Verkle Tree-based Post-Quantum Digital Signature Scheme using Stateless Updatable Vector Commitment

Maksim Iavich[1], Tamari Kuchukhidze[2], and Tetiana Okhrimenko[3]

[1] *Caucasus University, 1 Paata Saakadze str., Tbilisi, 0102, Georgia*
[2] *International Black Sea University, 2 David Agmashenebeli Alley 13th km, Tbilisi, 0131, Georgia*
[3] *National Aviation University, 1 Liubomyra Huzara ave., Kyiv, 03058, Ukraine*

Abstract
In recent years, work on quantum computers has made substantial progress. Many of the current public key cryptosystems can be broken if humans ever develop a powerful quantum computer. There are currently several commercial products that use these cryptosystems. Though we have developed defenses against quantum attacks, they are too risky and ineffective to be applied in daily life. The study analyzes hash-based digital signature methods. The evaluation of a digital signature using a Merkle tree. The paper investigates unique ideas using vector commitments and a Verkle tree. The authors of this study describe a novel, Verkle tree technology-based method for creating a digital signature system. This is accomplished using the Verkle tree, vector commitments, and vector commitments based on lattices for post-quantum aspects. This work also provides the theories behind post-quantum signature design using Verkle Tree.

Keywords
Post-quantum cryptography, quantum cryptography, Merkle tree, Verkle tree, vector commitments, lattice-based vector commitments, cryptographical application.

1. Introduction

Quantum computing will eventually prevail and spread more broadly. A cryptographic scheme for classical computers that can fend off attacks from quantum computers is known as post-quantum cryptography, also known as quantum encryption. Computers will be able to carry out complex calculations much more quickly than classical computers if they can take advantage of the special capabilities of quantum mechanics [1]. It should be obvious that a quantum computer might be able to perform some difficult tasks quickly. It is interesting to observe that a normal computer would need many years to accomplish these calculations.

Quantum computing will take over and become more prevalent when we get there. Most, if not all, currently in use conventional cryptosystems will likely be rendered useless by quantum computers. Particularly, systems based on the integer factorization problem (RSA). RSA-based cryptosystems are still widely employed in real-world applications; however, they are susceptible to assaults from quantum computers. The RSA cryptosystem is employed in a wide range of goods and programs. This cryptosystem is now used in an increasing number of commercial devices [2]. Because it is mostly utilized in encryption technologies, the RSA algorithm can be regarded as one of the most frequently used public key cryptosystems that develop with technology [3–7].

Many alternatives to RSA systems have been proposed, but none of them can be used in practice due to security or performance issues. One of the many proposed signature techniques is the hash-based one. Systems' security depends on the hash function's resistance to collisions because random integers are used as the initial random sequence [8]. It requires a lot of effort to create

CPITS-2023-II: Cybersecurity Providing in Information and Telecommunication Systems, October 26, 2023, Kyiv, Ukraine
EMAIL: miavich@cu.edu.ge (M. Iavich); tamari.kuchukhidze@gmail.com (T. Kuchukhidze); t.okhrimenko@npp.nau.edu.ua (T. Okhrimenko)
ORCID: 0000-0002-3109-7971 (M. Iavich); 0000-0003-1997-465X (T. Kuchukhidze); 0000-0001-9036-6556 (T. Okhrimenko)

secure and efficient post-quantum cryptosystems and put them into use.

Many alternatives to RSA systems have been offered, but none of them can be utilized in practice due to security or performance difficulties. The hash-based signature method is one of many that have been suggested. The security of those systems depends on the hash function's ability to withstand collisions because random numbers are utilized to generate the initial random sequences of those systems. Post-quantum cryptosystems need to be developed and put to use securely and effectively, which takes a lot of work [9–10]. Once quantum computing takes over, RSA and other asymmetric algorithms won't be able to protect our personal information. We are working to develop post-quantum systems as a result.

In practice, quantum computer assaults can jeopardize traditional digital signature approaches. Our objective is to create RSA substitutes that are impervious to assaults from quantum computers. Digital signature systems based on hashes are one option. These apps employ the cryptographic hash function. Due to the low collision rates of the hash algorithms they use, these digital signature techniques are safe. The hash-based digital signature method is one RSA substitute. How safe these systems are is dependent on the cryptographic hash functions used.

We examined Merkle tree-based hash-based one-time signature methods. These post-quantum approaches can withstand quantum attacks. Verkle trees, a powerful update to Merkle trees that keep only the most important data, are more effective and enable more efficient verification methods. This reduces the amount of necessary storage space needed. We spoke about the implicit commitments of Verkle trees and vectors. Additionally, vector commitments based on lattices are taken into consideration about post-quantum features.

