathematical operations that are quantum-resistant and can inter-operate with classical network protocols [12].

One of the many post-quantum cryptography areas is the set of hash-based algorithms to generate a digital signature, which is secure against preimage and collision attacks by a quantum computer [13]. A first preimage attack attempts to recover a message $m$ from only the hash value such that $H = hash(m)$. A second preimage attack finds another message $n$ such that $m \neq n$ and $hash(m) = hash(n)$. A collision attack aims to retrieve two different messages $m$ and $n$, such that $hash(m) = hash(n)$ [8,14]. SPHINCS-256 is a high-security post-quantum stateless hash-based signature scheme that uses two types of signatures, namely, an extension of Winternitz One-time Signature (WOTS+) and Hash to Obtain Random Subset (HORS) signature with Tree (HORST) for few time signature [13]. SPHINCS-256 uses a pseudo-random number generator based on Chacha-12, for generating the HORST secret-key. In 2008, Bernstein et al. [15] proposed Chacha, a cryptanalysis-resistant stream cipher algorithm that maps an input stream to a novel and irreversible output stream [16]. Chacha-12 is an extension of Chacha [17].

However, Chailloux et al. [18] proposed an efficient collision search algorithm based on quantum logic gates, which reduces the *security level* of post-quantum from $2^{128}$ to $2^{119.6}$. Thus, a quantum attack requires reduced number of queries, which is on the order of $2^{119.6}$, to crack a post-quantum algorithm. Therefore, it is crucial to strengthen the security of post-quantum hash algorithms. Moreover, the emergence of quantum computing has facilitated the application of quantum physics in resource-constrained devices. For example, researchers at the University of Bristol have introduced the chip-based Quantum Key Distribution (QKD) in 2017, and the ID Quantique company has developed a true Quantum Random Number Generator (QRNG) Chip and QKD systems as well [19,20].

### *1.1. Contributions*

This paper proposes a novel security scheme for SCADA systems with the following contributions:

- We propose an intrusion immune SCADA framework by incorporating quantum and post-quantum security scheme. We use the B92 protocol to generate a secret quantum key for encryption and SPHINCS-256, a preimage and collision-resistant algorithm to obtain a digital signature with a true random number generator.
- In general, SPHINCS-256 uses a Pseudo-Random Number Generator (PRNG) based on Chacha-12 to generate a secret key of the HORST tree. However, to increase the security of SPHINCS-256, we introduce QRNG to obtain a non-deterministic and truly random HORST secret key.
- In general, SPHINCS-256 sends a message, public key and digital signature over the public channel. In our proposed scheme, we replace the message with the cipher obtained in the encryption phase.

### 1.2. Outline of the Paper

Section 2 describes the related work to our proposed scheme. Section 3 discusses the research questions for existing SCADA security standards based on current attacks and possible quantum threats. We also discuss the research questions for the post-quantum scheme, namely, SPHINCS-256. Section 4 describes the proposed framework. Sections 5 and 6 present the formal analysis and statistical analysis of the framework, respectively. Section 7 provides conclusion and scope for future work. Table 1 provides a list of abbreviations.

**Table 1.** List of Abbreviations and Notations.

| Abbreviations/Notations | Definitions |
|---|---|
| SCADA | Supervisory control and data acquisition |
| AGA-12 | American Gas Association-12 |
| RSA | Rivest–Shamir–Adleman |
| AES | Advanced Encryption Standard |
| ECDSA | Elliptic Curve Digital Signature |
| SHA | Secure Hash Algorithm |
| SPHINCS | Stateless Hash-based Signatures |
| B92 | Bennett 1992 protocol |
| BB84 | Bennett and Brassard 1984 protocol |
| QKD | Quantum Key Distribution |
| HORS | Hash to Obtain Random Subset |
| HORST | Hash to Obtain Random Subset signature with Tree |
| OTS | One-time signature |
| FTS | Few-Time signature |
| WOTS+ | Winternitz One-time Signature |
| PRISM | Probabilistic Model Checking for Performance and Reliability Analysis |
| NIST | National Institute of Standards and Technology |
| QRNG | Quantum Random Number Generator |
| PRNG | Pseudo-Random Number Generator |
| LOTSS | Lamport One-Time Signature Scheme |
| MSS | Merkle-Signature Scheme |
| RTU | Remote Terminal Unit |
| MTU | Master Terminal Unit |
| QBER | Quantum bit error rate |
| ECC | Error Correction Code |
| R-S | Reed-Solomon code |
| IoT | Internet of Things |
| $w$ | Parameter used to generate Winternitz One-time Signature |
| $M$ | Number of messages |
| $X_i$ | Public key of Merkle-Signature Scheme |
| $Y_i$ | Private key of Merkle-Signature Scheme |
| $i$ | Leaf index |
| $K$ | Number of published HORST secret key elements |

**Table 1.** *Cont.*

| Abbreviations/Notations | Definitions |
|---|---|
| $t$ | Number of HORST secret key elements |
| $n$ | Security parameter refers to bit-length |
| $m$ | Bit-length of message |
| $d$ | Layers of the hypertree |
| $h$ | Height of the hypertree |
| $N$ | Number of qubits |

## 2. Related Work

Cyber-physical systems, including SCADA systems, are threatened by classical and quantum attacks [1]. Researchers and organizations have proposed various standards and algorithms to provide resistance against these threats. We briefly review part of related work from four aspects, namely, current standards for SCADA, quantum key distribution, and post-quantum digital signature.

### 2.1. Current Standard: AGA-12

The American Gas Association (AGA) has proposed a standard with the primary goal of providing a cost and time-efficient security scheme for SCADA systems. The AGA-12 standard has three main phases, namely, key management, encryption, and digital signature [1,21] that use the following algorithms, respectively.

- RSA with a blue minimum key size of 1024 bits
- AES with a minimum key size of 128 bits
- ECDSA Digital Signature using SHA-1 with a minimum key size of 160 bits.

Gidney et al [7] have hypothetically broken the RSA algorithm using Shor's algorithm and provides estimation of the quantum resources needed. Grover's algorithm has weakened AES. ECDSA algorithm is no longer secure against quantum computing and, SHA-1 fails to provide collision resistance [22,23]. Therefore, AGA-12 is a weak standard to protect SCADA systems from existing attacks as well as possible quantum attacks.

### 2.2. Quantum Key Distribution

In a traditional key exchange protocol, the key strength depends on the mathematical complexity. Moreover, classical computers operate with 'bits' with a state 0 or 1. Shor's algorithm on quantum hardware has been proven to crack traditional cryptography like RSA. However, we can mitigate attacks based on quantum algorithms, which factorize large numbers, by using quantum key distribution (QKD). Existing QKD protocols exploit quantum physics laws and make it difficult to measure the quantum state of a qubit without disturbing its a state. The quantum computers operate with 'qubits' as the basic unit of information with a state 0 or 1 or a superposed state [9,10,24].

The two QKD protocol categories, based on the principles used, are mainly, Heisenberg's uncertainty principle and quantum entanglement. In any QKD protocol using Heisenberg's uncertainty principle, the protocol prepares a qubit such that it is not possible to copy the state of qubit without disturbing its state. It is based on the No-Cloning theorem [25]. During the transmission from sender to receiver, the disturbance in the qubit state enables the system to detect an eavesdropper. In the entanglement-based QKD protocol, both the sender and receiver use entangled photons to generate the secret key [10,11,26].

In this paper, we review two primary examples of Heisenberg's uncertainty principle-based QKD, namely, BB84 and B92 protocol. The BB84 protocol, proposed by Bennett and Brassard in 1984 [27], involves two parties generating a secret key using two channels, namely, quantum and public channel. The sender polarizes the photon by using one of the two *bases*, rectilinear and diagonal. The *basis* refers to the pairs of orthogonal states.

The rectilinear basis generates either of the two polarization states: 0°, 90°. The diagonal basis generates either 45° or 135° [9,11]. When the receiver measures the qubits, there is a 50% probability of using the wrong basis. Further, the probability of getting a correct qubit state using the wrong basis is 50%. Thus, 25% of the qubits measured by the receiver are incorrect [9,24]. In 1992, Bennett proposed a simplified version of the BB84 protocol named B92 [28]. It handles only one of two polarization states for each basis. The rectilinear basis encodes a 0-bit state into a 0° polarized state. The diagonal basis encodes a 1-bit state into a 45° polarized state. In B92, when the receiver selects the wrong basis, it does not measure anything. This condition is called erasure [11].

Both BB84 and B92 algorithms have four main phases, namely, quantum key exchange, key sifting, error calculation, and error correction and privacy amplification. The mechanism in the key sifting phase of B92 differs from that in BB84. In B92, the key sifting is more efficient since a wrong basis does not measure qubits. Unlike BB84, B92 uses only one of two polarization states for each basis, making the protocol more efficient than BB84. In BB84, the wrong basis can be used to measure qubits, and there is a 50% probability of getting the correct qubit and a 50% probability of getting an incorrect qubit. In B92, when the receiver selects the wrong basis, it does not measure anything. This condition is called erasure. Thus, the receiver does not need to worry about reading incorrect bits because of measuring them wrong.

Nurhadi et al. [11] provides a survey on QKD protocols involving BB84 and B92, performed simulations that show B92 has a lower error value than BB84. One hypothetical reason is that the number of attempts to measure the attacker's qubits is low in B92. Another reason is that in the B92 protocol, the threshold for tolerating error can be set to a lower value. The lower the threshold, the more secure the protocol.The QKD protocol focuses on secure key distribution. It mainly focuses on attacks launched by quantum algorithms that factorize more significant numbers. It does not address authentication.

#### 2.2.1. Possible Technical Limitations in QKD Deployment Scenarios

When deploying low-cost QKD; the following limitations can affect the performance of the security framework.

- Key Rate: Achieving a high key rate can be challenging while deploying the security framework in a low-cost hardware setting. For high volumes of network traffic, QKD needs to improve the development of secure key rate generation. One of the approaches is to increase the performance of the detectors. A detector with high efficiency and short resolving time can increase the key rate [29]. Eleni et al. [29]'s research shows that a QKD can be performed on an optical fiber at a data rate of 1Mbps over a distance of 50 km.
- Distance: QKD protocols provided only point-to-point communication between the RTU and MTU using a quantum channel with noise from channel imperfections. The distance between the RTU and MTU can be increased by increasing the length of the quantum channel. However, the distance between the two units is directly proportional to the noise in the channel. Thus, increasing the point-to-point distance in QKD protocols is not practically feasible. In a QKD, it is crucial to reduce the channel noise as much as possible to improve the key rate and to estimate the noise value accurately. Moreover, the low noise of single-photon detectors plays a crucial role in tolerating losses. For example, a superconducting nanowire single-photon detector in a sub-zero temperature can tolerate a loss of 72 dB, which is equivalent to around 360 km of standard single-mode fiber [29].

### *2.3. Post Quantum Digital Signature Scheme*

A classical exhaustive search algorithm costs $O(N)$ and a Grover's search costs $O(\sqrt{N})$. A preimage and collision attack on existing classical hash functions can be launched using Grover's algorithm with quantum settings [30]. Both attacks need $2^{256}$ queries in non-quantum hardware, and $2^{128}$ queries in a quantum black box model [22].

Therefore, researchers have proposed various post-quantum digital signatures and hash algorithms for authentication.

There are four families of post-quantum cryptography, namely, lattice-based, multivariate, hash-based, and code-based cryptography. In this paper, we briefly review the following hash signature schemes.

#### 2.3.1. Lamport Signature

Lamport One-Time Signature Scheme (LOTSS) uses any secure cryptographic hash functions to generate a public key to sign a message. The advantage of using any hash function makes LOTSS versatile. However, LOTSS becomes insecure if there is a possibility of hash function exposure. Another disadvantage is the large size of the public and private keys. For example, a message M = $\{0,1\}^k$, when $k \in \mathbb{Z}^+$, will have 2*$k$ hash values from the hash function of at least 160 bits to provide a security level of minimum $O(2^{80})$. Thus, both private and public key size is of at least 160*2*k = 320*k bits [31].

#### 2.3.2. Winternitz One-Time Signature Scheme

To address the large key size of LOTSS, Winternitz One-time Signature Scheme (WOTS), an optimized version of LOTSS, uses a parameter $w$. A larger value of $w$ means smaller signature size. However, the time cost of the WOTS scheme increases as $w$ increases. The signature size decreases linearly, and the time cost increases exponentially [31].

#### 2.3.3. Merkle-Signature Scheme

The main disadvantages of the one-time signature schemes (OTS) are (i) large public key size and (ii) the need for a new public key generation for each message. The Merkle-Signature Scheme (MSS) addresses the flaw of OTS by generating one public key to sign more than one message. The number of messages should be a power of 2 such that M = $2^n$. It generates the public $X_i$ and private keys $Y_i$ such that $Y_i$ is within $1 \leq i \leq 2n$ and hash function is $h_i = H(Y_i)$. It creates a hash tree with leaf level $i = 0$ and the root level $i = n$ [31].

#### 2.3.4. HORS and HORST

In 2002, Reyzin et al. [32] proposed a few-time signature scheme using hash functions. It uses two parameters $t = 2^\tau$ for $\tau \in \mathbb{N}$ and $k \in \mathbb{N}$. The hash value of message is $m$ such that $m = k \log t = k\tau$. HORS uses two keys, a secret key consisting of $t$ random values obtained from a pseudorandom number generator. The public key is obtained from the $t$ hashes of the random values in the secret key, and the signature contains $k$ secret key values [33].

In 2015, Bernstein et al. [13] proposed HORS tree (HORST) for the SPHINCS protocol, which we have used in our proposed framework. HORST reduces the public key size and signature of the HORS scheme. However, it increases the runtime of the algorithm. The public key generated in HORST is the root of a binary hash tree of height $\log t$. The leaves of the tree are public-key elements of the root key. The signature generated has $k$ secret values and authentication path for each value [13]. SPHINCS-256, exhibited in our proposed scheme, comprises Merkle tree, WOTS, and HORST signature scheme. Due to the rapid inception of quantum computing, quantum algorithms have the potential to crack long term security protocols, and improving the efficiency of quantum algorithms can crack post-quantum schemes. Therefore, we need to build up a security system that mitigates such threats. Table 2 provides a synthesis of all the research work related to the proposed algorithm.

**Table 2.** Related algorithms to provide security current and possible attacks.

| Authors | Algorithm/ Organization | Overview | Advantages | Disadvantages |
|---|---|---|---|---|
| American Gas Association | AGA-12 | AGA-12 was developed to provide security as well as to save time and computation cost. | It provides confidentiality, integrity and authentication. | It fails to provide availability and is vulnerable against quantum attacks. |
| Bennett et al. 1984 | BB84 | It is Heisenberg's uncertainty principle based QKD proposed by Bennett and Brassard in 1984. It uses two polarization states for each basis. | It is resistant against quantum attack, mainly, Shor's algorithm. | There are possible practical challenges, based on distance and key rate, in real time application. |
| Bennett 1992 | B92 | In 1992, Bennett proposed a simplified version of the BB84 protocol named B92. It handles only one of two polarization states for each basis. | It is resistant against quantum attack, mainly, Shor's algorithm. | There are possible practical challenges, based on distance and key rate, in real-time application. |
| Lamport 1992 | LOTSS | It uses any secure cryptographic hash functions to generate a public key to sign a message | It is versatile. | Exposure of hash function reduces the security of LOTSS. It generates large key size. |
| Robert 1979 | WOTS | It is an optimized version of LOTSS. | It has smaller signature size. | The signature size decreases linearly, and the time cost increases exponentially |
| Merkle 1979 | MSS | It is a stateful hash-based signature scheme which creates a single binary hash tree to generate signature and key pair, using OTS scheme. | It uses one public key to sign multiple messages. | The signature size and key sizes have been improved later in the extension of MSS. |
| Leonid et al. 2002 | HORS and HORST | HORS is a few-time signature scheme using hash functions. HORST is HORS with a binary hash tree. | Both of them efficiently generates quantum safe signatures. HORST reduces the public key size and signature of the HORS scheme. | When compared to WOTS, the signature size is relatively larger. HORST increases the time cost of the HORS algorithm. |
| Bernstein et al. 2015 | SPHINCS | It is a stateless hash signature scheme using hyper-tree with OTS and FTS. | It efficiently generates quantum safe signatures. It provides $2^{128}$ security against quantum gates. | Chailloux et al. [18] proposed an efficient algorithm to reduce the security level of post quantum from $2^{128}$ to $2^{119.6}$ . It sends message, public key and digital signature over the public channel |
| The authors. 2020 | The proposed algorithm | It uses an intrusion resistant security framework based on quantum and post-quantum security scheme. | It replaces the message in SPHINCS-256 with cipher. It uses quantum random number generator to obtain truly random HORST secret key. | The computation cost of our proposed SPHINCS-QRNG is higher than that of AGA-12 and the existing SPHINCS-PRNG algorithm |

## 3. Research Questions

We propose a collision-resistant framework for SCADA systems which uses both quantum and post-quantum algorithm. We developed the proposed scheme based on the two sets of research questions.

