symmetry

IMPACT FACTOR 2.7

CITESCORE 4.9

Article

A Post-Quantum Digital Signature Using Verkle Trees and Lattices

Maksim Iavich, Tamari Kuchukhidze and Razvan Bocu

MDPI

https://doi.org/10.3390/sym15122165

symmetry

MDPI

Article

A Post-Quantum Digital Signature Using Verkle Trees and Lattices

Maksim Iavich [1], Tamari Kuchukhidze [2] and Razvan Bocu [3,*]

[1] Department of Computer Science, Caucasus University, Tbilisi 0102, Georgia; miavich@cu.edu.ge
[2] Department of Computer Science, International Black Sea University, Tbilisi 0131, Georgia; tkuchukhidze@ibsu.edu.ge
[3] Department of Mathematics and Computer Science, Transilvania University of Brasov, 500036 Brasov, Romania
* Correspondence: razvan.bocu@unitbv.ro

Abstract: Research on quantum computers has advanced significantly in recent years. If humanity ever creates an effective quantum computer, many of the present public key cryptosystems can be compromised. These cryptosystems are currently found in many commercial products. We have devised solutions that seem to protect us from quantum attacks, but they are unsafe and inefficient for use in everyday life. In the paper, hash-based digital signature techniques are analyzed. A Merkle-tree-based digital signature is assessed. Using a Verkle tree and vector commitments, the paper explores novel ideas. The authors of this article present a unique technology for developing a post-quantum digital signature system using state-of-the-art Verkle tree technology. A Verkle tree, vector commitments, and vector commitments based on lattices for post-quantum features are used for this purpose. The concepts of post-quantum signature design utilizing a Verkle tree are also provided in the paper.

Keywords: quantum cryptography; vector commitments; lattice-based vector commitments; Verkle tree; cryptographical application

Citation: Iavich, M.; Kuchukhidze, T.; Bocu, R. A Post-Quantum Digital Signature Using Verkle Trees and Lattices. *Symmetry* **2023**, *15*, 2165. https://doi.org/10.3390/sym15122165

Academic Editor: Michel Planat

Received: 10 November 2023
Revised: 30 November 2023
Accepted: 4 December 2023
Published: 6 December 2023

1. Introduction

In the future, quantum computing will become more common. Quantum encryption, a technique for regular computers, can protect against attacks from quantum computers. It is also called post-quantum cryptography. Quantum computers can do complex calculations much faster than current computers by using the special properties of quantum physics. For example, a quantum computer could complete tasks that take a regular computer several years in just a short time [1].

Quantum computers will probably break most, if not all, of the standard cryptosystems currently used in practice. RSA-based systems are widely used today and they are at risk of being hacked by quantum computers. Many commercial products and applications rely on the RSA encryption system because it is one of the most commonly used public key cryptosystems, especially in advancing encryption technologies [2].

There have been a number of suggested alternatives to RSA systems, but none of them can be utilized in practice because of security or performance difficulties. Hash-based signature schemes are one of several that have been suggested. Since random numbers are employed as the starting random sequence of systems, their security depends on the hash function's ability to resist collisions [3]. Designing and putting into practice secure and effective post-quantum cryptosystems takes a lot of work.

There have been numerous suggestions for RSA system substitutes, but none of them can be implemented in real life because of performance or security issues. The hash-based signature approach is one of the many that have been suggested. Since random numbers are used to create systems' initial random sequences, the hash function's ability to withstand collisions is crucial to their security. Developing and implementing safe and effective

post-quantum cryptosystems is a time-consuming process. When quantum computing takes over, RSA and other asymmetric algorithms will no longer be able to secure our private data. Because of this, we are aiming to create post-quantum systems [4,5].

In reality, attacks from quantum computers can compromise conventional digital signature technologies. Our objective is to create RSA substitutes that can withstand attacks from quantum computers. Hash-based digital signature schemes represent one of the choices. The cryptographic hash function is used by these schemes. These digital signature methods are secure because the hash algorithms they employ have low collision rates. The safety of these systems is determined by the security of their cryptographic hash functions.