2. Literature Review

Quantum computers can easily break the present encryption schemes. Therefore, it is now conceivable for attacks made possible by quantum computers to be successful. The study [1] also covers one-way functions and one-time signature methods. Digital signature methods that can withstand assaults from quantum computers are presented in this article [2]. The work [8] includes information on the current state of cryptanalysis as well as the construction of the McEliece public-key encryption system with algorithmic and parameter options.

In [9], a variety of QRNG integration techniques are offered. Scientists are interested in quantum computers, according to the article [10]. Cryptosystems based on the integer factoring problem can be cracked by quantum computers. It suggests that the RSA system, one of the best-known public-key cryptosystems, is vulnerable to attack by quantum computers. The authors of publications [11–15] discuss various quantum number generators based on hash-based digital signature techniques. The Merkle scheme is fully explained in the article [16]. The application of vector commitment is discussed in works [17–19]. Verkle trees are described in this paper [20]. A Merkle tree-like design based on the SIS lattice issue is used in the study [21] to create a stateless updatable VC method.

3. Hash-based One-Time Signature Schemes

One-time signature schemes based on hashes have a lot of potential for the post-quantum era. Such is the operation of hash-based one-time signature techniques. First, key generation must be completed. The process of signing is then followed by the process of verifying the signature. The private key for the signature scheme is created using a secret key that is produced at random.

We concentrate on signature methods whose security is solely derived from the collision resistance of cryptographic hash functions. The Lamport-Diffie One-Time Signature (L-DOTS) scheme is an illustration of such a system [11]. It is assumed that computers have access to a constant supply of really random bits, which are effectively a series of impartial and independent coin tosses when constructing randomized algorithms and protocols. In actual applications, a sample that produces this sequence is obtained via a "source of randomness".

The Lamport-Diffie one-time signature's security parameter n is an integer. A one-way function called $f : \{0,1\}^n \rightarrow \{0,1\}^n$ and a cryptographic hash function $f : \{0,1\}^n \rightarrow \{0,1\}^n$ are used by L-DOTS to construct an L-DOTS key pair [12]. By formula (1), the Lamport-Diffie one-time signature key X is a string of 2n bits with n length that is selected at random.

$$X = (x_{n-1}[0], x_{n-1}[1], \dots, x_1[0], x_1[1], x_0[0], x_0[1]) \epsilon R \ \{0,1\}^{(n,2n)} \qquad (1)$$

The L-DOTS verification key is Y:

$$Y = (y_{n-1}[0], y_{n-1}[1], \dots, y_1[0], y_1[1], y_0[0], y_0[1]) \epsilon \ \{0,1\}^{(n,2n)} \qquad (2)$$

The one-way function f, as defined by expression (3), is used to calculate the key:

$$y_i[j] = f(\ x_i[j]), 0 \leq i \leq n-1, j = 0,1. \qquad (3)$$

To generate a Lamport-Diffie one-time signature key, $2n$ evaluations of the f function are required. The verification and signature keys are n-length, $2n$-bit strings. During L-DOTS signature creation document $M \, \epsilon \, \{0,1\}^*$ is signed using L-DOTS and signature key X. $g(M) = d = (d_{n-1}, \dots, d_0)$ is the message digest for the message M. $sign = (x_{n-1}[d_{n-1}], \dots, x_1[d_1], x_0[d_0]) \, \epsilon \, \{0,1\}^{(n,n)}$ is the L-DOTS signature.

The n-bit strings of length n are used to create this signature. They are picked as message digest function d. The number of cryptographic functions that a processor can perform at a given time is usually calculated in hashes per second [13]. The i^{th} bit string of this signature is $x_i[0]$ if the i^{th} bit in d is 0, $x_i[1]$ otherwise. The signature does not require the evaluation of f. The signature is not dependent on the evaluation of f. The signature is n^2 bytes long.

In the case of L-DOTS verification, the verifier generates the message digest $\boldsymbol{d} = (\boldsymbol{d_{n-1}}, \dots, \boldsymbol{d_0})$ if we want to verify $\boldsymbol{M's}$ signature, $\boldsymbol{sign} = (\boldsymbol{sign_{n-1}}, \dots, \boldsymbol{sign_0})$. Consequently, whether it is or is not is decided:

$$\big(f(sign_{n-1}), \dots, f(sign_0)\big) = (y_{n-1}[d_{n-1}], \dots, y_0[d_0]). \qquad (4)$$

Hash-based one-time signature schemes involve key generation, signature creation, and verification. A secret key is generated at random to create the private key. The secret key and hash function are applied repeatedly to the message, resulting in the signature. The recipient verifies the signature using the same hash function and the received message. The signature is considered valid if the result matches the sent one.

Even though L-DOTS is big, it swiftly creates keys and signatures. The Winternitz One-Time Signature Scheme (W-OTS) is recommended as a way to reduce the number of signatures. W-OTS lowers the number of signatures by signing many bits in a message digest with a single string that serves as the one-time signature key. Like L-DOTS, W-OTS employs a cryptographic hashing method and a one-way function.