### *3.1. Set1*

Set 1 focuses on the research problem of AGA-12.

- Does AGA-12 provide confidentiality, integrity and authentication against quantum attacks?
- Does AGA-12 key management provide resistance against Shor's algorithm?
- Will AGA-12 encryption and digital signature provide enough resistance against Grover's algorithm?

### *3.2. Set2*

Set 2 focuses on the research problem of the post-quantum digital signature, namely, SPHINCS-256. In 2017, Chailloux et al. [18] proposed an efficient quantum collision search algorithm and presented its implications on symmetric cryptography. The authors also proved that it reduces the security level of post-quantum algorithms from $2^{128}$ to $2^{119.6}$. It reckons the following research questions, which we address in our scheme.

- Can we increase the security level of SPHINCS-256?
- Can a post-quantum algorithm use the laws of quantum physics to increase security?

## 4. Proposed Scheme

To address the research questions in Set 1, we developed an algorithm that includes the following three phases: key generation, encryption and authentication. We have used the B92 protocol for key generation and encryption, and SPHINCS-256 for integrity and authentication. Moreover, to address the research questions in Set 2, we have modified the SPHINCS-256 algorithm. The SPHINCS-256 algorithm has four main components, namely, PRNG based on Chacha-12 algorithm for HORST secret-key generation and expansion, hashing in WOTS and HORS public-key generation, hashing in trees, Chacha Permutation used in trees. Furthermore, the sender transmits a combination of signature, message, and SPHINCS-256 public key to the receiver. In our proposed algorithm, we use the quantum random number generator (QRNG), instead of pseudo random number generator (PRNG), to generate a truly random HORST secret key. Instead of the message, the sender sends the cipher along with signature and public key to the receiver. Figure 1 provides a brief overview of the steps in our proposed scheme.

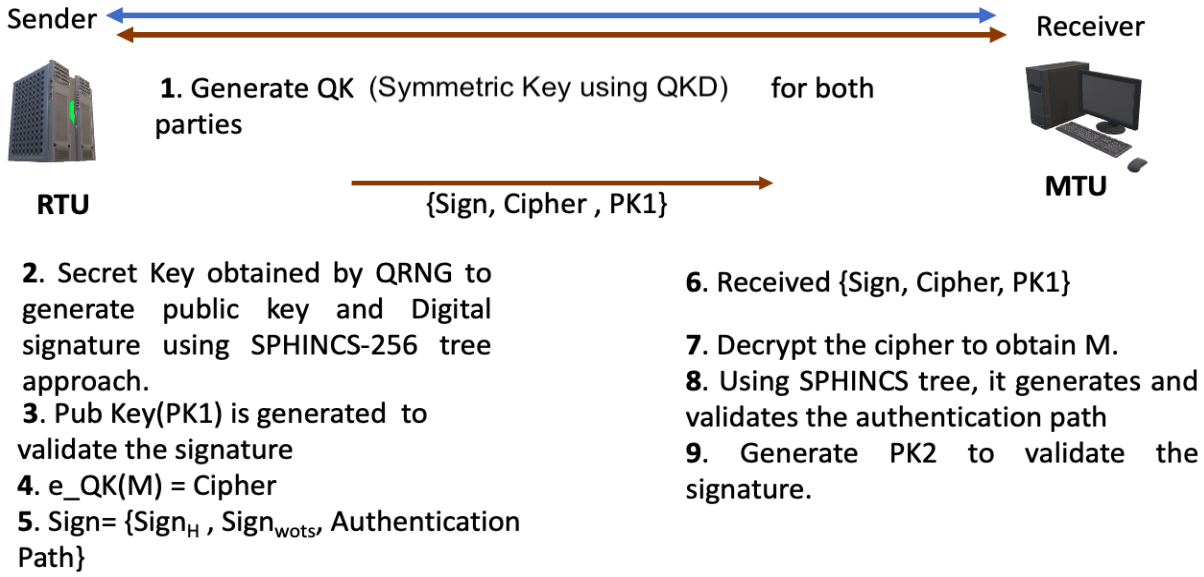


**Figure 1.** The Proposed Scheme Model

In the following, we present an overview of our proposed scheme.

- Key Generation: The quantum key is generated from the B92 protocol.
- Encryption: The quantum key is used to encrypt the message to obtain the cipher.
- Authentication: The SPHINCS-256 protocol is applied to generate a signature and public key for verification and validation.

### *4.1. Key Generation*

This subsection explains the first phase of our proposed algorithm. It explains the mechanism of the B92 protocol to generate a key for encryption. For a better and easier understanding of the B92 protocol, we have included the key sifting phase. However, key sifting of B92 is slightly different from that in BB84.

We assume that the Remote Terminal Unit (RTU) is the sender, and Master Terminal Unit (MTU) is the receiver. The key generation phase exhibits the B92 protocol, which comprises of the following sub-phases to generate a secret key, using quantum and public channel [11,34].

#### 4.1.1. Raw Key Exchange

The RTU generates a string of raw qubits with the help of either of the two basis, rectilinear and diagonal, randomly. The rectilinear basis polarizes the 0-bit into a $0^{\circ}$ polarized photon, and the diagonal basis polarizes the 1-bit into a $+45^{\circ}$ polarized photon. The RTU sends the sequence of superposed state qubits, called raw qubits, to the MTU. Furthermore, the raw key exchange utilizes the quantum channel, and the other sub-phases use the public channel.

#### 4.1.2. Key Sifting

The MTU has two sets of basis to measure the incoming string of qubits. The MTU has the same pair of basis. However, the analyzer of MTU has polarisation states orthogonal to that of the RTU's. The MTU has a rectilinear basis with polarization state $90^{\circ}$ and a diagonal basis with polarization state $-45^{\circ}$.

When the MTU can measure the photon with polarisation of $90^{\circ}$, that means the RTU must have sent the qubit with polarization of $+45^{\circ}$ and hence the state of the qubit is 1. Whereas, when the MTU can detect the photon with polarization $-45^{\circ}$, the state of qubit must be 0. The MTU, measuring with a wrong basis, fails to read the qubit. The MTU sends the string of the wrong basis to RTU over the public channel. The RTU discards the bit corresponding to the MTU's wrong basis and obtains the sifted key.

#### 4.1.3. QBER Calculation

Both RTU and MTU extract a small portion from their sifted key and use it to calculate the error rate. The RTU obtains the quantum bit error rate (QBER) by computing the ratio of the number of errors and the total number of bits in MTU's extracted key. Both the units discard the extracted portion and acquire the sub-sifted key. In our simulation, we set the threshold of noise or QBER up to 25%. If QBER $>$ 25%, both units discard the process, and when the QBER $\leq$ 25%, both units proceed to the next sub-phase.

#### 4.1.4. Error Correction Code and Privacy Amplification

To resolve the errors in the subsifted key, both the units exhibit Reed-Solomon (R-S) code. It is an error reconciliation algorithm that corrects multi-bit error with a low computational overhead [35,36]. The R-S code also evaluates the confidentiality and integrity of the subsifted key. The RTU encodes the key, which involves adding extra parity bits and sends the codeword to the receiver. The MTU receives the codeword, decodes it, and resolves the errors. Meanwhile, the eavesdropper fails to read the subsiftedkey [36,37].

The Reed Solomon(RS) error correction code has two modules: RS encoder and RS decoder. The RS encoder module involves encoding the information with a Generator polynomial to obtain the codeword. Therefore, it provides confidentiality to the informa-

tion, however not to the codeword. The codeword is sent to the decoder, which applies a decoding algorithm to correct the errors in the received codeword. When both units perform error correction, they obtain the same subsifted key. However, to reduce any information leakage during the R-S code, the MTU and the RTU hash the subsifted key to obtain the final key. This mechanism is called privacy amplification, as it increases the key secrecy.

#### 4.1.5. Classical Channel Authentication in QKD

The primary contribution and focus of our research are to provide message authentication and integrity by using SPHINCS-256 with QRNG, instead of PRNG, in HORST and by considering quantum key derived from QKD.

Our proposed algorithm has the basic steps which are predefined in QKD, namely, Raw key generation (via Quantum channel), Key Sifting, Error Correction Code (ECC), and Privacy Amplification (all three sub-phases via classical channel). The basic steps ensure the key distribution of the quantum key. However, the classical channel in QKD must be authenticated.

To address the above concern, we have introduced SPHINCS-256 in QKD to authenticate the classical channel. Figures 2 and 3 provides an overall structure of our proposed algorithm. Figure 2 illustrates all the sub phases of QKD. After Raw Key Generation, the key sifting and QBER involves sending a series of wrong basis and extracted portion of sifted key from the receiver to the sender over classical channel. During QBER and error correction code phases, the sender sends the codeword over the classical channel. Each information exchanged over the classical channel during QKD is fed to SPHINCS-256 for message authentication.

Our proposed algorithm includes symmetric key generation using quantum key distribution as opposed to relying on trusted third party or a pre-shared secret key scheme. The QKD, itself, does not rely on a pre-shared secret key scheme or a trusted third party. Moreover, the QKD uses a quantum-resistant post-quantum signature scheme during the classical communication phase. Thus, the proposed integration of QKD and SPHINCS-256, described in Figures 2 and 3, provides intrusion detection and security against both quantum and classical attacks, unlike other traditional algorithms. Moreover, for our future work, we will be performing comparative analysis of QKD authenticated by SPHINCS-256 and QKD authenticated by other universal hash algorithms.

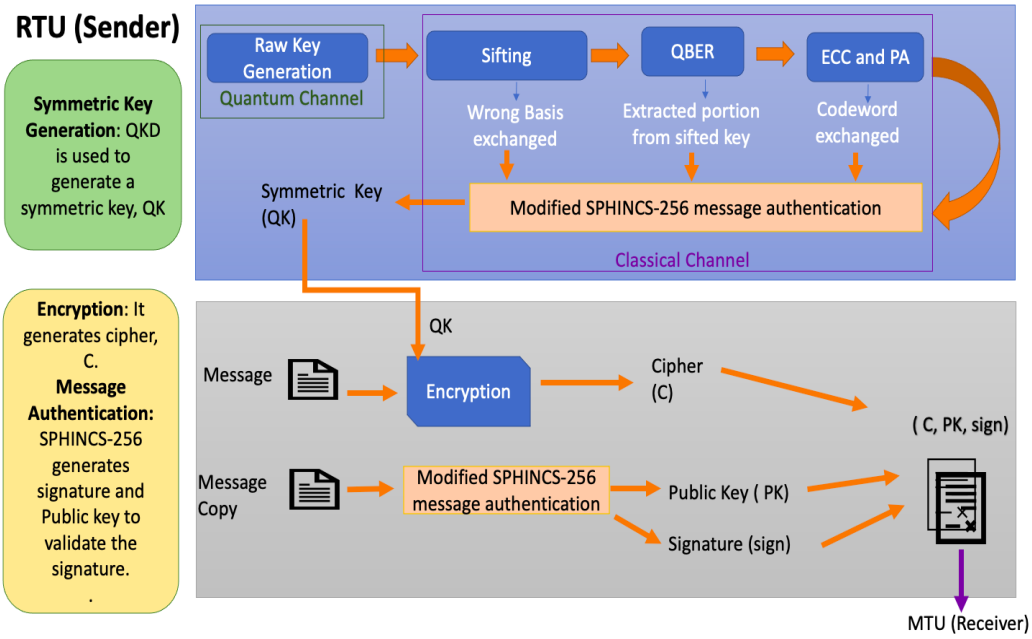


**Figure 2.** The structure of our proposed scheme. It includes symmetric key generation, Encryption and message authentication.

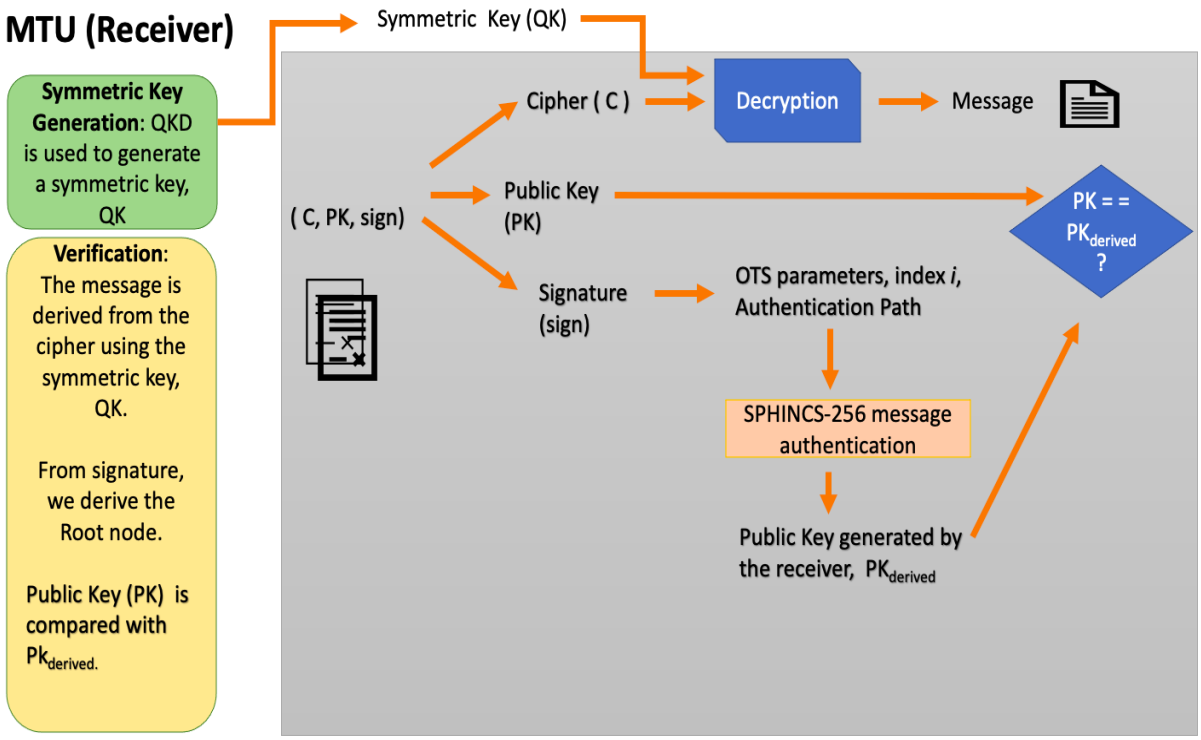


**Figure 3.** The structure of our proposed scheme. It includes decryption and signature validation.

#### 4.1.6. Impact of the Use of QKD Technology on the Network Topology

A typical SCADA network topology consist of two primary units: (1) Master Terminal Unit acting as the control center (2) Remote Terminal Unit acting as the field site which gather information from sensors [1,38,39]. Our proposed algorithm mainly focuses on securing communication between the RTU and MTU. The communication link between the MTU and the RTU in the SCADA network topology is a point-to-point link.

When QKD is used on the network topology, the RTU and MTU are connected by two channels: Quantum Channel (fiber optic) or Classical Channel (Internet or fiber optic). If the SCADA network topology is based on hardwired communication, we can say that 2*N fiber optics are needed to deploy QKD [40]. The point-to-point link provides low latency and fast communication. One major constraint of QKD on point-to-point network topology is that if one of the communication links is broken, the entire network fails [39,41]. However, the impact of authenticated QKD in SCADA network also includes resistance against intrusion and quantum attacks.

For future work in practical QKD research, a quantum channel can be designed which is capable of carrying both quantum and classical information. It may lead to reduction of number of fiber optics or, additional communication channel in a network topology.

### *4.2. Encryption and Digital Signature Algorithm*

The procured quantum key encrypts the message to generate cipher. It addresses information confidentiality. We use this generated cipher in the SPHINCS-256 algorithm and send it along with the signature and public key via the public channel. We use the stateless hash signature scheme, namely, SPHINCS, with a hyper-tree of height $h$ and $d$ layers of trees. Each tree uses the Merkle approach and has a height of $h/d$. Between the trees, it uses Goldreich's construction by applying WOTS as a one-time signature (OTS) scheme [13]. However, to sign the messages, the HORST scheme is used as a few-time signature. Using hyper-tree and Goldreich's construction significantly reduces the total tree height and reduces the signature size. We discuss the various aspects of SPHINCS-256 in the following sections.