We reviewed hash-based one-time signature schemes that make use of Merkle trees. These schemes are post-quantum and can resist quantum attacks. The problem of these schemes is a very large size of the signature. NIST has accepted hash-based digital signature SPHINC+, but it still has the efficiency problems. SPHINCS+ is a bit larger and slower compared to the other two NIST standards, but it serves as a valuable backup for a key reason: it relies on a different mathematical approach than the three selections made by NIST.

Nevertheless, there are Verkle trees, which are powerful upgrades to Merkle trees, which are more effective and offer more efficient verification procedures by retaining only essential information. This cuts down essential space required for storage purposes. Therefore, replacing Merkle can greatly reduce the size of the signature. In the paper, we discus Verkle tree and vector commitments, which are implicit for Verkle trees.

We present a model of the novel post-quantum digital signature using Verkle trees based on the research. Additionally, lattice-based vector commitments are taken into consideration with regard to post-quantum properties.

2. Literature Review

Current encryption methods are easily broken by quantum computers. As a result, attacks enabled by quantum computers can now be carried out successfully. Digital signature methods that can withstand attacks from quantum computers are presented in article [1]. Paper [2] also covers approaches for one-time signatures and one-way functions. Work [3] provides an in-depth analysis of the state of cryptanalyses as well as the implementation of the McEliece public-key encryption system with algorithmic and parameter options.

According to article [4], scientists are interested in quantum computers. Cryptosystems that rely on the integer factoring problem are susceptible to breach by quantum computing. It implies that the RSA system, one of the most well-known public-key cryptosystems, is vulnerable to attack by quantum computers. Numerous Quantum Random Number Generation integration methods are provided in [5]. Different quantum number generator-based hash-based digital signature schemes are discussed by the authors of papers [6–9]. In article [10], the Merkle plan is described in full. In papers [11–13], the application of vector commitment is described. Additionally, studies [11,14] describe Verkle trees. In paper [15], we have a Merkle-tree-like construction based on the SIS lattice problem, which gives us a stateless updatable VC scheme.

3. Hash-Based One-Time Signature Schemes

Hash-based signature schemes are a type of cryptographic signature scheme that creates digital signatures using the properties of cryptographic hash functions. The basic idea behind hash-based signatures is to hash a message and then use some transformation of the hash value as the signature. Unlike traditional digital signature schemes based on public-key cryptography (such as RSA or ECDSA), hash-based signatures do not rely on the mathematical difficulty of certain problems like factoring large numbers or solving elliptic curve discrete logarithms. Therefore, these schemes can be used in a post-quantum epoch.

In the realm of digital signatures, crucial for identity verification in digital transactions and remote document signing, NIST has chosen three algorithms: CRYSTALS-Dilithium,

FALCON, and SPHINCS+. SPHINCS+ is the representative of the hash-based digital signature family.

The following is how hash-based one-time signature methods operate. The creation of keys must come first. Next comes signature creation, and lastly comes signature verification. The private key for the signature scheme is created by randomly generating a secret key. The secret key must be kept private. The message is subjected to repeated application of the secret key and hash function to generate a signature for that specific communication. The signature is the result of the hash function after every iteration. Using the same hash function and the message they received, the receiver of the signature confirms its legitimacy. The message is concatenated with the public key (obtained from the secret key), and then the hash function is applied repeatedly. If the outcome corresponds with what was sent, the signature is considered legitimate.

Hash-based one-time signature methods show great potential for the post-quantum era. We focus on signature schemes that rely entirely on the collision resistance of cryptographic hash functions for their security. An example of such a scheme is the Lamport–Diffie one-time signature (LDOTS) system [6]. Assuming computers have access to a constant supply of truly random bits—basically, a series of impartial and independent coin flips—is necessary when creating randomized algorithms and protocols. In real-world applications, a sample that produces this sequence is obtained from a "source of randomness" [7].