Security and the integrity and authenticity of digital signatures are ensured by hash-based one-time signature structures, which require a special secret key for each signature. By guaranteeing the unique key, this strategy stops attackers from producing extra signatures if their system is breached. Due to the restricted use of each key combination, one-time signature schemes tend to be ineffective. To solve this problem, Ralph Merkle suggests a full binary hash tree that limits the use of different one-time verification keys to the root public key of the hash tree.

4. Merkle Tree Authentication Scheme

One-time signature schemes are challenging due to the need for storing n digests, making them impractical for everyday use. The Merkle Tree, a solution, uses a binary tree as the root to replace multiple verification keys with a single public key. This system uses a cryptographic hash function and a one-time Lamport or Winternitz signature scheme.

The customizable Merkle Signature Scheme (MSS) supports any cryptographic hash function and any one-time signature scheme. Users are allowed to select the hash function and signature scheme that best meets their needs and security concerns thanks to this flexibility. We assume that $g : \{0,1\}^* \rightarrow \{0,1\}^n$ is a cryptographic hash function. We also assume that the one-time signature scheme also completes the Merkle scheme by providing the necessary mechanisms for generating one-time signatures that give the necessary security features.

When generating the Merkle signature scheme key pair, the signer chooses $H \in \mathbb{N}$, where $H \geq 2$. As a result, a key pair is created. It will be possible to sign and validate 2^H papers as a result. It should be emphasized that this is very different from signature schemes like RSA, which allow the use of a single key pair to sign or validate numerous documents [14]. In actuality, however, this number is also limited by the methods employed to create the signature or by particular laws [15].

The signer will produce 2^H different key pairs $(X_j, Y_j), 0 \leq j < 2^H$. The verification key in this case is Y_j, and the signature key is X_j. Merkle tree has the following leaves: (Y_j), $0 \leq j < 2^H$. A parent node is the hash value of the concatenation of its left and right offspring. This is how a Merkle tree calculates its internal nodes. The Merkle signature scheme public key serves as the Merkle tree's root. One-time signature keys with a length of 2^H make up the MSS secret key [16].

This figure shows an example where the height of the Merkle tree is H=3.

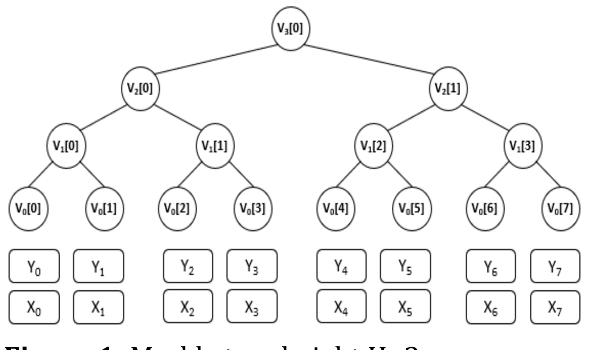

Figure 1: Merkle tree height H=3

Generating an MSS key pair requires computing 2^H unique key pairs and evaluating a $2^{H+1} - 1$ hash function.

One-time signing keys are successfully used by MSS to generate signatures. We must first calculate the n-bit $d = g(M)$ to sign a message on M. Then, using the sth one-time signature key $X_s, s \in \{0, \ldots, 2^H - 1\}$, the signer creates a one-time signature, $sign_{OTS}$. This one-time signature and the matching one-time verification key Y_s make up a Merkle signature.

The signer adds to the message the index s and the authentication path to the verification key Y_S; to validate its authenticity. There are two steps in Merkle's signature verification process. The one-time signature scheme verification algorithm is used by the verifier in the first stage to check the validity of d's signature $sign_{OTS}$, using the one-time verification key Y_s. The second stage of verification involves the verifier determining the validity of the one-time verification key Y_s.

A Merkle Tree with n nodes can be built in $O(n)$ time because Merkle Trees are extremely quick. Unfortunately, their $O(\log_2 n)$ proof size is quite large and can be costly in terms of width. When a Merkle tree has many nodes, the resulting Merkle proofs may be extremely massive. Our local storage may experience a large and costly strain as a result of the Merkle Proof itself.

5. Verkle Tree

Verkle trees are a powerful upgrade to Merkle trees, enabling smaller verifications and increased efficiency [17]. They are essential for post-quantum cryptography due to their ability to lower computing and storage costs while maintaining high-security levels. Verkle trees are more efficient than Merkle trees, as they reduce redundant data and storage space required by intermediate nodes. They provide more effective verification processes by keeping only the necessary data. Verkle trees offer more flexibility than Merkle trees, as they require additional hash calculations to verify the integrity of specific data blocks. This scalability advantage makes them better suited for efficient handling of large datasets.