#### 4.2.1. Security Parameters of SPHINCS-256

In our proposed scheme, we use SPHINCS-256 to generate digital signature for integrity and authentication. It uses the following security parameters to provide a quality trade-off between speed and signature size [33].

- Security parameter $n$ referring to the bit-length of hashes in HORST and WOTS, $n = 256$.
- Bit length of message, $m = 512$.
- Height of the hypertree, $h = 60$.
- Layers of the hypertree, $d = 12$.
- WOTS parameter, $w = 16$, used for generating One-time signature.
- Number of HORST secret key elements, $t = 2^{16}$.
- Number of published HORST secret key elements, $k = 32$.

#### 4.2.2. Security Level of SPHINCS-256

As mentioned above, Bernstein et al. [13] claim that SPHINCS-256 provides $2^{128}$ security against the quantum computing attack. However, Chailloux et al. [18] proposed an efficient quantum algorithm that claims to trim the post-quantum security from $2^{128}$ to $2^{119.6}$. Moreover, Philippe et al. [42] prove a practical attack exploiting the greedy algorithm on the HORS signature scheme. To mitigate such quantum threats, we need to increase the security of the post-quantum SPHINCS-256 scheme. In our proposed algorithm, we infuse the laws of quantum physics in SPHINCS-256, by using a quantum random number generator and a quantum key management algorithm.

#### 4.2.3. SPHINCS-256 Tree Construction and Generating Digital Signature

Figure 4 provides the general construction of SPHINCS tree. Figure 5 provides the mechanism of private and public key generation of both original and modified HORST algorithm. In the following paragraphs, we first explain the construction of SPHINCS tree and then dive into HORST private key generation.

Figure 4 provides a generic construction of SPHINCS tree, with $h = 9$ and $d = 3$. The tree provides a hyper tree construction which includes layers of Merkle's tree with OTS nodes as WOTS (+) scheme. At the bottom of the tree lies the FTS nodes denoted as $\diamondsuit$. The Few Time Signature (FTS) scheme used in SPHINCS is HORST algorithm [43]. The HORST algorithm has two keys, namely, the internal secret key or the private key, and the public key. We have made modification in HORST.

Generally, the HORST algorithm takes as input the seed and the bitmask Q. As shown in Figure 5, the initial secret key $SK_H$ is obtained by feeding seed into a PRNG denoted as $G_t$. The $G_t$(seed) generates $SK_H = SK_1, SK_2, \ldots, SK_t$ [13]. However, in our proposed SPHINCS algorithm, the HORST replaces $G_t$ with $QRNG_t$.

QRNG is a type of TRNG deployed in quantum hardware and it does not rely on seeding and a deterministic algorithm since it generates random numbers from measuring or observing quantum processes. A practical QRNG includes a source of randomness and, a measuring or detection system. The source of randomness is the entropy source for generating qubits. For example, an optical QRNG inherits randomness from quantum states of light with the help of a photon source, a polarizing beam splitter and detection systems [44–46].

The $QRNG_t$ generates the truly random secret key $SK_H = SK_1, SK_2, \ldots, SK_t$. It does not need the seed value. The HORST tree is a binary tree constructed using bitmask $Q$ and, the leaves, $Li$, of the tree is computed using cryptographic hash function F on the elements of the secret key, SKi. Thus, we can denote it as $Li = F(SK_i)$ for $i \epsilon [t\text{-}1]$. And the root node of the constructed binary tree on the $L_i$ is the public key of HORST.

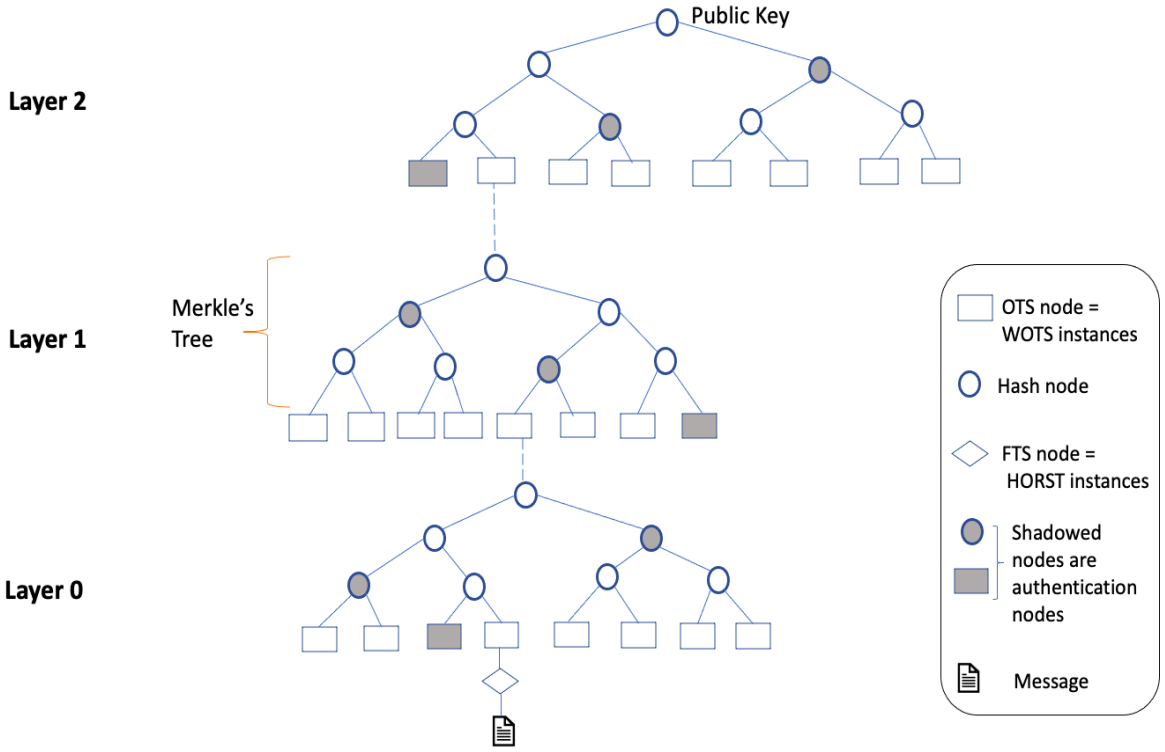


**Figure 4.** General Construction of SPHINCS tree with $h = 9$ and $d = 3$.

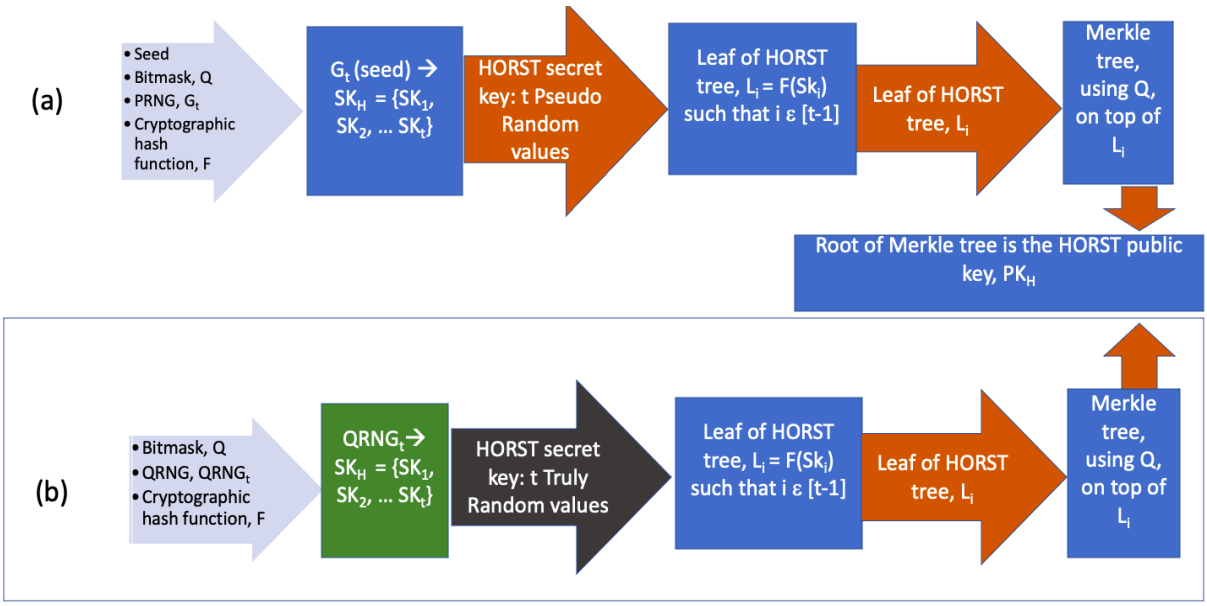


**Figure 5.** Key Generation: (**a**) Original HORST algorithm and (**b**) Proposed HORST algorithm.

SPHINCS-256 tree is a combination of four types of trees: Hyper-tree that includes Merkle's, WOTS, and HORST. The hyper-tree comprises of Merkle's tree connected by WOTS key pairs. Furthermore, the leaves of the SPHINCS-256 are HORS trees. Using the above-mentioned security parameters, the height of each Merkle tree is $h/d = 60/12 = 5$. The HORST tree follows a Merkle's construction with height $\tau = \log_2 t$ [13].

The SPHINCS-256 follows a stateless Goldreich Signature scheme. Each node of the tree has an OTS pair. The key pair at the root, at the outermost layer ($d - 1$), has SPHINCS-256 public key, $PK_H$ and private key, $SK_{d-1}$ as shown in Figure 6. However, the message is first signed with the FTS scheme, named HORST, situated is at the bottom of the tree. We used the following steps to generate the signature.

- We obtain the HORST secret key using QRNG, and using the HORS tree algorithm, we obtain its root key, which is the public key. The obtained HORST instances are used to sign the message.
- The obtained HORST signature comprises of k keys, and their respective authentication paths are a part of the SPHINCS-256 signature.
- We then sign the public key of each tree, obtained from the lower layer trees, with WOTS instances as we climb the SPHINCS-256 tree. The signatures obtained in each layer along with its corresponding authentication path to a public key, the root of SPHINCS-256 tree.

After obtaining the signature, the RTU sends the concatenation of SPHINCS-256 public key *PK*, cipher obtained, and the signature generated by SPHINCS-256 algorithm.

#### 4.2.4. Verification

The MTU receives the package and deciphers the cipher using the same quantum key and, obtains the signature $Sign_h$ , $Sign_{wots}$ , *Authentication Path*. The RTU derives the digest of cipher by using the OTS parameters hidden in *Sign*. Using the SPHINCS tree algorithm, it generates and validates the authentication path. The MTU also obtains its public key, $PK_{derived}$, to validate the signature.

### *4.3. Relationship between Keys and Their Derivations in Our Proposed Scheme*

In the following paragraphs, to address the relationship between keys and their derivations, we explain our proposed main framework, and then explain how the SPHINCS-256 public key is generated and to whom it is related or derived from. We have also added three figures. Figures 2 and 3 provide the structure of our proposed scheme. Figure 4 provides the general construction of a SPHINCS tree. Figure 6 shows the computation of SPHINCS at the topmost layer of the tree.

In our proposed framework, as shown in Figures 2 and 3, we use two primary keys:

1. Symmetric Key, QK, derived from Quantum Key Distribution.
2. A public key, PK, derived from SPHINCS-256 algorithm.

Flow of the proposed algorithm: The symmetric key, QK, finalised from the authenticated QKD, is used to encrypt the message (data gathered by RTU) to generate cipher. The sender copies the message before encryption, and the message copy is fed to the SPHINCS-256 to generate signature and the public key, PK. A tuple containing Cipher, Signature and public key is sent to the receiver. The public key, PK, is used to validate the signature when received by the receiver.

From the above explanation, we can conclude that the public key, PK, of SPHINCS-256 is not derived from the symmetric key, QK. This feature makes the framework secure since the public key does not carry the essence of the symmetric key, QK because QK is used to encrypt the message.

However, the SPHINCS-256 algorithm has key generation phase which generates its own key pair, (Public key, Private key). The Public key is the PK used in our main framework, explained above and in Figures 2 and 3.

**Key Generation in SPHINCS-256 [13,43]:** The SPHINCS-256 tree is based on hyper tree construction. When the SPHINCS-256 tree is seen as bottom to top approach, the bottom layer of tree has leaf trees as HORST tree with its own leaves. The leaf of the HORST tree is denoted as $L_i$ and the root of the HORST tree is $PK_H$. The $L_i$ is computed by hashing the elements of HORST secret/private key, $SK_H$. In Figure 4, we can see that the root node of the below tree is computed to generate the leaf of the following above tree which further generates its own root node. Thus, in simple terms, we can say that the private key of HORST tree is further computed, passing through the main tree via FTS node, OTS nodes and hash nodes, to generate the root of the main tree, SPHINCS-256 public key. Thus, SPHINCS-256 public key, PK, is deterministically derived from HORST private or secret key, $SK_H$.

In our proposed scheme, only the HORST secret key, $SK_H$ , is generated by QRNG. Thus, the QRNG is used only once in the SPHINCS-256 tree. And, it is used only in the HORST tree.

For a detailed view, Figure 6 shows a SPHINCS tree with *d* layers, and the root computation of SPHINCS-256 public key, PK, at *d-1* layer. The *d-1* layer has the secret key, $SK_{d-1}$, computed and passed on from the below trees. The $SK_{d-1}$ is further hashed to generate seed. The seed along with bitmask is fed to the WOTS scheme to generate public key of WOTS, $PK_{WOTS}$ . The Leaf, $L_{i(d-1)}$ is computed by hashing $PK_{WOTS}$. And, the $L_{i(d-1)}$ is fed to binary hash tree to obtain the root of the main tree, PK.

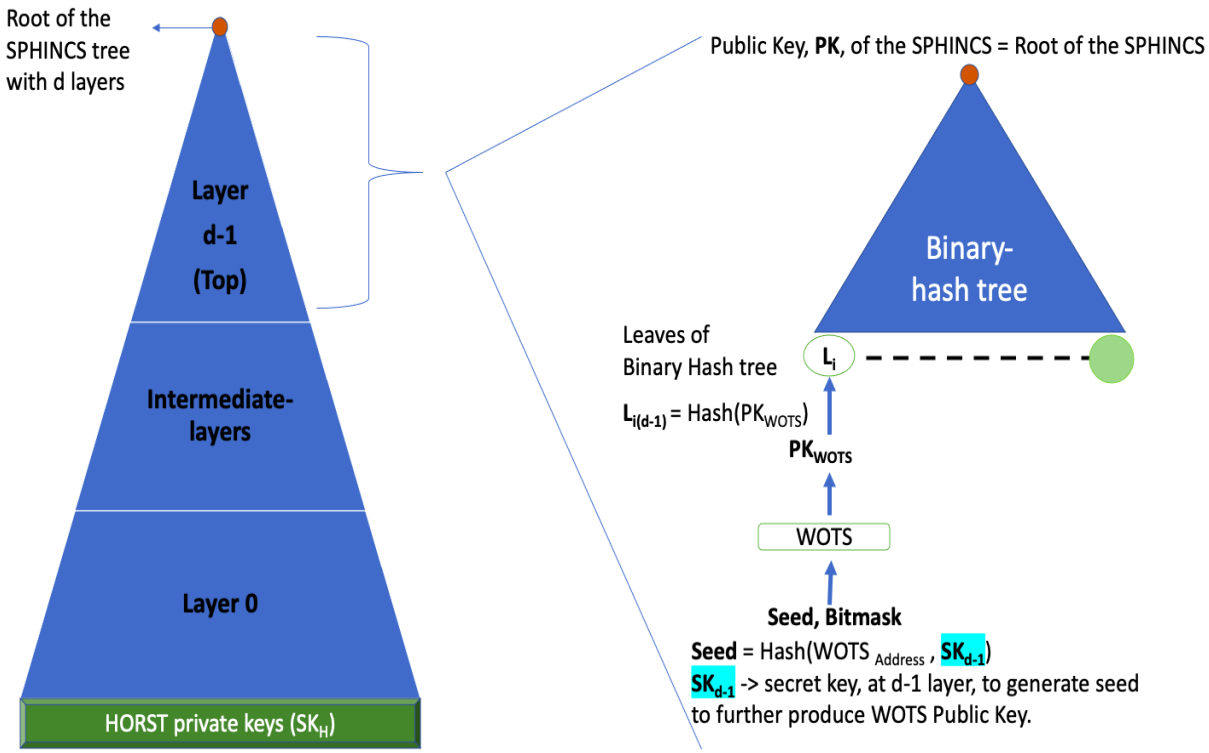


**Figure 6.** SPHINCS tree with d layers. It shows the computation at d-1 layer. The d-1 layer is the topmost layer which has key pairs: (1) Public Key, PK. (2) Private or secret key, $SK_{d-1}$.