We consider the Lamport–Diffie one-time signature's security parameter n to be an integer. LDOTS generates an LDOTS key pair using a one-way function, $f : \{0,1\}^n \rightarrow \{0,1\}^n$, and a cryptographic hash function, $g : \{0,1\}^n \rightarrow \{0,1\}^n$, to generate a Lamport–Diffie one-time signature key pair. Expression (1) states that the string of $2n$ bits of length n, which makes up the LDOTS signature key X, is selected at random.

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

The Lamport–Diffie one-time signature verification key is Y, which is calculated according to expression (2):

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

The calculation of the key is conducted using the one-way function f, as represented by function (3):

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

The Lamport–Diffie one-time signature key generation therefore requires $2n$ evaluations of f. $2n$-bit strings of n length make up the signature and verification keys. If an LDOTS signature is generated, document $M \epsilon \{0,1\}^*$ is signed using LDOTS with signature key X. The message digest of M is $g(M) = d = (d_{n-1}, \ldots, d_0)$. The LDOTS signature is $sign = (x_{n-1}[d_{n-1}], \ldots, x_1[d_1], x_0[d_0]) \epsilon \{0,1\}^{(n,n)}$.

A length of n bit strings is used to construct this signature. They are selected as function d for the message digest. A common way to measure how many cryptographic operations a CPU can execute at once is to look at hashes per second [8]. The i-th bit string of this signature is $x_i[0]$, but if the i-th bit in d is 0, the i-th bit string of this signature is $x_i[1]$. The signature can be obtained without the evaluation of f. The signature is n^2 in length.

Considering the instance of LDOTS verification, if we want to verify M's signature, $sign = (sign_{n-1}, \ldots, sign_0)$, the verifier produces the message digest $d = (d_{n-1}, \ldots, d_0)$. Consequently, it is decided whether it is or it is not:

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

The LDOTS generates keys and signatures fairly quickly, even with the large size of the signature. It is advised to use the Winternitz one-time signature scheme (WOTS) to lower the quantity of signatures. The idea is to sign several bits in a message digest with a

single string, or to use a single string in a one-time signature key. Similar to LDOTS, WOTS employs a cryptographic hash function and a one-way function.

A crucial characteristic of hash-based one-time signature structures is that the secret key is only ever utilized to produce a single signature. This makes sure that an attacker cannot produce additional signatures, if the secret key is compromised. There is a major security benefit to using the secret key in this particular way. We use a hash-based method only once to ensure the accuracy and legitimacy of digital signatures.

In most real-world scenarios, one-time signature approaches are ineffective since a key pair can only be used once to create a signature. Ralph Merkle offered a solution to this issue. He recommends employing a complete binary hash tree. Using a full binary hash tree, the goal is to limit the authenticity of an arbitrary, but fixed, number of one-time verification keys to one public key, which serves as the hash tree's root.

4. Merkle Tree Authentication Scheme

It is difficult to use one-time signature schemes because each message requires a different key pair. The problem is that this requires storing numerous digests (n), which is impractical for everyday use. Ideally, we want a method where we can save a consistent-sized digest regardless of the number of files. An idea to deal with this problem was the Merkle tree. It substitutes a single public key for numerous verification keys using a binary tree structure. Using a one-time Lamport or Winternitz signature scheme, this system integrates a cryptographic hashing function. Any of these functions, and any one-time signature scheme, can be utilized with the flexible Merkle signature scheme (MSS). Because of this adaptability, users can select the hash function and signature scheme that best meet their requirements and assurance levels. In this case, we imagine the existence of a cryptographic hash function, abbreviated as $g : \{0,1\}^* \rightarrow \{0,1\}^n$, that converts binary strings of any length to binary strings of a fixed length n. The Merkle system uses the hash function g as a fundamental building block to produce safe and trustworthy signatures. Additionally, by providing the essential mechanisms for generating one-time signatures that offer the necessary security features, the already selected one-time signature scheme accomplishes the Merkle scheme.

When the person signing selects $H \in \mathbb{N}$, where $H \geq 2$, this creates the MSS key pair. Consequently, a key pair is generated. This will make it feasible to sign and validate 2^H documents. It should be noted that this differs significantly from signature protocols like RSA and ECDSA, where a single key pair may be used to sign/verify a large number of documents. Nevertheless, in practice, this figure is also limited by the instruments utilized to create the signature or by particular laws [9].