The Verkle Tree is a method to construct a Merkle Tree using Vector Commitments instead of cryptographic hash functions. To construct a tree, choose k pieces and compute a Verkle Tree using files $f_0, f_1, \ldots, f_n$. Compute a Vector Commitment (VC) for each subset of files and determine if each membership proves PR_i with relation to VC. Compute Vector Commitments across the tree until the root commitment is calculated [18].

In Fig. 2, 9 files with a branching factor of 3 are divided into subsets of size k = 3. Vector Commitment and membership proofs are computed over each subset, leaving obligations VC_1, VC_2, and VC_3. Membership proofs PR_9, PR_{10}, and PR_{11} are computed for commitments VC_1, VC_2, and VC_3 concerning commitment VC_4. The Vector Commitment VC_4 is computed over these commitments, with the root commitment being the digest of the Verkle Tree.

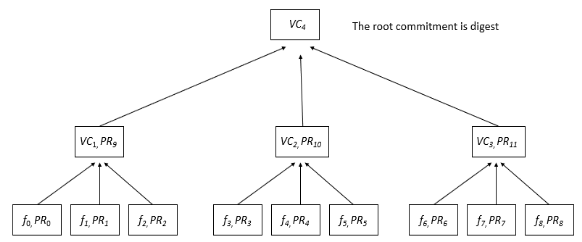

Figure 2: Verkle tree when $K = 3$

The Merkle and Verkle trees have distinct characteristics for proofs. In a Merkle tree, all sister nodes must be considered, requiring proof that contains all nodes. However, the Verkle tree uses “batching nodes” to verify multiple pathways simultaneously, reducing the amount of evidence needed to establish a value. This makes the Verkle tree more efficient and faster than the Merkle tree in proving values.

Verkle trees require only the path and minimal additional information for proof, without sibling nodes. They benefit from a wider width, but Merkle Patricia trees do not. Wider trees lead to shorter routes, but the higher cost of proving every width -1 sister node per level is offset by the Verkle tree's lack of this cost.

A Verkle tree uses a unique hash algorithm, using vector commitments instead of conventional hashes, to compute an inner node from its offspring. This method is more efficient than Merkle trees, as they achieve the same goal but are smaller in size in bytes. The main difference lies in the use of vector commitments for cryptographic hash functions.

6. Vector Commitments

Fundamentals of cryptography called commitment schemes allow a value to be concealed and then revealed. Two essential traits of commitment systems are binding, which limits access to other values, and hiding, which reveals the bare minimum of the value's properties. The VC schemes enhance commitments to accommodate ordered value sequences. One goal of VC schemes is to enable commitment to a vector and then opening at any preferred indices binding, which makes it challenging to open relative to several values at the same time. This goal is combined with potential attribute concealing.

Users can commit to an ordered list of q values, known as a vector, using VC The commitment can be opened concerning specific places in the future (for example, to demonstrate that m_i is the i-th committed message). Position bound requires vector commitments, ensuring adversaries cannot open commitments to two separate values simultaneously. To achieve conciseness [19], the length of the commitment string and the size of each opening must be independent of the vector length.

Security characteristics like hiding property may also be required for vector commitments. The commitment should keep the values and order of the vector’s components a secret. Contrarily, the hiding property does not play a significant role in the execution of vector commitments.

The ability to update VC is necessary. To update the commitment and the related openings, we have two algorithms. By switching the i^{th} message from m_i to m_i' and taking into account the modification of a commitment Com, the committer can acquire a (modified) Com’ holding the updated message. Holders of a message opening at position j concerning Com can use the second way to modify their evidence and make it valid concerning the new Com’.

Multiple methods are employed for committing to and verifying vector messages while taking into account our vector commitment system. The technique employs a message space M, a commitment space Com, and a proof space Pr, depending on the setup settings.

In essence, updating a commitment and proof should produce results that are comparable to those of generating a new commitment and proof on the altered message vector. Because state information is included in the results, numerous updates are possible within polynomial constraints.

To implement VC, we can make either the well-known RSA assumption or the Diffie-Helman assumption. In terms of the effectiveness of the resulting solutions, the "quality" of the underlying assumption, or both, vector commitment provides compact and effective solutions that significantly outperform prior studies [20].

However, the resulting techniques must protect us from attacks by quantum computers. Unfortunately, quantum computers can currently defeat VC based on RSA. In this article, we strengthen earlier vector commitment and RSA assumption-based schemes to increase their efficiency and security. We are developing Verkle tree-based signature systems, but we construct vector commitments using lattices. Our schemes are predicated on post-quantum assumptions.

7. Lattice-Based Vector Commitment

The use of VC methods enables concisely committing to an ordered series of values, enabling the illustration of desired points in a condensed manner. Since VCs can be updated without knowing the complete vector, modifications to commitments and proofs are possible. Cryptographic accumulators, external databases that have been verified, and cryptocurrencies are only a few of the important cryptographic uses that VCs have. They are helpful for databases with zero knowledge, cryptographic accumulators, and pseudonymous credentials, as well as for efficiently updated, publicly verifiable databases.