## 5. Formal Analysis of the Proposed Scheme

We have performed a formal analysis of our proposed scheme. We used PRISM [47], a probabilistic model checker, for key generation and Scyther [48] for encryption and digital signature.

### *5.1. Analysis of Key Generation Phase*

We verified the B92 protocol based on the Discrete-Time Markov chain by using the PRISM tool [49]. There are three main steps to build a PRISM model. The first step is system specification, involving building a model of a given system with modules, variables, and constants. We constructed three modules, mainly Alice, Bob, and Eve. The second step is property specification, which addresses the two hypothetical questions: How much information is leaked processing the protocol? Can the B92 protocol discard or prevent the eavesdropping attack? Thus, we created two properties to address the questions, namely, *Probability of detecting an eavesdropper (Eve)* and, *Probability that Eve measures more than half of the information correctly*. The third step is feeding the model into the PRISM tool [49]. We created two models [50,51], namely, B92 protocol with Intercept Resend attack, and B92 protocol with Random Substitute attack.

PRISM is a probabilistic model checker that indicates design flaws in the security protocol before moving to the simulation phase and deploying in a real-time hardware setting. Thus, PRISM does not address the challenges during hardware implementation. Based on Sibson et al. [20] 's experiment on the BB84 protocol, the researchers have observed that the secret key rate is 345 kbps with a clock rate of 560 MHz and QBER of 1.05%. Moreover, Rishab et al. [52] 's paper provides both experimental implementations and software simulation of the B92 protocol. The practical implementation shows the key rate and QBER is 51 ± 0.5 Kbits/sec and 4.79% ± 0.01%, respectively. The B92 protocol simulation generates a key rate of 52.83 ± 0.36 Kbits/s and QBER of and 4.79% ± 0.01%.

While conducting formal analysis using PRISM, we considered the worst-case scenario of information leakage. The worst-case scenario is that Eve reads more than half of the qubits over the quantum channel correctly. Because, When Eve measures the qubits, there is a 50% probability of using the wrong basis. Further, the probability of getting a correct qubit state using the wrong basis is 50%. Thus, 25% of the qubits measured by the receiver is incorrect, and 75% of the qubits measured are correct. Therefore, currently, we are considering more than 50% information leakage. Moreover, simulation of B92 in PRISM deals with a low number of qubits due to computational limitations and significantly high elapsed time.

#### 5.1.1. B92 with Intercept-Resend Attack Model

The Intercept-Resend attack model is based on active eavesdropping. Eve tries to read the qubits or bits of information, exchanged through quantum and public channels, during key generation [49]. Figure 7 shows the probability of detecting Eve. Figure 8 shows the probability of detecting information leakage more than N/2, while N is the number of qubits.

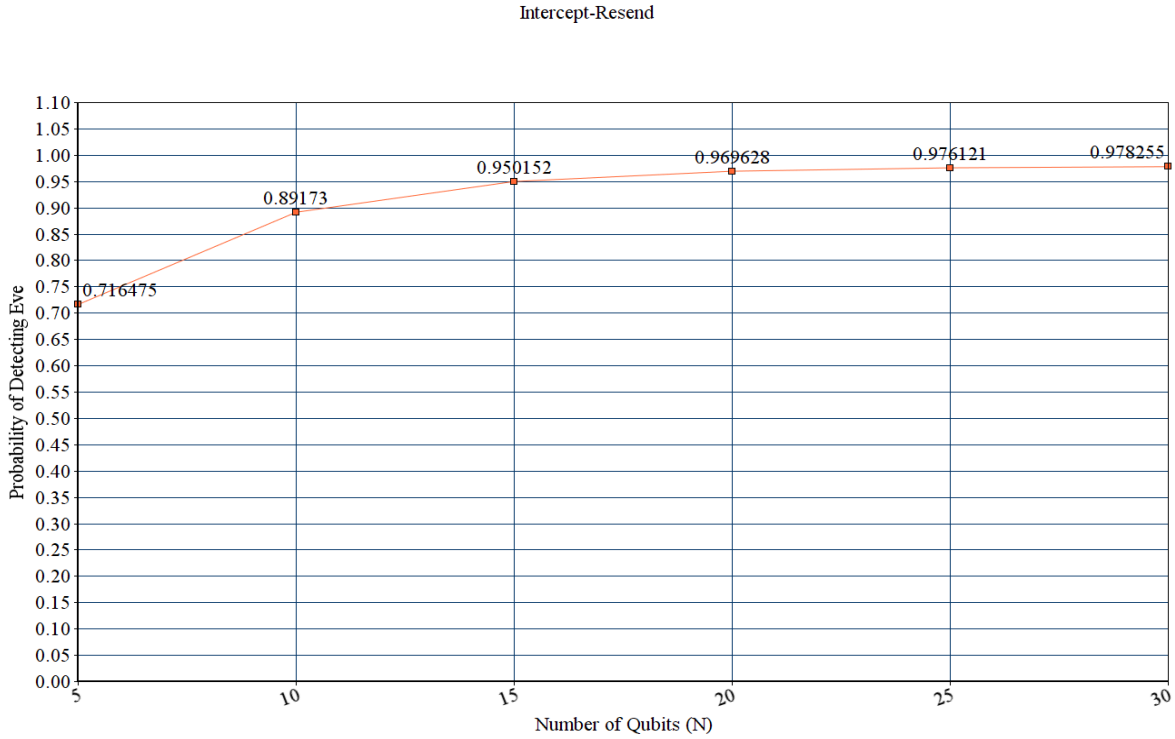


**Figure 7.** Probability of detecting Eve during Intercept Resend Attack.

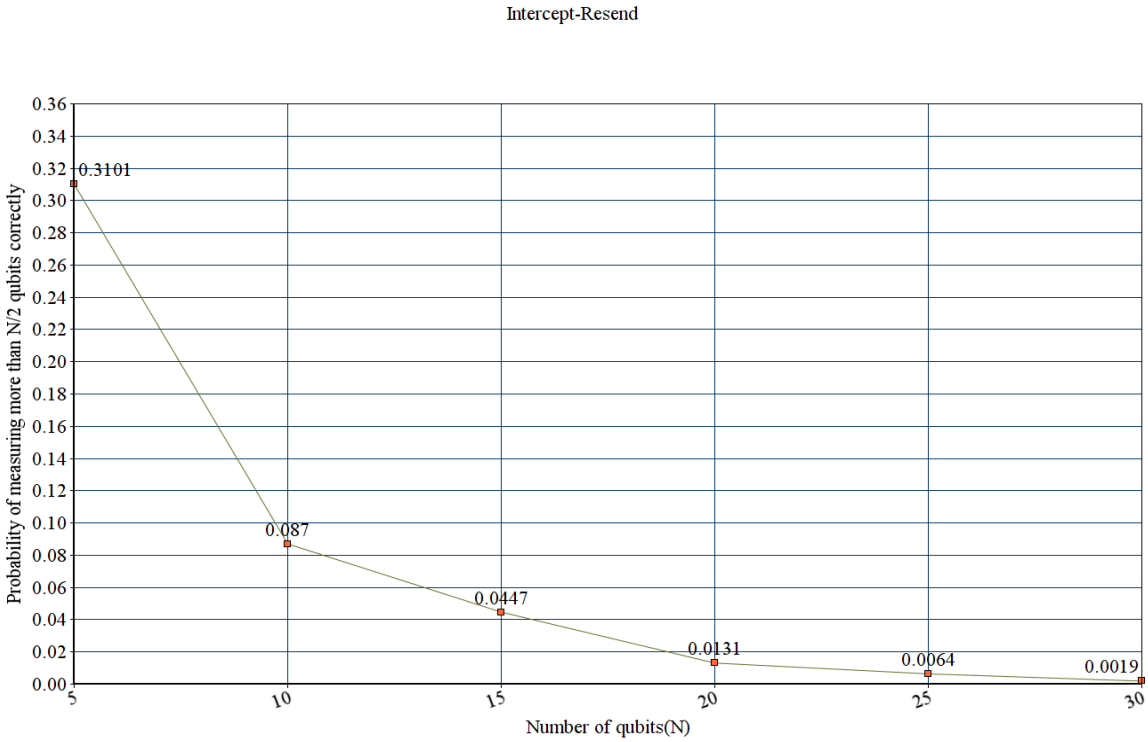


**Figure 8.** Probability that EVE measures more than half of the information correctly during Intercept Resend Attack.

#### 5.1.2. B92 with Random-Substitute Attack Model

A Random-Substitute attack is a cloning attack. The eavesdropper measures the qubits, which disturbs the state of the qubit. The eavesdropper attempts to duplicate the original state of the qubit and sends it to Bob. Figure 9 shows the probability of detecting Eve. Figure 10 shows the probability of detecting information leakage more than $N/2$ by Eve, where $N$ is the number of qubits and $N \in \mathbb{Z}^+$.

Therefore, we conclude that the probability that Eve is detected increases exponentially with the number of qubits. Furthermore, the probability that an Eve obtains a correct measurement result for over half the transmissions decreases exponentially with $N$. We also conclude that an Intercept-Resend attack is more plausible to cloak an Eve's presence than a Random-Substitute attack.

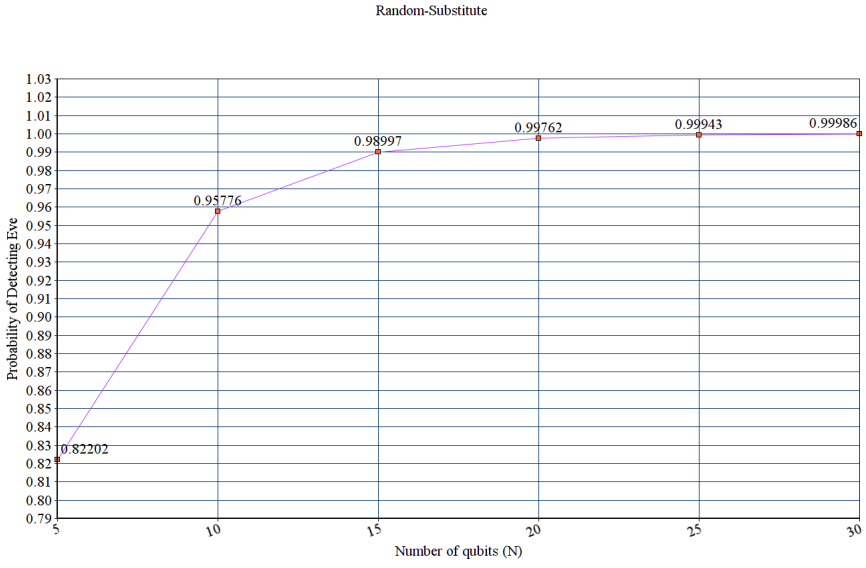


**Figure 9.** Probability of detecting Eve during Random Substitute Attack.

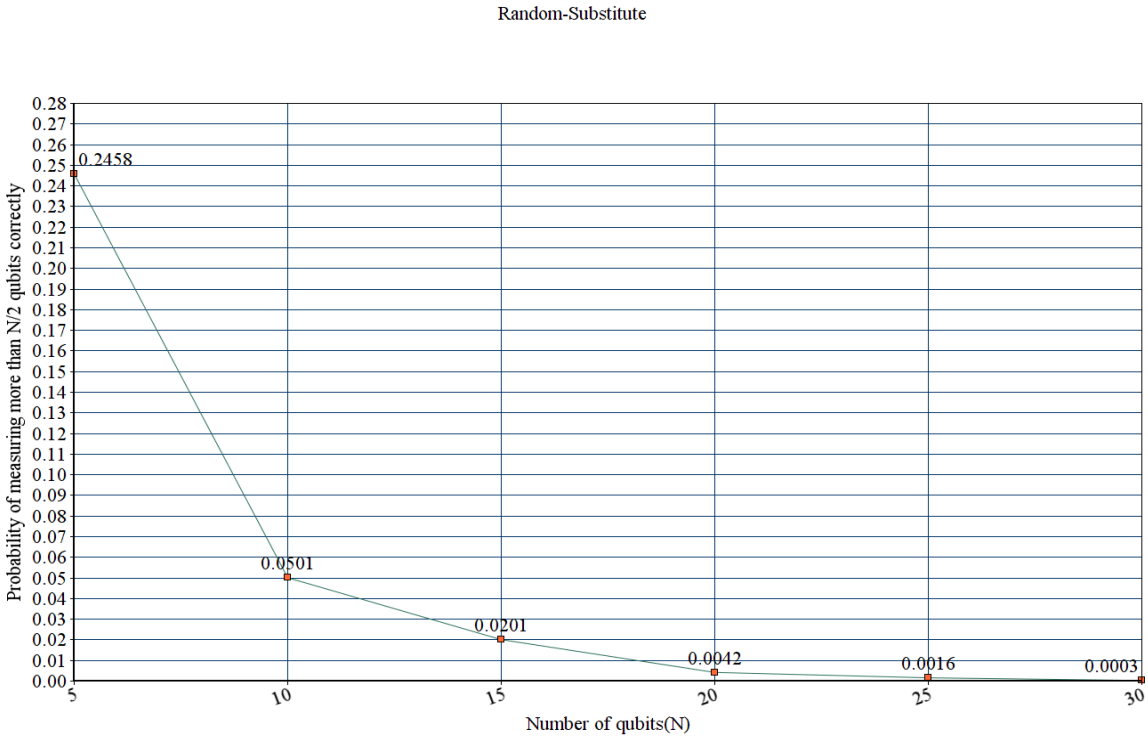


**Figure 10.** Probability that EVE measures more than half of the information correctly during Random Substitute Attack.

### *5.2. Analysis of Encryption and Digital Signature Phase*

We use Scyther to verify the post-quantum scheme used in our proposed framework. Scyther is an open-source tool used for automatic verification of security protocols [48]. The verification is based on three main aspects: (1) Logical message components verify whether a key is public, secret, constant, or freshly generated in each run. (2) Message structure includes key pairing, encryption, signature, and hash schemes. (3) Message order verifies the synchronization and involvement of agents.

To build a model in Scyther, we use *roles* and *events* to send and receive messages between two agents. We have also used *claims* that refers to the intended security properties. We verified two properties, secrecy and authentication. Secrecy analyzes, whether either of the involved agents, communicates to a trusted party on a dubious channel. Authentication addresses the aliveness of agents, synchronization, the commitment of the protocol and, message agreement between two parties [48]. Figure 11 shows the verification results of the proposed scheme. We conclude that using quantum scheme in the SPHINCS-256 algorithm makes our proposed algorithm resistant to classical and quantum threats.

```
Sagarikas-MacBook-Air:scyther-mac-v1.1.3_2 sg$ Scyther/Scyther-mac --dot-output
--output=ns3-attacks.dot test.spdl
claim   test,A  Secret_a1       ni      Ok      [no attack within bounds]
claim   test,A  Secret_a11      qk      Ok      [no attack within bounds]
claim   test,A  Secret_a111     sk(A)   Ok      [proof of correctness]
claim   test,A  Weakagree_i4    -       Ok      [no attack within bounds]
claim   test,A  Commit_i5       (B,ni,nr)       Ok      [no attack within bound
]
claim   test,A  Niagree_i6      -       Ok      [no attack within bounds]
claim   test,A  Nisynch_i7      -       Ok      [no attack within bounds]
claim   test,B  Secret_b1       ni      Ok      [no attack within bounds]
claim   test,B  Secret_b2       ni      Ok      [no attack within bounds]
claim   test,B  Secret_b11      qk      Ok      [no attack within bounds]
claim   test,B  Secret_b111     sk(B)   Ok      [proof of correctness]
claim   test,B  Alive_b3        -       Ok      [no attack within bounds]
claim   test,B  Weakagree_b4    -       Ok      [no attack within bounds]
claim   test,B  Commit_b5       (A,nr,ni)       Ok      [no attack within bound
]
claim   test,B  Niagree_b6      -       Ok      [no attack within bounds]
claim   test,B  Nisynch_b7      -       Ok      [no attack within bounds]
```

**Figure 11.** The encryption and signature phase verified by Scyther tool.

## 6. Results

We have implemented our proposed scheme in Python 3.6. We used the Quantum Information Toolkit [53] to simulate the instances based on quantum physics. Furthermore, to validate our hypothesis, we have generated results and performed a statistical analysis of measured variables. Table 3 provides a list of NIST tests executed on the algorithms used in the current and our proposed scheme. We present the results of our algorithm in the following sections.