For every $0 \leq j < 2^H$, the one who signed will produce 2^H unique key pairs $(X_j, Y_j), 0 \leq j < 2^H$, where X_j is the signature key and Y_j is the verification key. The Merkle tree's leaves are $g(Y_j)$, where $0 \leq j < 2^H$. The following formula determines a Merkle tree's internal nodes: a parent node's hash value is equal to the sum of its left and right children. The Merkle tree's base is the MSS public key. A series of 2^H signature keys make up the MSS secret key [10].

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

In order to validate Y_s, the signer additionally appends the authentication path and index s to the verification key Y_s. There are two steps in the verification of Merkle's signature. First, the verifier employs the matching one-time signature scheme verification algorithm to verify d's signature $sign_{\mathrm{OTS}}$ using the one-time verification key Y_s. The verifier assesses the one-time verification key Y_s trustworthiness in the second step.

Merkle trees are quick to compute; in O(n) time, a Merkle tree with n nodes can be created. Merkle proofs of a Merkle tree with a large number of nodes may then be

unreasonably large. The tree's height needs to be n in order to sign 2^n messages. Our local storage may be severely and expensively taxed by the Merkle proof itself.

5. Verkle Tree

Strongly superior to Merkle trees, Verkle trees enable much smaller verifications and are more effective [11]. The ability of Verkle trees to reduce compute and storage costs while preserving high security makes them indispensable for post-quantum cryptography. Compared to standard Merkle trees, Verkle trees are more effective. As the amount of cryptographic data grows, Merkle trees require more processing and storage. Verkle trees provide a solution to this problem by reducing redundant data and the amount of storage space required by intermediate nodes, considering scenarios when speed and efficiency are essential relative to applications with limited resources. Verkle trees retain only the data that are required, resulting in verification procedures that are more efficient.

Comparatively speaking, Verkle trees offer greater flexibility than Merkle trees. Merkle trees need more hash computations as the dataset gets larger in order to confirm the integrity of particular data blocks. Conversely, Verkle trees lessen the need for pointless intermediary nodes, which lowers the amount of hash computations required for verification. Verkle trees have the advantage of scalability, which makes them more appropriate for handling massive databases effectively.

The primary assertion of the Verkle tree is that vector commitments, rather than cryptographic hashing functions, can be used to create a Merkle tree. We first choose how many pieces to divide our tree into (k pieces). After that, let us compute a Verkle tree using the files $f_0, f_1, \ldots, f_n$. After splitting our files into k sub-groups, we also compute a vector commitment over each of the portions of files. Additionally, for each file f_i in the subset, we determine whether each membership of the vector commitment proves PR_i with relation to VC. Then, along the tree, until we calculate the root commitment, we compute vector commitments across previously computed commitments [12].

There are nine files and a branching factor of three in Figure 1. Specific sets of size k = 3 are created from the files, and membership proofs and a vector commitment are calculated for each group. We are now left with the obligations VC_1, VC_2, and VC_3. In addition to computing the membership proofs PR_9, PR_{10}, and PR_{11} for the commitments VC_1, VC_2, and VC_3 with respect to the commitment VC_4, we also compute the vector commitment VC_4 over these three commitments. The root commitment, in this case, is VC_4.

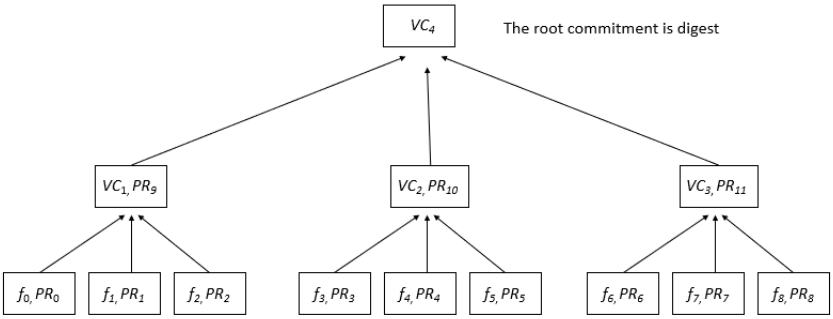

Figure 1. A Verkle tree—$K = 3$.