The research on post-quantum VC methods, or those that are conceivably secure against quantum attacks, has likewise been rather sparse. It is possible to employ Merkle trees that were created using a post-quantum hash function, but they have updates that are comparatively expensive and unavoidably stateful. A stateless updatable VC scheme is produced naturally by a Merkle tree-like construction presented in the article [21] that is based on the SIS lattice problem.

This study introduces stateless updatable vector commitment constructions based on the SIS lattice problem, allowing for secure commitment and verification of vector messages. The constructions are efficient and shorter than previous stateless updatable constructions, using private-key configuration and a central authority for public parameters.

Due to the committer and verifier parameters' widths being quadratic and linear in the message dimension d, respectively, in our prior construction's VC scheme, the construction is unsuited for large dimensions in its current state. We provide an analysis of a general d-ary tree construction that, with no increase in the sizes of the parameters or commitments, converts a VC scheme for dimension d into one for dimension d^h for any desirable positive integer h. The only sizes that increase are the proof and proof-update sizes, which grow linearly with h but independently of d. The stateless updatability quality of the basic scheme and the combinability of commitments and proofs, both of which are crucial for distributed VCs, are not preserved by this modification [22–25].

Here, we provide a customized tree-like transformation of our VC scheme built on SIS. Our transform, in contrast to the general one, keeps combinability and (differential) stateless updates because commitments are essentially linear in committed messages, though at the cost of slightly larger objects and a stronger SIS requirement. The transformation is based on the context's core idea that was employed in a Merkle-tree-like structure based on SIS (which can be used as a VC).

In that case, the proof size ends up being proportional to $hd\log^2\left(d^h\right)$ since a proof must contain all of the brother-node information for each step in a root-to-leaf path. (The length of the proofs along the path and the sizes of the integers inside of them are the sources of the $l\log^2\left(d^h\right)$ factor.) The same general principle holds true for our SIS-based VC scheme, with the added benefit that proofs need not include sibling information, meaning that the proof size increases simply as $h\log^2\left(d^h\right) = h^3\log^2\ d$.

For any choice of magnitude d and tree height h, our design is quantitatively a strict improvement (although at the cost of private setup). Additionally, according to its performance profile, using a moderately large d and a correspondingly smaller h can ultimately produce an asymptotic proof size that is equivalent to that of the generic tree transformation for VCs while maintaining combinability and stateless updates (though at the cost of private setup and a stronger SIS assumption).

Our constructs employ conventional methods for preimage sampling and SIS—based trapdoors. The "gadget" matrix, abbreviated G, is used in the constructions This matrix has an integer dimension of n by w. In building the trapdoor, the matrix G is essential. G illustrated by the formula: $\mathrm{G} = \mathrm{I}_n \otimes (1, 2, \ldots, 2^{\lceil \log_2 q \rceil - 1})$, where I_n is the n×n identity matrix, and $\otimes$ stands for the Kronecker product. This illustration serves as a model, although any acceptable matrix G with certain characteristics may be utilized. Deterministic function $\mathrm{G}^{-1}: \mathbb{Z}_q^n \to \mathbb{Z}^w$ is an inversion operation with particular characteristics. The formula $\mathrm{G} \cdot \mathrm{G}^{-1}(\mathrm{u}) = \mathrm{u}$ for small values of g, and the norm of $\mathrm{G}^{-1}(\mathrm{u})$ is constrained by g for all u in $\mathbb{Z}_q^n$.

G is a matrix with n×w dimensions and values Z_q. When performing the G^{-1}operation (computing the inverse of G), the magnitude bound g_{G} for the gadget matrix G is employed. Vectors for messages are chosen from a set $\bar{M}$. $\bar{M}$ is a collection of w-dimensional vectors, where each element is drawn from a subset of $\bar{I}$. $\bar{I}$ is a subset of the integers and is defined by the values— M_I and M_I. The chosen integer, h The chosen integer. We employ algorithms that were constructed for previous construction [26–29].

The following algorithms have been defined.

Setup Algorithm:

The $\overline{\mathrm{Setup}}_h$ algorithm generates outputs based on input parameters. A commitment parameter cp, a matrix U (made up of multiple submatrices), and vp are included in outputs.

$\mathrm{U}^{(1)}$ is the same as the matrix U that was discussed earlier. The matrix $\mathrm{S}^{(1)}$ is an identity matrix of size wd×wd.