**Table 3.** NIST tests on algorithms.

| **NIST Tests for Randomness** | | |
|---|---|---|
| RSA and ECDSA (AGA-12) | B92 (in our proposed algorithm) | QRNG and Chacha-12 SPHINCS-256 (in our proposed algorithm) |
| Frequency Test (Monobit) | Frequency Test (Monobit) | Frequency Test (Monobit) |
| Frequency Test within a Block | Frequency Test within a Block | Frequency Test within a Block |
| Run Test | Run Test | Run Test |
| Longest Run of Ones in a Block | Longest Run of Ones in a Block | Longest Run of Ones in a Block |
| Discrete Fourier Transform (Spectral) Test | Discrete Fourier Transform (Spectral) Test | Binary Matrix Rank Test |
| Serial Test 1 and 2 | Serial Test 1 and 2 | Discrete Fourier Transform (Spectral) Test |
| Approximate Entropy Test | Approximate Entropy Test | Non-Overlapping Template Matching Test |
| Cumulative Sums Forward Test | Cumulative Sums Forward Test | Overlapping Template Matching Test |
| Cumulative Sums Reverse Test | Cumulative Sums Reverse Test | Maurer's Universal Statistical test |
| Random Excursions Test | | Linear Complexity Test |
| Random Excursions Variant Test | | Serial Test 1 and 2 |
| | | Approximate Entropy Test |
| | | Cumulative Sums (Forward) Test |
| | | Cumulative Sums (Reverse) Test |
| | | Random Excursions Test |
| | | Random Excursions Variant Test |

### *6.1. Results Obtained in the Key Generation Phase of the Proposed Scheme*

We have implemented our proposed framework in Python using the Quantum Information toolbox. A practical quantum channel consists of noise from the channel imper-

fections. To simulate such a channel, we implemented the logic of the binary symmetric channel. It sends and receives a message in binary with error probability $p$ [54].

During key generation, we implemented and observed two scenarios, with and without Eve. Figures 12 and 13 show that whenever Eve is present, 70% of the time, QBER $>$ 25. In the remaining 30% of the cases, the QBER was around 20%. In Figure 13, the green bar represents QBER $<$ 25% and, the blue bar represents QBER $>$ 25%. Figure 14 exhibits that out of 10 simulations, the mean execution time of the B92 protocol is 0.0198 s.

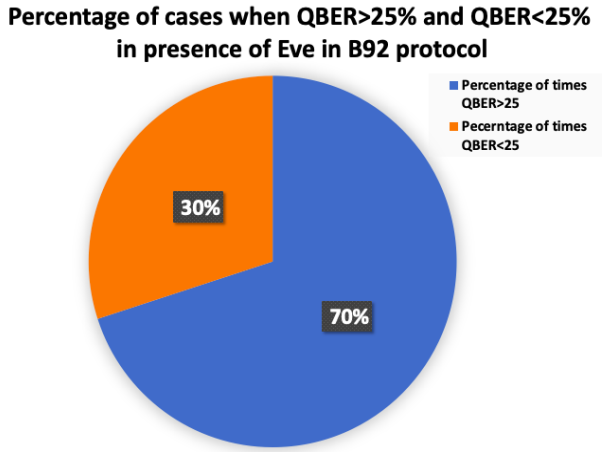


**Figure 12.** Percentage of cases when QBER is greater and less than 25% in the presence of an Eve in B92 protocol.

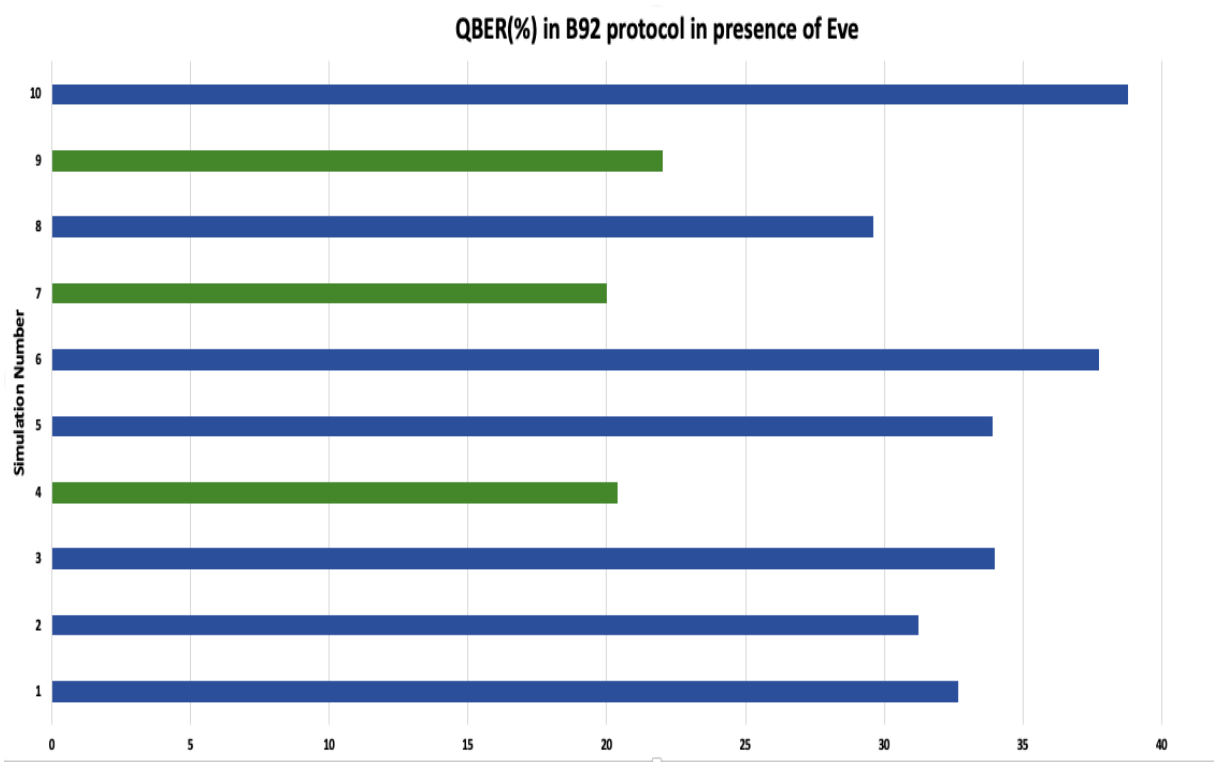


**Figure 13.** QBER in the presence of an Eve in B92 protocol over 10 simulations.

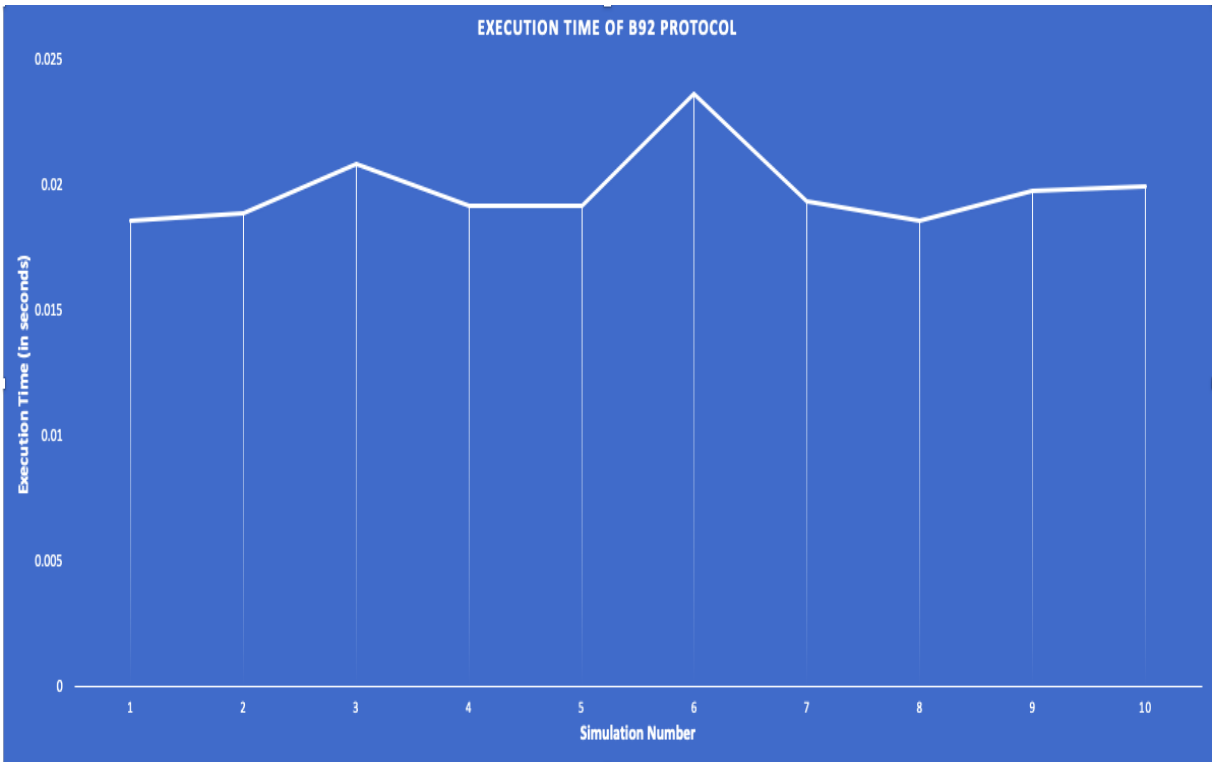


**Figure 14.** Execution Time of B92 protocol.

To perform a comparative analysis between two scenarios, Eve and without Eve, we used the following variables.

- Sifted key size
- QBER
- Incorrect basis count
- Final Key size

Table 4 depicts that only QBER gets affected by the presence of Eve. Since QBER is used to discard the keys, if QBER>25, the other variables, mainly the final key and sifted key, do not vary much. Furthermore, the incorrect basis count does not show much difference with or without Eve.

**Table 4.** Comparative Analysis of B92 in presence of Eve vs B92 in absence of Eve.

| Variables Measured | Simulation in Presence of Eve | | | | | Simulation in Absence of Eve | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| Final Key (bits) | 63 | 63 | 63 | 63 | 61 | 62 | 62 | 63 | 63 | 60 |
| Incorrect Basis Count (IBC) | 252 | 264 | 259 | 260 | 259 | 256 | 260 | 262 | 248 | 268 |
| QBER (%) | 21.5 | 24.4 | 20 | 22 | 24 | 9.83 | 8 | 4 | 9.61 | 12.5 |
| Sifted Key (bits) | 260 | 248 | 253 | 252 | 253 | 256 | 252 | 250 | 264 | 244 |

There are 15 tests for randomness in the NIST statistical test suite. All tests are not suited or required for all random number generators as it depends on various factors, mainly, sample size and algorithms used [55,56]. We are using a 512-bit key size for raw key and 62-bit key size for the final key. We followed Doganaksoy et al.'s paper [56] to use the appropriate statistical tests on both raw key and final key. We generated ten final and raw keys for testing. All ten keys passed the tests. Thus, we conclude that both the raw key and final key are random in each simulation. Tables 5 and 6 provide the *p*-values of

the statistical tests on raw key and final key, respectively. Since the *p*-values are greater than 0.01, the NIST tool concludes the sequences to be random.

**Table 5.** *p*-values of appropriate NIST statistical tests on Randomness applied to Raw Key (512 bit) of B92. Conclusion: Random.

| *p*-Values of Raw Keys | Frequency Test | Frequency Test within a Block | Run Test | Longest Run of Ones in a Block | Discrete Fourier Transform (Spectral) Test | Serial Test 1 | Serial Test 2 | Approx-Imate Entropy Test | Cumulative Sums (Forward) Test | Cummu-Lative Sums (Reverse) Test |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0.92956 | 0.65024 | 0.42611 | 0.13129 | 0.62649 | 0.99402 | 0.99932 | 1 | 0.89202 | 0.81876 |
| **2** | 0.11161 | 0.12026 | 0.54970 | 0.29407 | 0.93535 | 0.49896 | 0.07918 | 1 | 0.22321 | 0.18615 |
| **3** | 0.79088 | 0.44540 | 0.53400 | 0.02329 | 0.25614 | 0.15865 | 0.14452 | 1 | 0.49993 | 0.73751 |
| **4** | 0.47950 | 0.32088 | 0.67429 | 0.26711 | 0.46539 | 0.49896 | 0.23917 | 1 | 0.69601 | 0.36965 |
| **5** | 0.42632 | 0.94899 | 0.36125 | 0.89453 | 0.62649 | 0.49896 | 0.36062 | 1 | 0.57476 | 0.53660 |
| **6** | 0.72367 | 0.54742 | 0.33355 | 0.20087 | 0.93535 | 0.49896 | 0.36062 | 1 | 0.61422 | 0.92314 |
| **7** | 0.929568 | 0.66149 | 0.85941 | 0.37619 | 0.62649 | 0.69077 | 0.85568 | 1 | 0.85688 | 0.77868 |
| **8** | 0.536101 | 0.09507 | 0.98649 | 0.79974 | 0.93535 | 0.69077 | 0.36062 | 1 | 0.89202 | 0.36965 |
| **9** | 0.376759 | 0.75873 | 0.17306 | 0.73362 | 0.62649 | 0.06722 | 0.63695 | 1 | 0.46486 | 0.65476 |
| **10** | 0.790882 | 0.19682 | 0.01713 | 0.25351 | 0.93535 | 0.49896 | 0.49853 | 1 | 0.77868 | 0.73751 |

**Table 6.** *p*-values of appropriate NIST statistical Tests on Randomness applied to Final Key (mean size = 62 bit) of B92. Conclusion: Random.

| *p*-Values of Final Keys | Frequency Test | Frequency Test within a Block | Run Test | Discrete Fourier Transform (Spectral) Test | Serial Test 1 | Serial Test 2 | Approximate Entropy Test | Cumulative Sums (Forward) Test | Cumulative Sums (Reverse) Test |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 0.44605 | 0.44605 | 0.34151 | 0.59996 | 0.49896 | 0.49853 | 1 | 0.25500 | 0.84788 |
| **2** | 0.52873 | 0.52873 | 0.49381 | 0.93090 | 0.49896 | 0.49853 | 1 | 0.85301 | 0.85301 |
| **3** | 0.52873 | 0.52873 | 0.06409 | 0.02604 | 0.49896 | 0.49853 | 1 | 0.41518 | 0.85301 |
| **4** | 0.20408 | 0.20408 | 0.06066 | 0.52153 | 0.49896 | 0.49853 | 1 | 0.19747 | 0.15080 |
| **5** | 0.10145 | 0.10145 | 0.03904 | 0.28487 | 0.49896 | 0.49853 | 1 | 0.11756 | 0.15551 |
| **6** | 0.44605 | 0.44605 | 0.55704 | 0.52153 | 0.49896 | 0.49853 | 1 | 0.50516 | 0.50516 |
| **7** | 0.44605 | 0.44605 | 0.55704 | 0.52153 | 0.49896 | 0.49853 | 1 | 0.50516 | 0.50516 |
| **8** | 0.52873 | 0.52873 | 0.02740 | 0.28487 | 0.49896 | 0.99813 | 1 | 0.94315 | 0.51267 |
| **9** | 0.20408 | 0.20408 | 0.04177 | 0.04177 | 0.49896 | 0.49853 | 1 | 0.32478 | 0.32478 |
| **10** | 0.61145 | 0.61145 | 0.19079 | 0.59996 | 0.49896 | 0.49853 | 1 | 0.73271 | 0.94027 |

### *6.2. Comparative Analysis between SPHINCS-QRNG and Current Algorithms Used in AGA-12*

To address the first set of research questions, we implemented and performed a comparative analysis between the Digital signature algorithm used, RSA and ECDSA, in AGA-12 vs. Quantum Key Distribution protocol and SPHINCS-256 with the QRNG algorithm in our proposed scheme. We performed five simulations for RSA and ECDSA and generated five key pairs for each algorithm. We tested the keys by performing the appropriate NIST statistical tests for randomness. Figure 15 shows the observations that they do not satisfy 100% of them.