Each of them has specific characteristics when offering Merkle and Verkle proofs, and the Verkle tree is an improved form of the Merkle tree. You must consider all of the sister nodes in the tree, whose parent has a relationship with the node you want to check if you want to prove a value in a Merkle tree. This indicates that the proof must contain all nodes, which can take a lot of time. When it comes to providing proof, the Verkle tree requires a different approach. It relies on "batching nodes" to verify multiple pathways at once, significantly reducing the amount of evidence needed to establish a value. As a result, while proving values, the Verkle tree is quicker and more effective than the Merkle tree.

Verkle trees do not even need sibling nodes, unlike Merkle trees, which only need the path and a small amount of extra information as proof. Consequently, a wider width is advantageous to Verkle trees but not to Merkle Patricia trees. Both scenarios yield shorter routes with a wider tree; however, in a Merkle Patricia tree, this advantage is lessened by the additional expense of proving each width -1 sister node at each level. This cost is absent in a Verkle tree.

A Verkle tree computes an inner node from its descendant using a hash algorithm different from a conventional hash. A vector commitment is used instead. This small parameter will help us demonstrate our point. The primary statement of the Verkle tree is that a Merkle tree can be produced by replacing the cryptographic hash functions with vector commitments. The same objective is achieved with a Verkle tree as with a Merkle tree. The main difference is that they are much more efficient in terms of size in bytes.

6. Vector Commitments

Commitment schemes are cryptographic fundamentals that enable a value to be hidden and later exposed. Two essential features of commitment systems are hiding, which reveals only the most essential aspects of the value, and binding, which limits access to other values.

Commitments are expanded to incorporate ordered value sequences in the vector commitment (VC) schemes. Enabling commitment to a vector and then opening at any preferred indices' binding is one of the goals of VC schemes, along with potential attribute hiding. This makes it challenging to open relative to different values simultaneously.

With vector commitment, users can commit to a vector—an ordered list of q values—instead of to individual messages (VC). This is carried out in order to enable the commitment to be opened in the future with regard to particular locations (e.g., to prove that m_i is the i-th committed message). More specifically, vector commitments are required for position bounds. An adversary should not be able to openly commit to two different values at the same time, according to the idea of position binding. The length of the commitment string and the size of each opening must be independent of the vector length in order to meet our conciseness criteria [13].

Vector commitments may also need to have security properties, such as, hiding property, which stipulates that it should be hard to identify whether a commitment was made to the vectors $(m_1, \ldots, m_q)$, or to $\left(m'_1, \ldots, m'_q\right)$. That is, the commitment should not divulge any information about the members of the vector, such as their values or order. The implementation of vector commitments, on the other hand, is not heavily reliant on the hiding attribute.

Furthermore, the ability to update vector commitments is required. Thus, two algorithms are provided to update the commitment and the associated openings. Considering the amendment of a commitment, Com, the committer can obtain a Com', a modified commitment, containing the revised message by changing the i-th message from m_i to m'_i. Holders of a message opening at position j with respect to Com may amend their evidence using the second approach to make it legitimate with respect to the new Com'.

Considering our vector commitment system, multiple techniques are used for committing to and verifying vector messages. Depending on the configuration options, the scheme uses the message space M, commitment space Com, and proof space Pr.

The following algorithms are provided with special interfaces by the scheme:

- Setup$\left(1^\gamma, 1^d\right)$—This algorithm takes security parameter γ, and a value d as input, and generates public committer parameters (cp), and verifier parameters (vp).
- Commit$\left(\text{cp}, m \in \text{M}^d\right)$—Given the committer parameters (cp) and a message m from the message space M^d, this algorithm outputs a commitment c $\in$ Com, and a committer state (st).