For each k from 1 to h:

$$\mathrm{S}^{(k)} = \mathrm{I}_d \otimes \mathrm{G}^{-1}\left(\mathrm{U}^{(k-1)}\right) \in \mathbb{Z}^{wd \times wd^k} \quad (5)$$

$$\mathrm{U}^{(k)} = \mathrm{U}\mathrm{S}^{(k)} \in \mathbb{Z}_q^{n \times wd^k} \quad (6)$$

The inverse of G and the matrix $\mathrm{U}^{(k-1)}$are used in an operation to produce $\mathrm{S}^{(k)}$. The matrix $\mathrm{U}^{(k-1)}$ is multiplied by $\mathrm{S}^{(k)}$ to get $\mathrm{U}^{(k)}$. Each of the blocks that make up $\mathrm{U}^{(k)}$ can be calculated separately using U and a value $\bar{J}$ ($\bar{J} \in [d^k]$) without having to calculate the complete matrix.

Commit Algorithm:

From 1 to h, for each k:

The $\overline{\mathrm{Commit}}_k$ algorithm requires a commitment parameter cp and a message vector $\overline{\mathrm{m}}$. Applying the inverse of G on $\mathrm{U}^{(k)}$ results in the computation of the value $\bar{\mathrm{c}}$. Outputs include $\bar{\mathrm{c}}$ and $\overline{st}$.

Open Algorithm:

For k = 1, the $\overline{\mathrm{Open}}_1$algorithm is the same as the "Open" operation.

For each k from 2 to h:

- Based on the given values and the value $\bar{\iota}'$, a value $\bar{\iota}$ is calculated.
- Subvectors are created from a message vector $\overline{\mathrm{m}}$.
- The Open algorithm and specific inputs are used to determine the p_i value.
- Using the Commit algorithm from the prior phase, a commitment value $(\bar{\mathrm{c}}_{i,-})$ is generated.
- Using specific inputs and the Open algorithm, a value $\bar{p}'_{\bar{v}'}$ is produced.
- The result is $\bar{p}_{\bar{z}}$.

Verify Algorithms:

Using the Verify operation, the $\overline{\mathrm{Verify}}_1$ algorithm verifies some inputs for k = 1.

For each k from 2 to h:

- An index i and a value $\bar{\iota}'$ are defined based on $\bar{\iota}$. The components of the $\bar{p}_{\bar{z}}$ value is separated into components. The process is rejected if a few verification requirements are not satisfied.
- The process is denied if the verification from the (k–1)th algorithm—$\overline{\mathrm{Verify}}_{k-1}$ fails. If not, it's accepted.

Update Algorithms—From the linearity of commitment operations, update algorithms can be inferred.

8. Novel Scheme Using Verkle Tree

Implementing one-time signature methods can be challenging because a different key pair must be used to sign every message. These systems' drawback is that n digests must be saved, which makes them impractical for routine use. Therefore, regardless of the number of files we have, we would require a method that allows us to save a uniform-sized digest. The Merkle tree was suggested as a solution to this issue. This method can replace numerous verification keys with a single public key by employing a binary tree as the root [30–33].

Merkle trees are quickly computed; it takes $O(n)$ time to build a Merkle tree with n nodes. A Merkle tree with several nodes can be used to generate large Merkle proofs. Unfortunately, the size of their $O(\log_2 n)$ proof is extremely huge and can be expensive. The Merkle proof alone might be a heavy and costly burden on our local storage. To sign 2^n messages, the tree must be n heights tall. The size of their proofs, $O(w \log_w n)$, is more than that of Merkle Trees when using wider-width trees (w-ary trees). The proof size is fixed at $O(1)$, when a vector commitment method is used, however, the building of the vector commitment is extremely time-consuming and costly, requiring an $O(n^2)$ calculation.

Merkle proofs can be vastly improved using Verkle trees, which allow for significantly reduced proof volumes. The w width Verkle Tree can be built in just $O(wn)$ time. The verifier simply needs to offer a single proof that demonstrates all parent-child ties between all commitments along the paths from each leaf node to the root, as opposed to having to present all "brother nodes" at every level. When compared to perfect Merkle trees, proof sizes can be reduced by a factor of 6–8.

Instead of using the Merkle tree, we use the Verkle tree. $H \in \mathbb{N}, H \geq 2$ are chosen by the signer when a key pair is formed. After that, the key pair is generated. They will enable the validation and signing of 2^H papers. The signer will generate 2^H distinct key pairs $(X_j, Y_j), 0 \leq j < 2^H$. In this case, the verification key is Y_j and the signature key is X_j. Both of them are bit strings. The leaves of the Verkle tree are $g(Y_j)$, $0 \leq j < 2^H$. Each node is a hash value that is formed by concatenating the hashes of its offspring, and they are calculated and used as the tree's leaves. The public key is the primary commitment in the Verkle cryptography system. To create a public key, 2^H pairs of keys must be calculated [34].