The 192-bit private key and 384-bit public key of ECDSA passed 96% of the NIST tests over five simulations. RSA private key size with 2048 bit passed 90% of the NIST tests over five simulations, and RSA public passed 98% of them. In contrast, the raw key and final key in QKD passed all the tests (100%) over ten simulations. Moreover, SPHINCS-256, with QRNG key pairs, passed 98% of the tests over five simulations.

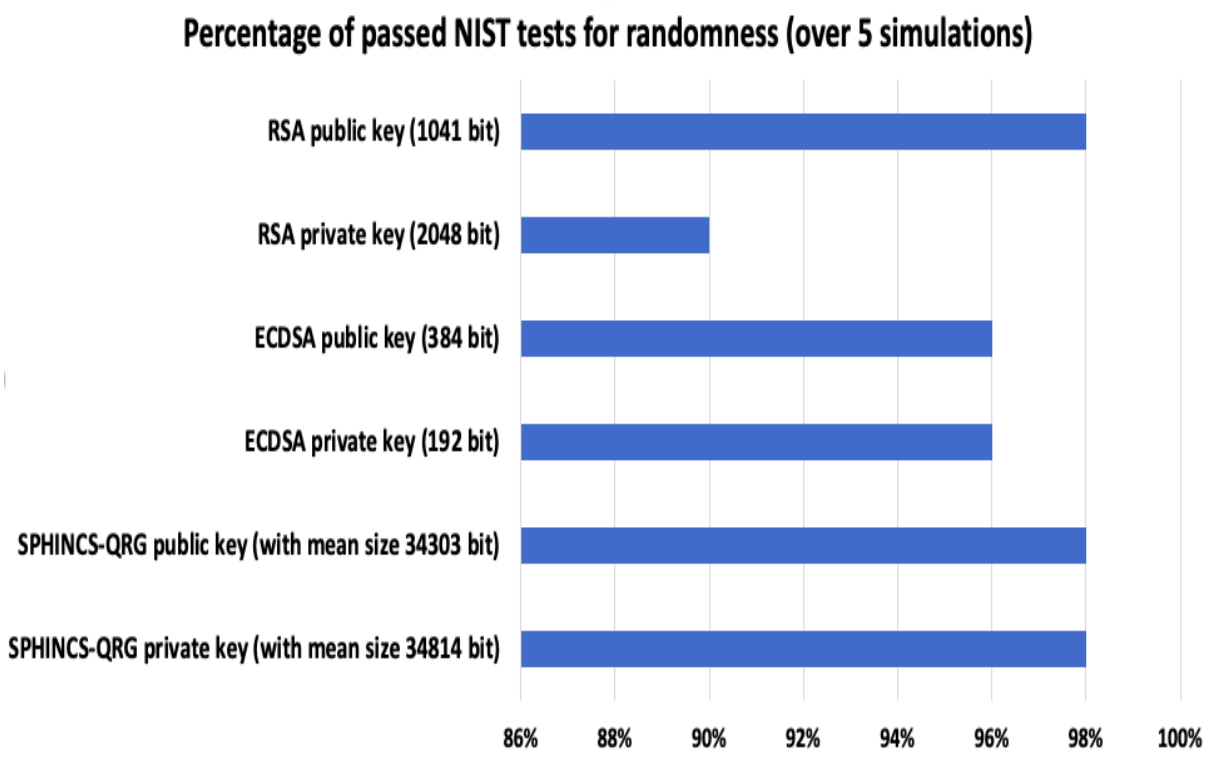


**Figure 15.** Percentage of passed tests, by RSA, ECDSA and SPHINCS-QRNG key pairs, based on NIST statistical test suite on randomness.

### *6.3. Results Obtained in the Signature Generation Phase*

To address the second set of research questions, we implemented both algorithms: SPHINCS-256 using Chacha12 PRNG and SPHINCS-256 using QRNG to generate HORST secret key. We named the two models *SPHINCS-Chacha12* and *SPHINCS-QRNG*. For comparative analysis, we considered the root of the SPHINCS tree as the public key. And, we considered the private key of the SPHINCS tree at the topmost (d-1) layer. The signature size obtained in both models is 27873 bytes. We generated five random numbers from each algorithm and fed each of them to the NIST statistical tool. We compared both the algorithms by measuring execution time, testing randomness of the generators used in each algorithm, and based on Datcu et al.'s [57] research on testing Chacha12 PRNG.

#### 6.3.1. Comparative Analysis of SPHINCS-QRNG and SPHINCS-Chacha12 Based on Execution Time

Figure 16 shows that the execution time of SPHINCS QRNG is more than that of the existing SPHINCS algorithm with PRNG. The mean execution time of SPHINCS-QRNG public key is 160 µs, and of the private key is 238.89 µs. Whereas, in SPHINCS-Chacha12 algorithm, the average time to generate the public key 112.15 µs and the private key is 110.12 µs.

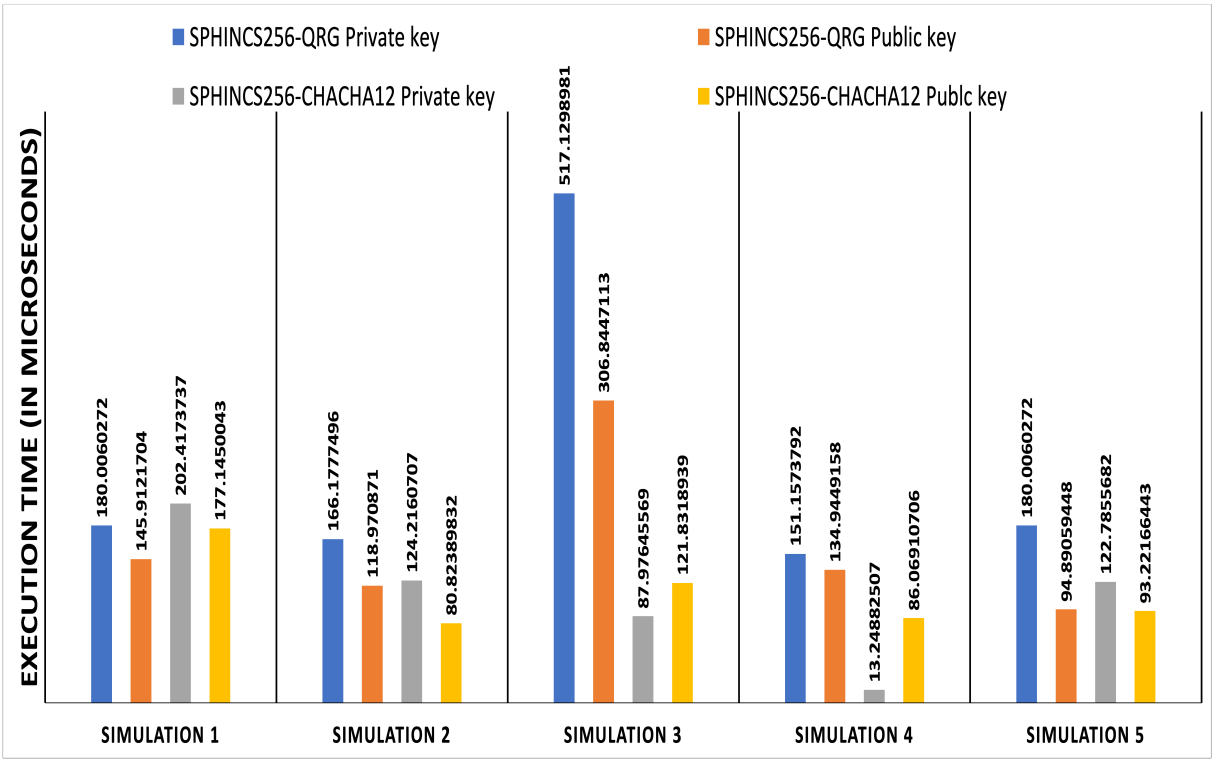


**Figure 16.** Execution Time for SPHINCS-Chacha12 vs. SPHINCS-QRNG

Moreover, we also performed a comparative analysis based on theoretical performance. Since SPHINCS is a *h/d-ary* certification tree, Daniel et al. [13] provided a rough theoretical run time values based on the count of pseudo random number functions (PRFs), PRNG and, hashes. The total height of hyper tree is $h$ with $d$ multiple layers of trees. It takes $d2^{h/d}$ OTS key generations, $d2^{h/d}$ + 2 PRF calls, $d2^{h/d}$ + 1 PRNG calls and $2t + d((\ l\ (w + 1))$ $2^{h/d} - 1)$ hashes. HORST uses parameter $t$, such that $t = 2^{\tau}$. The height of the HORST tree is $\tau = \log t$. WOTS uses a signature size and runtime tradeoff parameter $w$ such that $w \in \mathbb{N}$. However, the time complexity of RSA is O($n^2$) and that of ECC is O($n^3$)) [58]. Further, all pseudo-random number generator requires O(n) bit operations. In contrast, QRNG based on Hadamard transformation can be computed in O(nlogn) operations in classical hardware and, in O(1) in quantum hardware [59,60]. Therefore, we conclude that theoretically, the computational complexity of our proposed algorithm is higher than that of existing SPHINCS based on PRNG and AGA-12.

#### 6.3.2. Comparative Analysis of SPHINCS-QRNG and SPHINCS-Chacha12 Based on Randomness

To test the randomness of the quantum random number generator (QRNG), we used all 15 tests of the NIST statistical test suite and, the QRNG passed all of them in every simulation. Thus, we conclude that QRNG is truly random. Table 7 displays the $p$-values of random number generated by QRNG used in SPHINCS-256. As the $p$-values $\geq 0.01$, the NIST tool concludes the sequence to be random with a confidence of 99%. Figures 17 and 18 display the results of Random Excursions Test and Random Excursions Variant Test, respectively for *SPHINCS-QRNG*. The Random Excursions test executes sub-tests on each of the following states; $-4$, $-3$, $-2$, $-1$, +1, +2, +3 and +4, to check the frequency of visits to a cumulative sums state within a cycle of a random walk matches with that one would expect for random sequence [55]. For a certain state, if the $p$-value $\geq 0.01$, the sequence is random. For example, in Figure 17, the $p$-value of QRN 1 with state +1 is approximately 0.8492. Thus, the sequence at state +1 is random.

**Table 7.** *p*-values of appropriate NIST Statistical Tests for Random and PRNG applied on Quantum Random Number Generator(QRNG) used in SPHINCS-256.

| *p*-Values of QRNG Used in SPHINCS-256 | QRN 1 | QRN 2 | QRN 3 | QRN 4 | QRN 5 |
|---|---|---|---|---|---|
| Frequency Test (Monobit) | 0.852444761 | 0.011735483 | 0.149302374 | 0.406538784 | 0.649828827 |
| Frequency Test within a Block | 0.161769992 | 0.733286939 | 0.575099934 | 0.665007898 | 0.518616736 |
| Run Test | 0.217961992 | 0.827155028 | 0.32712429 | 0.732869216 | 0.924945411 |
| Longest Run of Ones in a Block | 0.513396784 | 0.969974739 | 0.637115311 | 0.248794857 | 0.913444629 |
| Binary Matrix Rank Test | 0.262333935 | 0.822162028 | 0.862431288 | 0.111413103 | 0.071873969 |
| Discrete Fourier Transform (Spectral) Test | 0.142033423 | 0.011616891 | 0.157596656 | 0.776045999 | 0.613759295 |
| Non-Overlapping Template Matching Test | 0.63600041 | 0.504047187 | 0.415637608 | 0.169519974 | 0.262129471 |
| Overlapping Template Matching Test | 0.200756533 | 0.505736739 | 0.353171054 | 0.591057811 | 0.508073676 |
| Maurer's Universal Statistical test | 0.124605474 | 0.243079644 | 0.01674513 | 0.73273627 | 0.889955415 |
| Linear Complexity Test | 0.8575379 | 0.547314553 | 0.126281648 | 0.190952376 | 0.371943788 |
| Serial test 1 | 0.667804599 | 0.9238906 | 0.773287042 | 0.949470145 | 0.285436568 |
| Serial test 2 | 0.691057349 | 0.953760341 | 0.7814567 | 0.920487151 | 0.576467535 |
| Approximate Entropy Test | 0.569869981 | 0.89057965 | 0.092335008 | 0.350697922 | 0.18177552 |
| Cumulative Sums (Forward) Test | 0.800359989 | 0.013619809 | 0.215229911 | 0.508140027 | 0.69796035 |
| Cumulative Sums (Reverse) Test | 0.943118012 | 0.018863213 | 0.268772338 | 0.771928763 | 0.764505358 |