- Open$(\mathrm{cp}, \mathrm{st}, i \in [d])$—Given the committer parameters (cp), committer state (st), and an index i from the range d, this algorithm produces a proof pr_i for the i-th entry of the committed message associated with st.
- Verify$(\mathrm{vp}, c \in \mathrm{Com}, i \in [d], m \in \mathrm{M}, \mathrm{pr} \in \mathrm{Pr})$—This algorithm takes the verifier parameters (vp), commitment (c), index (i), message (m), and proof (pr) as input, and determines whether the proof is valid or not.

The correctness condition of the scheme ensures that for any polynomial value d and message $m \in \mathrm{M}^d$, and for a given setup (cp, vp), commitment (c, st) obtained from Commit, and proof pr_i obtained from Open, the Verify algorithm accepts with high probability, indicating that the commitment and proof are valid.

Furthermore, if the scheme offers a collection of algorithms with the following interfaces, it can be categorized as updatable:

- PrepareUpdates $\left(\mathrm{cp}, \mathrm{st}, j \in [d], m'_j \in \mathrm{M}\right)$—This algorithm takes the committer parameters (cp), committer state (st), index (j), and a new message entry (m'_j), and produces a commitment update (σ_c), proof update (σ_{pr}), and state update (σ_{s}) required to modify the i-th entry point of the committed message vector.
- UpdateC $(\mathrm{vp}, c \in \mathrm{Com}, \sigma_c)$—Given the verifier parameters (vp), commitment (c), and commitment update (σ_c), this algorithm deterministically produces an updated commitment (c').
- UpdateP $\left(\mathrm{vp}, i \in [d], \mathrm{pr}_i \in \mathrm{Pr}, \sigma_{\mathrm{pr}}\right)$—Given the verifier parameters (vp), index (i), proof (pr_i), and proof update (σ_{pr}), this algorithm deterministically generates an updated proof (pr'_i).
- UpdateS $(\mathrm{cp}, \mathrm{st}, \sigma_{\mathrm{s}})$—Given the committer parameters (cp), committer state (st), and state update (σ_{s}), this algorithm deterministically produces an updated committer state (st').

If the scheme satisfies the above interfaces, it allows for the implementation of modifications to the committed message vector by generating appropriate commitment, proof, and state updates. The scheme can further be classified as stateless updatable if PrepareUpdates can be implemented without requiring the committer state st as an input. In this case, PrepareUpdates^{nost} is used, which takes the committer parameters cp, index j, and old and new message entries m_j and m'_j as inputs.

Furthermore, the scheme can be considered differentially updatable if PrepareUpdates^{nost} (and thus PrepareUpdates) can be used to implement PrepareUpdates^{diff}. This alternative algorithm takes the committer parameters cp, index j, and a "difference" $\sigma = m'_j - m_j$ as inputs, where the operation denotes an abstract operation on the message space M. This allows for more compact representations of the updates.

The updatable scheme's correctness condition guarantees that the outputs of two experiments are statistically identical for any polynomial value d, committer and verifier parameters (cp, vp) supplied from Setup, and messages m and m' that differ in at most the j-th coordinate. While the second experiment creates a new commitment and proof on the revised message vector, the first experiment focuses on committing, opening, and updating the commitment, proof, and state.

Updating a commitment and proof should effectively provide results that are comparable to creating a new commitment and proof on the altered message vector. The incorporation of state information into the results enables composability, allowing for numerous updates within exponential bounds.

Compact and efficient solutions that significantly outperform earlier research in terms of the "quality" of the fundamental assumption, the effectiveness of the generated solutions, or both are made possible by vector commitment [14].

However, it is crucial that the approaches that emerge protect us from quantum computer challenges. Unfortunately, quantum computers can currently break vector commitments based on RSA and other popular asymmetrical systems. We are enhancing the framework to make it more effective and safer in this part. While we employ lattices to

construct vector commitments, we are working on developing signature systems that will utilize Verkle trees. We build our schemes on post-quantum assumptions.

7. Lattice-Based Vector Commitment

It is possible to commit to an ordered series of values succinctly using vector commitment (VC) methods, so that the values at needed points can then be demonstrated succinctly. Additionally, a vector can be stateless updatable, meaning that commitments and proofs can be updated to reflect changes to individual entries while only being aware of those changes and not the vector as a whole.