One-time signature key generation allows us to create signatures. To sign a message on M, the n-bit $d = g(M)$ must first be calculated. An arbitrary size message of size m is first changed into a message of size n using the hash function. The document's signature is made up of the root commitment, one-time signature, one-time verification key, and finally, indexes for the evidence [35–36].

The one-time signature of $sign$ should be authenticated using Y_s according to how Verkle's signature verification works. The VC_i commitments are confirmed if this is the case. If the root of the tree matches the root commitment, the signature is verified. The root commitment in a Verkle tree is digest d.

9. Conclusions

The paper gave a thorough overview of cryptography methods that might be used in both traditional and quantum settings. We covered post-quantum cryptography systems. We discussed hash-based one-way functions, as well as how we can integrate them into Merkle and Verkle trees. Lattice-based commitments and vector commitments were investigated. We spoke about how to calculate and integrate the effective Merkle tree—Verkle tree enhancement. Novel Shames' success inspired the development of a new model, which then included Verkle tree. While they do necessitate more difficult cryptography, there may be huge scaling benefits.

The defense against attacks from both classical and quantum computers is essential for us, so the resulting schemes must be. After reviewing the work that had been done, we were given the systems that were incorporated into Merkle. Merkle Trees, which are built using cryptographic hash functions, provide a strong defense against quantum assaults even though the verification size is too big. Each child in a Merkle tree is represented by the hash of the parent node.

Our revised Verkle tree model defines a parent node as the vector commitment of its children. To implement the new technology, we discussed vector commitment and commitments based on hard lattice issues. The Verkle system allows for substantially smaller verifications, which is a significant improvement over the Merkle scheme. Instead of showing all nodes at each level, verification only needs one proof to authenticate all parent-descendant relationships, namely all commits from each leaf node to the root. This makes it possible to reduce the verification size by around 6–8 times when compared to the conventional Merkle method.

In our revised method, a Verkle tree was therefore taken into consideration instead of a Merkle tree. In this case, vector commitment is sufficient to establish the proof. Based on the premise of the lattice, we used vector commitments to build the Verkle tree.

We must ensure that the strategies that emerge safeguard us against attacks from quantum computers. Our earlier vector commitments based on RSA could be compromised by quantum computers. The strategy has since been improved to make it safer and more effective. Verkle trees are used in our signature procedures, whereas lattices are used to create vector commitments. We use post-quantum assumptions to underpin our systems.

10. Acknowledgment

This work was supported by Shota Rustaveli National Science Foundation of Georgia (SRNSF) STEM-22-1076.

References

[1] J. Buchmann, E. Dahmen, M. Szydlo, Hash-based Digital Signature Schemes, Post-Quantum Cryptography, Springer, (2009) 35–93. doi: 10.1007/978-3-540-88702-7_3.

[2] L. Chen, et al, Report on Post-Quantum Cryptography, National Institute of Standards and Technology 12 (2016). doi: 10.6028/nist.ir.8105.

[3] A. Bessalov, et al., Implementation of the CSIDH Algorithm Model on Supersingular Twisted and Quadratic Edwards Curves, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, vol. 3187, no. 1 (2022) 302–309.

[4] A. Bessalov, et al., CSIKE-ENC Combined Encryption Scheme with Optimized Degrees of Isogeny Distribution, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, vol. 3421 (2023) 36–45.

[5] A. Bessalov, et al., Modeling CSIKE Algorithm on Non-Cyclic Edwards Curves, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, vol. 3288 (2022) 1–10.

[6] A. Bessalov, et al., Multifunctional CRS Encryption Scheme on Isogenies of Non-Supersingular Edwards Curves, in: Workshop on Classic, Quantum, and Post-Quantum Cryptography, vol. 3504 (2023) 12–25.

[7] A. Bessalov, et al., Computing of Odd Degree Isogenies on Supersingular Twisted Edwards Curves, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, vol. 2923 (2021) 1–11.

[8] B. Bhaskar, N. Sendrier. McEliece Cryptosystem Implementation: Theory and Practice, Post-Quantum Cryptography, (2008) 47–62. doi: 10.1007/978-3-540-88403-3_4.

[9] I. Maksim, et al,. Advantages and Challenges of QRNG Integration into Merkle, Sci. Practical Cyber Secur. J. 4(1) (2020) 93–102.

[10] A. Gagnidze, M. Iavich, G. Iashvili, Novel Version of Merkle Cryptosystem, Bulletin of the Georgian National Academy of Sciences 11(4) (2017).

[11] L. Lamport, Constructing Digital Signatures From a One Way Function (1979).

[12] M. Iavich, et al., Post-Quantum Digital Signatures with Attenuated Pulse Generator, in: Information Society and University Studies vol. 2698 (2020) 42–45.

[13] M. Iavich, et al., Improvement of Merkle Signature Scheme by Means of Optical Quantum Random Number Generators, Advances in Intelligent Systems and Computing (2021) 478–487.