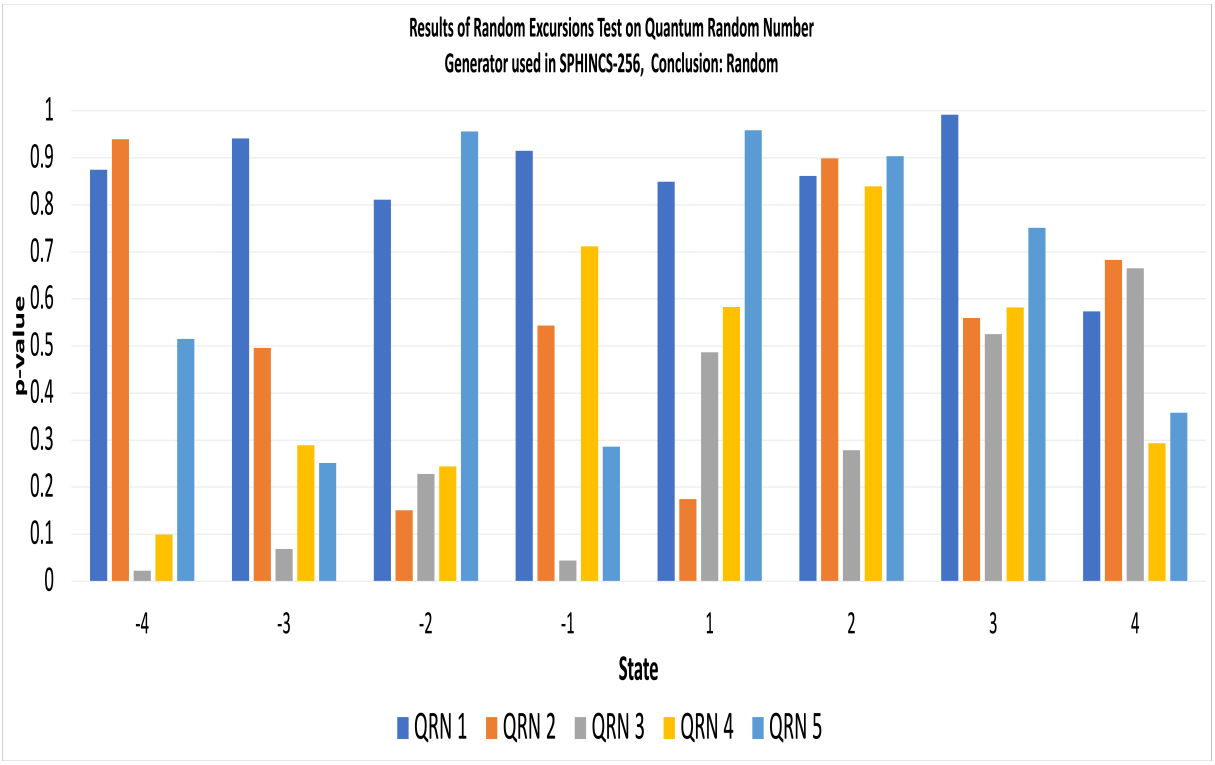


**Figure 17.** Results of Random Excursions Test on Quantum Random Number Generator(QRNG) used in SPHINCS-256.

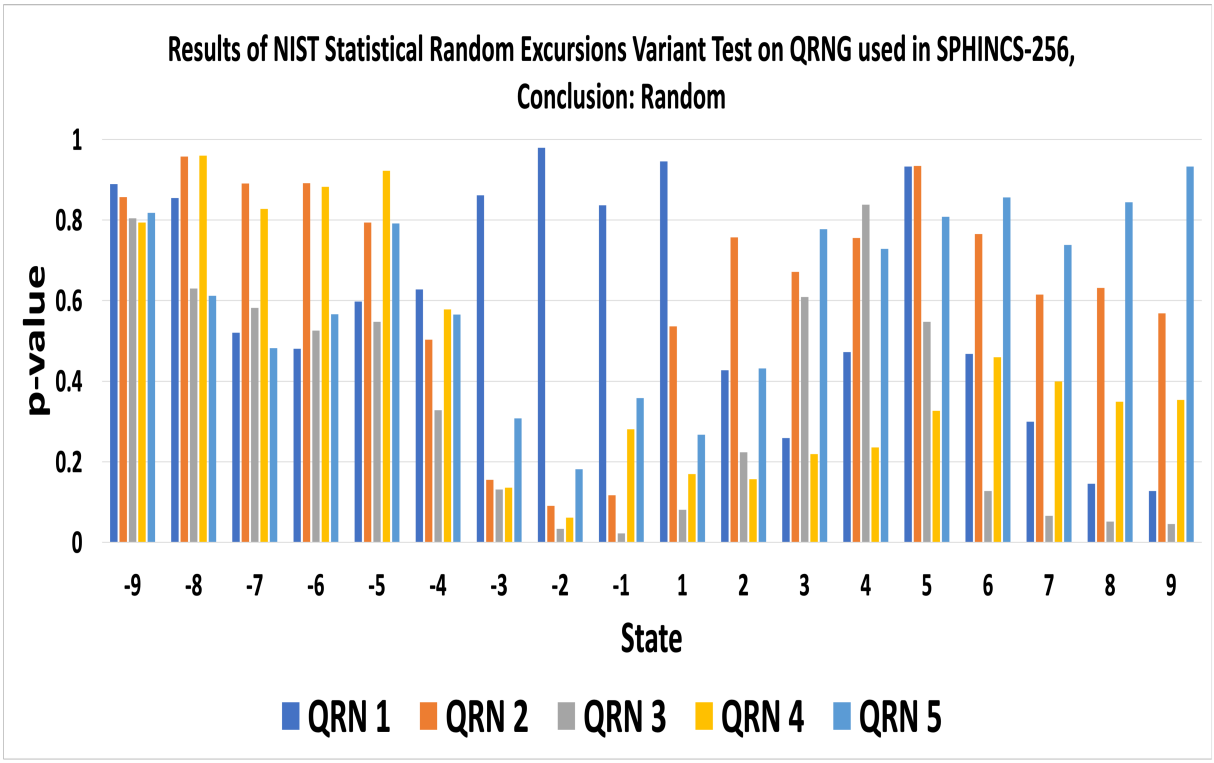


**Figure 18.** Results of Random Excursions Variant Test on Quantum Random Number Generator(QRNG) used in SPHINCS-256.

The Random Excursions Variant test verifies whether the number of visits to a particular state in a cumulative sum random walk deviates from the expected number of visits in the random walk. It considers 18 states consisting of $\{-9, -8, \ldots, +8, +9\}$ and, if the $p$-value of a particular state is greater than or equal to 0.01, that means the sequence is random. However, when we ran all 15 tests on a random number generated by *SPHINCS-Chacha12*, it does not pass 100% of the NIST statistical tests. The pseudo-random number, PRN 5, generated by Chacha-12, failed the Maurer's Universal Statistical test. Also, PRN 1 and PRN 5 does not pass the Random Excursions Test and Random Excursions Variant Test with all states. Table 8, Figures 19 and 20 exhibit the results of the randomness of Chacha-12 PRNG tested by NIST statistical test suite.

We performed 15 statistical analysis on five random numbers for each algorithm. Thus, we performed $15 \times 5 = 75$ tests. Out of 75 tests, Chacha-12 passed 70 tests and, QRNG passed all of them. Figure 21 exhibits a graph that shows Chacha-12 PRNG scored 93.3%, and QRNG scored 100% for successfully passing the tests. The Chacha-12 algorithm is feasible for resource-constrained devices [61]. However, Datcu et al. [57], showed that secure PRNG Chacha does not pass all the statistical tests of Monte-Carlo analysis. Moreover, ID Quantique has developed a QRNG chip for critical infrastructure involving the Internet of Things (IoT) and other resource constrained devices [19]. Table 9 provides a synthesis of results obtained from comparative analysis of the proposed algorithm, RSA, ECDSA and SPHINCS-256.

**Table 8.** *p*-values of appropriate NIST Statistical Tests for Random and PRNG applied on Chacha-12 used in SPHINCS-256.

| ***p*-Values of PRNG Used in SPHINCS-256** | **PRN 1** | **PRN 2** | **PRN 3** | **PRN 4** | **PRN 5** |
|---|---|---|---|---|---|
| Frequency Test (Monobit) | 0.756560956 | 0.61285665 | 1 | 0.87288107 | 0.76723008 |
| Frequency Test within a Block | 0.466408066 | 0.2519857 | 0.66537382 | 0.05620039 | 0.27892162 |
| Run Test | 0.046601502 | 0.41321088 | 0.95693516 | 0.93466707 | 0.13415598 |
| Longest Run of Ones in a Block | 0.39001338 | 0.57336093 | 0.31763597 | 0.60612676 | 0.43188342 |
| Binary Matrix Rank Test | 0.234372205 | 0.70947779 | 0.89969577 | 0.38005323 | 0.77390385 |
| Discrete Fourier Transform (Spectral) Test | 0.520636833 | 0.87603089 | 0.52063683 | 0.21202315 | 0.4798148 |
| Non-Overlapping Template Matching Test | 0.17548433 | 0.67642991 | 0.08873813 | 0.25081909 | 0.0158475 |
| Overlapping Template Matching Test | 0.341609907 | 0.93826949 | 0.03571624 | 0.88583617 | 0.19903682 |
| Maurer's Universal Statistical test | 0.251209205 | 0.22365261 | 0.37912508 | 0.21287032 | 0.00739755 |
| Linear Complexity Test | 0.636767031 | 0.43256403 | 0.95718573 | 0.68049537 | 0.26611467 |
| Serial test 1 | 0.266168112 | 0.04303137 | 0.85174723 | 0.77514063 | 0.06351778 |
| Serial test 2 | 0.111246791 | 0.36341311 | 0.71254001 | 0.62943052 | 0.1555089 |
| Approximate Entropy Test | 0.991465303 | 0.50945017 | 0.72260759 | 0.55896218 | 0.70516401 |
| Cumulative Sums (Forward) Test | 0.550993298 | 0.64760991 | 0.550134 | 0.6264806 | 0.39388365 |
| Cumulative Sums (Reverse) Test | 0.834146968 | 0.85546589 | 0.550134 | 0.77562923 | 0.22547471 |

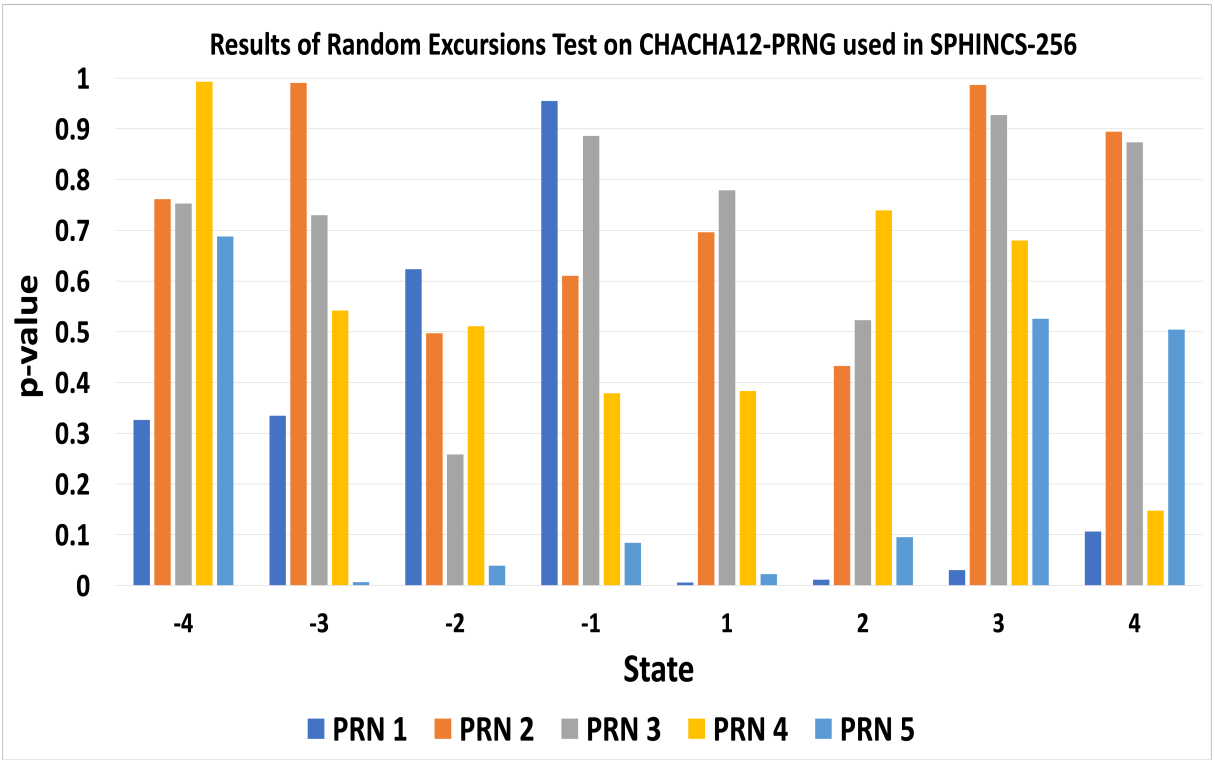


**Figure 19.** Results of Random Excursions Test on Chacha-12 PRNG used in SPHINCS-256.

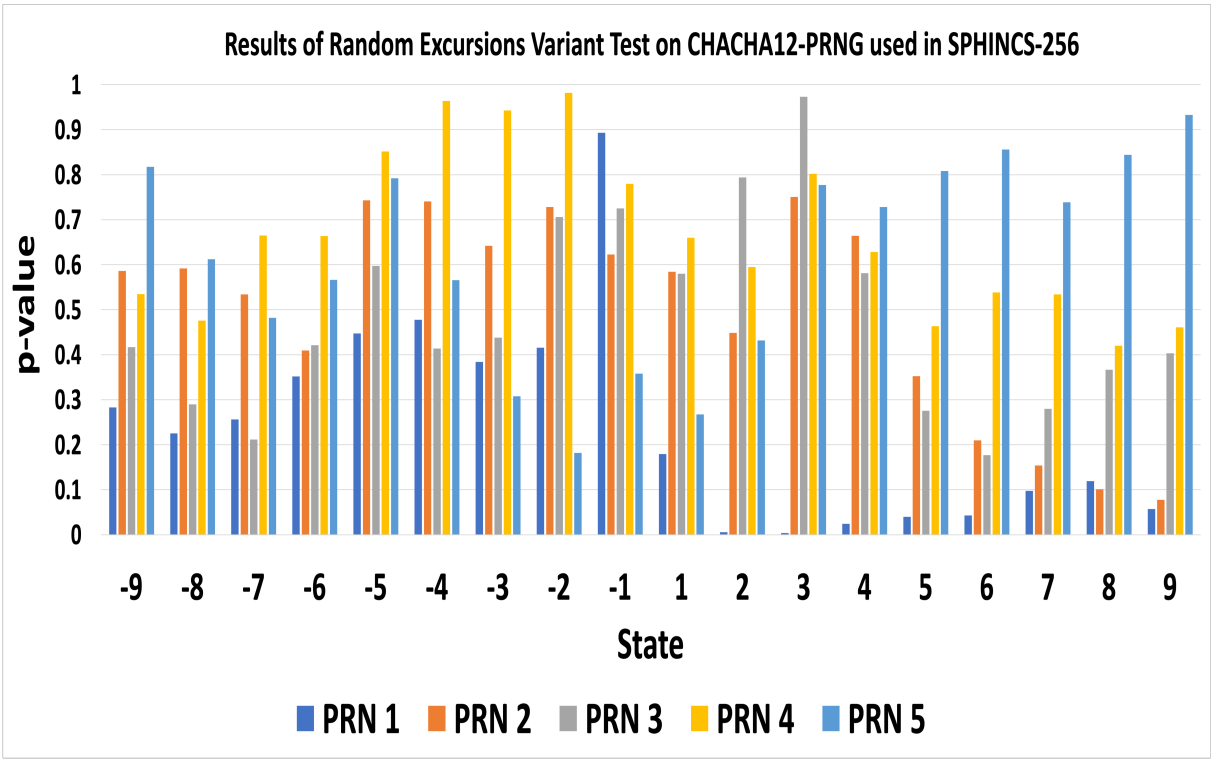


**Figure 20.** Results of Random Excursions Variant Test on Chacha-12 PRNG used in SPHINCS-256.

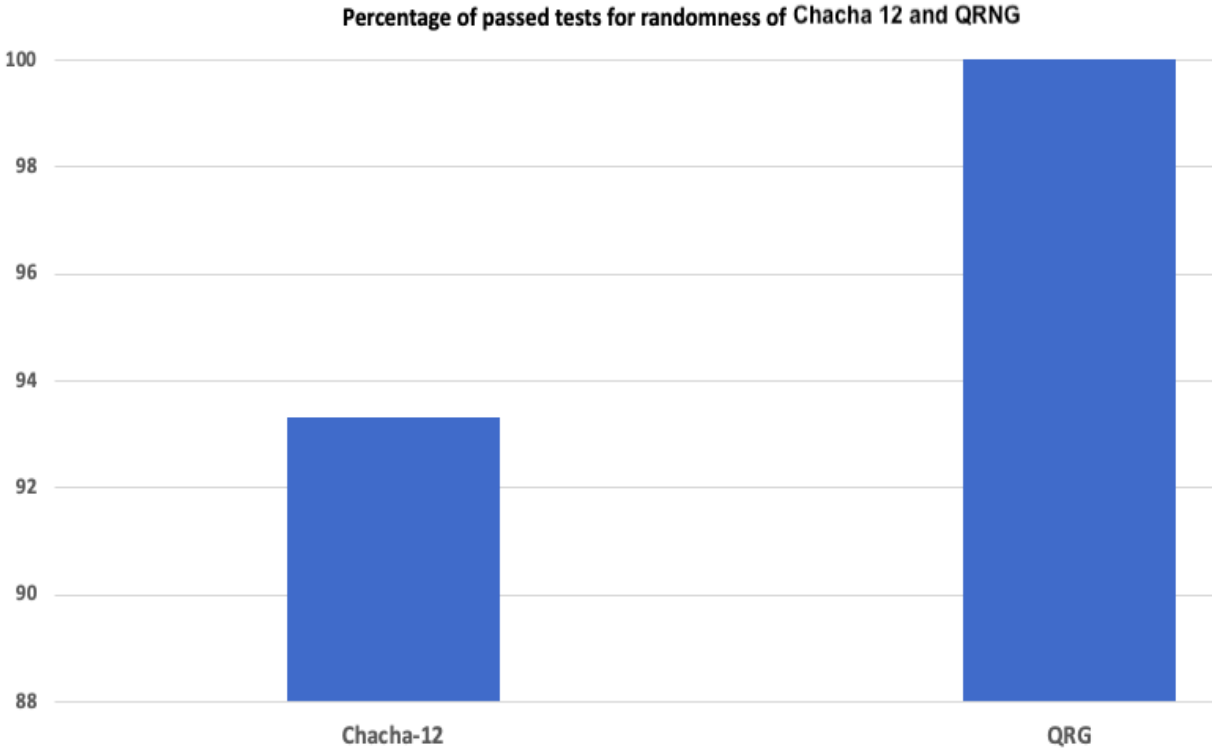


**Figure 21.** Percentage of passed tests for randomness scored by Chacha-12 and QRNG.

**Table 9.** Synthesis of comparative analysis of the proposed algorithm, RSA, ECDSA and SPHINCS-256.

| Algorithms | Public Key Size (bits) | Private Key Size (bits) | Raw key Size (bits) | Final Key Size (bits) | QBER (mean) | Percentage of Passed NIST Randomness Tests | Execution Time/ Time Complexity |
|---|---|---|---|---|---|---|---|
| RSA | 1041 | 2048 | N/A | N/A | No intrusion detection | Private key: 92% Public key: 98% | O ($n^2$) |
| ECDSA | 384 | 192 | N/A | N/A | No intrusion detection | 96% | O ($n^3$) |
| The proposed algorithm | 34303 | 34814 | 512 | 62 | 15.584% Detects eavesdropping. | Both keys: 98% QRNG: 100% | $d2^{h/d}$ OTS key generations $d2^{h/d}$ + 2 PRF calls. $d2^{h/d}$ + 1 PRNG calls and $2t + d((l(w+1))\,2^{h/d} - )1$ hashes QRNG : *O(nlogn)* in classical hardware. *O(1)* in quantum hardware. Public Key Generation: 160 µs Private Key Generation: 238.89 µs |
| SPHINCS-256 | 34303 | 34814 | N/A | N/A | No intrusion detection | Chacha-12: 93.3% | $d2^{h/d}$ OTS key generations $2d^{h/d}$ + 2 PRF calls. $d2^{h/d}$ + 1 PRNG calls and $2t + d((l(w+1))\,2^{h/d} -)1$ hashes PRNG : *O(n)* bit operations. Public Key Generation: 112.5 µs Private Key Generation: 110.12 µs |