Numerous cryptographic uses have been discovered for vector commitments. VCs have discovered significant uses for cryptocurrencies, cryptographic accumulators, and verified external databases. They are helpful for databases that are efficiently updated and are publicly verifiable.

On the other hand, very little research has been performed on post-quantum vector commitment schemes—that is, ones that are conceivably safe from quantum attacks. Merkle trees built with a post-quantum hash function can be employed, but they are impacted by updates that are required and relatively inefficient. We present a stateless, updatable VC scheme directly from a Merkle tree-like construction based on the SIS lattice problem [15].

We give constructions of post-quantum vector commitments based on the traditional Short Integer Solution lattice problem, suitably defined in this work. Compared to the only prior post-quantum stateless updatable construction, we present new stateless updatable VCs that are more efficient and have substantially shorter proofs. With our private-key configuration, public parameters are generated by a central authority prior to its downtime.

Based on the post-quantum SIS lattice problem, we constructed new vector commitments. The protocol enables vector message verification and secure commitment. A stateless, updatable "base" VC structure is the first of these. Because of the public parameters' quadratic dependence on d, it is especially suitable for only a moderately big d.

The construction of the scheme uses a vector space, M, where messages are vectors of length $\updownarrow$, and belong to an interval I of contiguous integers. The maximum magnitude of integers in I is denoted as M_I. We have gadget matrix G and an invertible-difference encoding scheme. The encoding maps each index i in the range $[d+1]$ to a matrix, $\mathrm{H}_i \in Z_q^{n \times n}$, such that the difference $\mathrm{H}_{i'} - \mathrm{H}_i$ is reversible for any i, i' in $[d+1]$.

Setup—This algorithm generates the committer parameters (cp), and verifier parameters (vp). It involves choosing a random matrix ($\overline{\mathrm{A}} \leftarrow \mathbb{Z}_q^{n \times m}$), and performing the TrapGen algorithm to obtain matrices A and T. The algorithm constructs A_i matrices and a random matrix (U), where each U_j is in $\mathbb{Z}_q^{n \times \updownarrow}$. $\mathrm{R}_{i,j}$ matrices are derived using the SamplePre algorithm, ensuring that $\mathrm{H}_d - \mathrm{H}_i$ is invertible. The output of the Setup algorithm is $\mathrm{cp} = \left(\mathrm{U}, \mathrm{R} = \left(\mathrm{R}_{i,j}\right)_{i,j \in [d]}\right)$, $\mathrm{vp} = (\mathrm{A}, \mathrm{U})$.

Commit—Given the committer parameters (cp) and a message (m) from the message space (M^d), this algorithm computes the commitment (c) as the sum of element-wise products of U and m. The state (st) is set to the message (m).

Open—This algorithm takes the committer parameters (cp), the committer state (st), and an index (i), and computes the proof (pr_i) as the element-wise product of $\mathrm{R}_{i,j}$ and m_j, where $\mathrm{R}_{i,j}$ is the j-th row of the matrix (R_i) associated with the i-th entry of the committed message.

Verify—The verifier algorithm takes the verifier parameters (vp), the commitment (c), the index (i), the message (m_i), and the proof (pr_i) as input. It verifies the proof by checking the conditions $\| \mathrm{p}_i \| \leq \gamma$ and $\mathrm{c} = \mathrm{A}_i \mathrm{p}_i + \mathrm{U}_i \mathrm{m}_i$. Here, γ is a security parameter. If the conditions are met, the algorithm accepts the proof; otherwise, it rejects it.

We also have updated algorithms to modify the commitment, proof, and state.

- PrepareUpdates^{diff}–This algorithm takes the committer parameters (cp), an index (j), and a difference ($\sigma \in \mathbb{Z}^{\updownarrow}$) as input. It generates the commitment update (σ_c), the proof

update (σ_{pr}), and the state update (σ_s) required to change the j-th admission of the committed message vector.