[14] M. Iavich, A. Gagnidze, G. Iashvili, Hash Based Digital Signature Scheme with Integrated TRNG, CEUR Workshop Proceedings vol. 2145 (2018) 79–82.

[15] A. Hülsing, J. Rijneveld, F. Song, Mitigating Multi-Target Attacks in Hash-Based Signatures, Public-Key Cryptography—PKC 2016 (2016) 387–416. doi: 10.1007/978-3-662-49384-7_15.

[16] R. Merkle, A Digital Signature Based on a Conventional Encryption Function, Advances in Cryptology—CRYPTO'87 (1988) 369–378. doi: 10.1007/3-540-48184-2_32.

[17] H. Chen, D. Liang, Adaptive Spatio-Temporal Query Strategies in Blockchain. ISPRS Int. J. Geo-Inf. 11(7) (2022) 409. doi: 10.3390/ijgi11070409.

[18] W. Wang, A. Ulichney, C. Papamanthou, BalanceProofs: Maintainable Vector Commitments with Fast Aggregation, Yale University, Cryptology ePrint Archive (2022).

[19] K. Kurosaw, H. Goichiro, Public-Key Cryptography—PKC 2013, 16th International Conference on Practice and Theory in Public-Key Cryptography (2013).

[20] J. Kuszmaul, Verkle Trees (2019). URL: https://math.mit.edu/research/highschool/primes/materials/2018/Kuszmaul.pdf

[21] C. Papamanthou, et al., Streaming Authenticated Data Structures, Advances in Cryptology—EUROCRYPT (2013) 353–370. doi: 10.1007/978-3-642-38348-9_22.

[22] M. Iavich, et al., Improved Post-Quantum Merkle Algorithm Based on Threads, Advances in Intelligent Systems and Computing (2021) 454–464. doi: 10.1007/978-3-030-55506-1_41.

[23] M. Iavich, et al., Lattice based Merkle, CEUR Workshop Proceedings, vol. 2470 (2019) 13–16.

[24] Z. Hu, et al., High-Speed and Secure PRNG for Cryptographic Applications, Int. J. Comput. Network Inf. Secur. 12(3) (2020) 1–10. doi: 10.5815/ijcnis.2020.03.01.

[25] B. Bünz, et al., FlyClient: Super-light Clients for Cryptocurrencies, IEEE Symposium on Security and Privacy (SP) (2020) 928–946. doi: 10.1109/SP40000.2020.00049.

[26] S. Tynymbayev, et al., Modular Reduction Based on the Divider by Blocking Negative Remainders, News of the National Academy of Sciences of the Republic of Kazakhstan, Series of Geology and Technical Sciences 2(434) (2019) 238–248. doi: 10.32014/2019.2518-170x.60.

[27] D. Cooper, et al., NIST Special Publication 800-208: Recommendation for Stateful Hash-Based Signature Schemes, NIST (2020). doi: 10.6028/NIST.SP.800-208.

[28] S. Gnatyuk, et al., New Secure Block Cipher for Critical Applications: Design, Implementation, Speed and Security Analysis, Advances in Intelligent Systems and Computing (2020) 93–104.

[29] S. Gnatyuk, et al., Method of Algorithm Building for Modular Reducing by Irreducible Polynomial, 16th International Conference on Control, Automation and Systems (2016) 1476–1479. doi: 10.1109/iccas.2016.7832498.

[30] G. Alagic, et al., Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process, NIST (2022). doi: 10.6028/NIST.IR.8413-upd1.

[31] S. Gnatyuk, T. Zhmurko, P. Falat, Efficiency Increasing Method for Quantum Secure Direct Communication Protocols, IEEE 8th Int. Conf. Intelligent Data Acquisition Adv. Comput. Syst. Technol. Appl. 1 (2015) 468–472. doi: 10.1109/idaacs.2015.7340780.

[32] Q. Yuan, M. Tibouchi, M. Abe, Security Notions for Stateful Signature Schemes, IET Information Security 16(1) (2022) 1–17. doi: 10.1049/ise2.12040.

[33] M. Iavich, et al., Hybrid Encryption Model of AES and ElGamal Cryptosystems for Flight Control Systems, IEEE 5th Int. Conf. Methods Syst. Navigation Motion Control (2018) 229–233.

[34] I. Khaburzaniya, et al., Aggregating and Thresholdizing Hash-Based Signatures Using STARKs, ACM Asia Conf. Comput. Commun. Secur. (2022) 393–407. doi: 10.1145/3488932.3524128.

[35] M. Kalimoldayev, et al., Matrix Multiplier of Polynomials Modulo Analysis Starting with the Lower Order Digits of the Multiplier, News of the National Academy of Sciences of the Republic of Kazakhstan, Series of Geology and Technical Sciences 4(436) (2019) 181–187. doi: 10.32014/2019.2518-170x.113.