## 7. Conclusions

We have proposed a collision and preimage resistant framework for ensuring SCADA system security. SPHINCS-256, a post-quantum algorithm, provides $2^{128}$ security against a quantum threat. However, researchers have increased the efficiency and speed of quantum algorithm based on Grover's algorithm, which reduces the post-quantum security from $2^{128}$ to $2^{119.6}$ in a quantum setting. Therefore, we proposed to use B92, a quantum key distribution protocol, to obtain the cipher and a quantum random number generator to generate a truly random number for the HORST secret key used in SPHINCS-256. We have formally verified our proposed scheme by using PRISM and Scyther. We conclude that with the number of qubits, the probability of detecting intruder Eve increases exponentially, decreasing the likelihood of information leakage. Furthermore, B92 is more likely to detect Eve's presence in case of a Random-Substitute attack than an Intercept-Resend attack. We validated our hypothesis by simulating our proposed scheme and performing statistical analysis. We analyzed the randomness of RSA, ECDSA keys used in AGA-12 against the randomness of the quantum key used in our proposed algorithm. We observe that keys generated by QKD satisfy 100% of the NIST randomness tests, unlike RSA and ECDSA keys. RSA private key passed 90%, and ECDSA keys passed 96% of the tests. We also performed a comparative analysis between two algorithms, SPHINCS-256 with Chacha-12 and SPHINCS-256 with QRNG. We observe that the Quantum Random Number

Generator passes all the statistical tests for randomness, unlike Chacha-12 PRNG. However, in computational hardware, the computation cost of our proposed SPHINCS-QRNG is higher than that of AGA-12 and the existing SPHINCS-PRNG algorithm. Thus, we conclude that there is a trade-off between security and computation cost. Our proposed framework, using true random numbers based on uncertainty and quantum superposition principles, provides more resistance than AGA-12 and SPHINCS-256 against quantum and classical threats. Moreover, as part of future work, we will propose a quantum-resistant encryption algorithm and compare it with various encryption algorithms to obtain a more effective cipher for the SCADA security framework.

**Author Contributions:** S.G.: conceptualization, formal analysis, methodology, writing—first draft; M.Z.: project administration, writing—review and editing; G.S.: project administration; S.S.: funding acquisition, supervision, writing—review and editing. All authors have read and agreed to the published version of the manuscript.

**Funding:** The authors gratefully acknowledge the support in part by the Natural Sciences and Engineering Research Council (NSERC) and industry partners Cistel Technology Inc., through a Collaborative Research Grant.

**Institutional Review Board Statement:** "Not applicable" for studies not involving humans or animals.

**Informed Consent Statement:** "Not applicable" for studies not involving humans or animals.

**Data Availability Statement:** Data is available within the manuscript.

**Acknowledgments:** Thanks to Rohit Joshi of Cistel Technologies for valuable feedback.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Ghosh, S.; Sampalli, S. A survey of security in SCADA networks: Current issues and future challenges. *IEEE Access* **2019**, *7*, 135812–135831. [CrossRef]
2. Kang, D.J.; Lee, J.J.; Kim, S.J.; Park, J.H. Analysis on cyber threats to SCADA systems. In Proceedings of the 2009 Transmission & Distribution Conference & Exposition: Asia and Pacific; Seoul, Korea (South), IEEE: 2009; pp. 1–4.
3. Lomonaco, S. Shor's quantum factoring algorithm. In Proceedings of Symposia in Applied Mathematics, 2002; American Mathematical Society, Providence, Rhode Island(USA); Volume 58, pp. 161–180.
4. Grover, L.K. A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, New York, United States, 1996; pp. 212–219.
5. Yanofsky, N.S.; Mannucci, M.A. *Quantum Computing for Computer Scientists*; Cambridge University Press: Cambridge, UK, 2008. [CrossRef]
6. Kaye, P.; Laflamme, R.; Mosca, M. *An Introduction to Quantum Computing*; Oxford University Press: Oxford, UK, 2007.
7. Gidney, C.; Ekerå, M. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *arXiv* **2019**, arXiv:1905.09749.
8. Wilcox, Z. Lessons From The History Of Attacks On Secure Hash Functions. Available online: https://electriccoin.co/blog/lessons-from-the-history-of-attacks-on-secure-hash-functions/ (accessed on 29 August 2020).
9. Zhang, X.; Dong, Z.Y.; Wang, Z.; Xiao, C.; Luo, F. Quantum Cryptography Based Cyber-Physical Security Technology for Smart Grids. In Proceedings of the 10th International Conference on Advances in Power System Control, Operation & Management (APSCOM 2015), Hong Kong, China, 8–12 November 2015; pp. 1–6. [CrossRef]
10. Padamvathi, V.; Vardhan, B.V.; Krishna, A. Quantum cryptography and quantum key distribution protocols: A survey. In Proceedings of the 2016 IEEE 6th International Conference on Advanced Computing (IACC); Bhimavaram, India, 27–28 February 2016; pp. 556–562.
11. Nurhadi, A.I.; Syambas, N.R. Quantum key distribution (QKD) protocols: A survey. In Proceedings of the 2018 4th International Conference on Wireless and Telematics (ICWT), Bali, Indonesia, 12–13 July 2018; pp. 1–5.
12. Chen, L.; Chen, L.; Jordan, S.; Liu, Y.K.; Moody, D.; Peralta, R.; Perlner, R.; Smith-Tone, D. *Report on Post-Quantum Cryptography*; US Department of Commerce, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2016; Volume 12.
13. Bernstein, D.J.; Hopwood, D.; Hülsing, A.; Lange, T.; Niederhagen, R.; Papachristodoulou, L.; Schneider, M.; Schwabe, P.; Wilcox-O'Hearn, Z. SPHINCS: Practical stateless hash-based signatures. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, 26–30 April 2015; 2015; pp. 368–397.
14. Sharma, A.; Mittal, S.K. Attacks on Cryptographic Hash Functions and Advances. *Int. J. Inf. Comput. Sci.* **2018**, *5*, 89–96.
15. Bernstein, D.J. ChaCha, a variant of Salsa20. In *Workshop Record of SASC*; Lausanne, Switzerland, 2008; Volume 8; pp. 3–5.

16. Goll, M.; Gueron, S. Vectorization on ChaCha stream cipher. In Proceedings of the 2014 11th International Conference on Information Technology: New Generations, Las Vegas, NV, USA, 7–9 April 2014; pp. 612–615.
17. Choudhuri, A.R.; Maitra, S. Differential Cryptanalysis of Salsa and ChaCha-An Evaluation with a Hybrid Model. *IACR Cryptol. ePrint Arch.* **2016**, *2016*, 377.
18. Chailloux, A.; Naya-Plasencia, M.; Schrottenloher, A. An efficient quantum collision search algorithm and implications on symmetric cryptography. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*; Springer: Hong Kong, China, 2017; pp. 211–240.
19. Quantis QRNG Chip System-on-Chip for Automotive, Computing, Critical Infrastructure, IoT, Mobile & sEcurity Applications. Available Online: https://www.idquantique.com/random-number-generation/products/quantis-qrng-chip/ (accessed on 30 August 2020).
20. Sibson, P.; Erven, C.; Godfrey, M.; Miki, S.; Yamashita, T.; Fujiwara, M.; Sasaki, M.; Terai, H.; Tanner, M.G.; Natarajan, C.M. Chip-based quantum key distribution. *Nat. Commun.* **2017**, *8*, 1–6. [CrossRef] [PubMed]
21. Parvez, B.; Ali, J.; Ahmed, U.; Farhan, M. Framework for implementation of AGA 12 for secured SCADA operation in Oil and Gas Industry. In Proceedings of the 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 11–13 March 2015; pp. 1281–1284.
22. Amy, M.; Di Matteo, O.; Gheorghiu, V.; Mosca, M.; Parent, A.; Schanck, J. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In Proceedings of the International Conference on Selected Areas in Cryptography; Springer: St. John's, NL, Canada, 2016; pp. 317–337.
23. Mavroeidis, V.; Vishi, K.; Zych, M.D.; Jøsang, A. The impact of quantum computing on present cryptography. *arXiv* **2018**, arXiv:1804.00200.
24. Routray, S.K.; Jha, M.K.; Sharma, L.; Nyamangoudar, R.; Javali, A.; Sarkar, S. Quantum cryptography for iot: Aperspective. In *Proceedings of the 2017 International Conference on IoT and Application (ICIOT)*; IEEE: Nagapattinam, India, 2017; pp. 1–4.
25. Wootters, W.K.; Zurek, W.H. A single quantum cannot be cloned. *Nature* **1982**, *299*, 802–803. [CrossRef]
26. Muller, A.; Zbinden, H.; Gisin, N. Quantum cryptography over 23 km in installed under-lake telecom fibre. *EPL Europhys. Lett.* **1996**, *33*, 335. [CrossRef]
27. Bennett, C.H.; Brassard, G. Quantum cryptography: Public key distribution and coin tossing. In Proceedings of the International Conference on Computers, Systems and Signal Processing, Bangalore, India, 9–12 December 1984; pp. 175–179.
28. Bennett, C.H. Quantum cryptography using any two nonorthogonal states. *Phys. Rev. Lett.* **1992**, *68*, 3121. [CrossRef] [PubMed]
29. Diamanti, E.; Lo, H.K.; Qi, B.; Yuan, Z. Practical challenges in quantum key distribution. *npj Quantum Inf.* **2016**, *2*, 1–12. [CrossRef]
30. Brassard, G.; Høyer, P.; Tapp, A. Quantum cryptanalysis of hash and claw-free functions. *ACM Sigact News* **1997**, *28*, 14–19. [CrossRef]
31. Becker, G. *Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis*; Ruhr-University Bochum: Bochum, Germany, 2008.
32. Reyzin, L.; Reyzin, N. Better than BiBa: Short one-time signatures with fast signing and verifying. In *Proceedings of the Australasian Conference on Information Security and Privacy*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 144–153.
33. Aumasson, J.P.; Endignoux, G. Improving stateless hash-based signatures. In *Cryptographers' Track at the RSA Conference*; Springer: Cham, Switzerland, 2018; pp. 219–242.
34. Nuhamara, B.R.H.; Syambas, N.R. An Evaluation of Quantum Key Distribution in QuVis Simulation Software. In *Proceedings of the 2018 4th International Conference on Wireless and Telematics (ICWT)*; IEEE: Bali, Indonesia, 2018; pp. 1–4.
35. Ruj, S.; Roy, B. Key predistribution schemes using codes in wireless sensor networks. In *Proceedings of the International Conference on Information Security and Cryptology*; Springer: Beijing, China,2008; pp. 275–288.
36. Choudhari, S.P.; Chakole, M.B. Reed solomon code for WiMAX network. In *Proceedings of the 2017 International Conference on Communication and Signal Processing (ICCSP)*; IEEE: Chennai, India, 2017; pp. 0176–0179.
37. Riley, M.; Richardson, I. An Introduction to Reed-Solomon Codes: Principles, Architecture and Implementation. Available online: https://www.cs.cmu.edu/~guyb/realworld/reedsolomon/reed$_$solomon$_$codes.html (accessed on 30 August 2020).
38. Upadhyay, Darshana and Sampalli, Srinivas. SCADA (Supervisory Control and Data Acquisition) systems: Vulnerability assessment and security recommendations. *Comput. Secur.* **2020**, *89*, 101666.
39. Stouffer, Keith and Falco, Joseph and Kent, Karen. Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security. 2006. Available online: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf (accessed on 30 August 2020).
40. Scalable Quantum Cryptography Network For Protected Automation Communication. Energy.gov.2017. US Department. Available online: https://www.energy.gov/sites/prod/files/2017/05/f34/Qubitekk$_$QKD$_$FactSheet.pdf.2017 (accessed on 30 August 2020).
41. Bailey, David and Wright, Edwin. *Practical SCADA for Industry*; Elsevier: Amsterdam, The Netherlands, 2003.
42. Aumasson, J.P.; Endignoux, G. Clarifying the subset-resilience problem. *IACR Cryptol. Eprint Arch.* **2017**, *2017*, 909.
43. Bernstein, Daniel J and Hülsing, Andreas and Kölbl, Stefan and Niederhagen, Ruben and Rijneveld, Joost and Schwabe, Peter. The SPHINCS+ signature framework. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 2129–2146.
44. Herrero-Collantes, Miguel and Garcia-Escartin, Juan Carlos. Quantum random number generators. *Rev. Mod. Phys.* **2017**, *89*, 015004. [CrossRef]

45. Stipcevic, Mario. Quantum random number generators and their applications in cryptography. In *Advanced Photon Counting Techniques VI*; International Society for Optics and Photonics: Chennai, India, 2012; Volume 8375, p. 837504.
46. ID Quantique. Quantum Versus Classical Random Number Generator. Available online: https://marketing.idquantique.com/acton/attachment/11868/f-64900ef6-6e7e-4b4c-a9f9-c912a2cfde59/1/-/-/-/-/Classical$%$20RNG$%$20Vs$%$20QRNG$_$White$%$20Paper.pdf (accessed on 30 August 2020).
47. Kwiatkowska, M.; Norman, G.; Parker, D. PRISM: Probabilistic symbolic model checker. In *Proceedings of the International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 200–204.
48. Cremers, C. *Scyther User Manual*; Department of Computer Science, University of Oxford: Oxford, UK, 2014.
49. Papanikolaou, N.K. Techniques for Design and Validation of Quantum Protocols. Ph.D Thesis, Department of Computer Science, University of Warwick, Coventry, UK, 2004.
50. Kuppam, S. Modelling and Analysis of Quantum Key Distribution Protocols, BB84 and B92, in Communicating Quantum Processes (CQP) language and Analysing in PRISM. *arXiv* **2018**, arXiv:1612.03706.
51. Kuppam, A. Modelling BB84, B92 in CQP and Analysing in PRISM. *arXiv* **2016**, arXiv:1612.03706v1.
52. Chatterjee, R.; Joarder, K.; Chatterjee, S.; Sanders, B.C.; Sinha, U. qkdSim: An experimenter's simulation toolkit for QKD with imperfections, and its performance analysis with a demonstration of the B92 protocol using heralded photon. *arXiv* **2019**, arXiv:1912.10061.
53. Bergholm, V.; Biamonte, J.D.; Whitfield, J.D. Quantum Information Toolkit. Available online: http://qit.sourceforge.net (accessed on 30 August 2020).
54. Crépeau, C. Efficient cryptographic protocols based on noisy channels. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 1997; pp. 306–317.
55. Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Barker, E. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*; Technical Report; Booz-Allen and Hamilton inc Mclean va: Tysons Corner, VA, USA, 2001.
56. Doganaksoy, A.; Ege, B.; Koçak, O.; Sulak, F. Statistical Analysis of Reduced Round Compression Functions of SHA-3 Second Round Candidates. *IACR Cryptol. Eprint Arch.* **2010**, *2010*, 611.
57. Datcu, O.; Macovei, C.; Hobincu, R. Chaos Based Cryptographic Pseudo-Random Number Generator Template with Dynamic State Change. *Appl. Sci.* **2020**, *10*, 451. [CrossRef]
58. RSA Cryptography Algorithm. Available online: http://algohub.me/algo/rsa-cryptography-algorithm.html (accessed on 30 August 2020).
59. How Fast Does a Pseudorandom Number Generator Have to be in Order to be Competitive? Available online: https://crypto.stackexchange.com/questions/62736/how-fast-does-a-pseudorandom-number-generator-have-to-be-in-order-to-be-competit (accessed on 30 August 2020).
60. Aung, A.; Ng, B.P.; Rahardja, S. Sequency-ordered complex Hadamard transform: Properties, computational complexity and applications. *IEEE Trans. Signal Process.* **2008**, *56*, 3562–3571. [CrossRef]
61. Philip, M.A. A survey on lightweight ciphers for IoT devices. In *Proceedings of the 2017 International Conference on Technological Advancements in Power and Energy (TAP Energy)*; IEEE: Kollam, India, 2017; pp. 1–4.