- UpdateC—Given the verifier parameters (vp), the commitment (c), and the commitment update (σ_c), this algorithm deterministically computes the updated commitment (c').
- UpdateP—Given the verifier parameters (vp), the index (i), the proof (pr_i), and the proof update (σ_{pr}), this algorithm deterministically generates the updated proof (pr'_i).
- UpdateS—Given the committer parameters (cp), the committer state (st), and the state update (σ_s), this algorithm deterministically produces the updated committer state (st').

The algorithms guarantee the accuracy and security of the scheme, which enables secure commitment, opening of proofs, verification, and modifications to committed message vectors.

8. Novel Scheme Using Verkle Tree

Since an individual key pair must be used to sign each message, one-time signature algorithms are especially challenging to implement. These systems' drawback is that they require the saving of n digests, which makes them prohibitively expensive for frequent use. Therefore, we would require an approach that allows us to save a digest of the same size regardless of the number of files we have. That problem was suggested to be solved with the Merkle tree. With that approach, multiple verification keys can be replaced with a single public key by using a binary tree as the root.

Merkle trees are quick to compute; it takes $O(n)$ time to build a tree with n nodes. A multi-node Merkle tree can be used to generate large Merkle proofs. To sign two messages, the tree's height needs to be 2^n. The Merkle proof alone may put a significant and expensive load on our local storage.

Verkle trees, which allow for substantially smaller proof sizes, can greatly enhance Merkle proofs. The verifier only needs to submit a single proof that demonstrates all parent–child relationships between all commitments along the paths from each leaf node to the root, as opposed to having to submit all "brother nodes" at every level. In comparison to ideal Merkle trees, proof sizes can be reduced by a factor of 6–8, and in comparison to Merkle Patricia trees, by a factor of 20–30 or more.

We employ the Verkle tree in place of the Merkle tree. The person signing chooses $H \in \mathbb{N}, H \geq 2$ during key pair formation. The key pair is then generated after that. They will make it feasible to sign and validate 2^H documents. The signer will generate 2^H unique key pairs $(X_j, Y_j), 0 \leq j < 2^H$. In this instance, the signature key is X_j, and the verification key is Y_j. They are both bit strings. The Verkle tree's leaves are $g(Y_j), 0 \leq j < 2^H$. As the leaves of the tree, they are computed and used, and every node is a hash value formed by joining the hashes of its descendants. The root commitment in the Verkle cryptography scheme is the public key. A computation of 2^H pairs of keys is required to produce a public key.

We can create signatures using one-time signature key generation. Before we can sign a message on M, we have to compute the n-bit digest $d = g(M)$. A message of size n is first created by converting a random size message of size m using the hash function. The document's signature will be created by combining the root commitment, one-time signature, one-time verification key, and finally, the proof's index s.

Verkle's signature verification works as follows: the one-time signature of *sign* should be validated with Y_s. If this is true, the VC_i commitments are validated. The signature is confirmed if the root of the tree equals the root commitment. Considering a Verkle tree, the root commitment is d.

9. Experiments

Merkle trees are very fast and have an $O(n)$ computational time. Regretfully, their proof size of $O(\log_2 n)$ is relatively large and can incur significant width costs. The size

of their proofs $O(w\log_w n)$ is actually larger than Merkle trees when using greater width trees (w-ary trees). Utilizing a vector commitment scheme reduces the proof size to a fixed value, $O(1)$; however, the vector commitment construction is very costly and labor-intensive, requiring an $O(n^2)$ calculation.

The w-width Verkle tree requires only $O(wn)$ time for construction. Additionally, compared to the Merkle tree membership proofs, its proof size is only $O(\log_w n)$, which is significantly less than $O(\log_2 w)$. This is a good trade-off. Thus, Table 1 provides the relevant algorithmic schemes comparison.

Table 1. Scheme comparison.

Scheme	Construction	Update	Proof Size
Merkle Tree	$O(n)$	$O(\log_2 n)$	$O(\log_2 n)$
Merkle Tree (w-ary)	$O(n)$	$O(w\log_w n)$	$O(w\log_w n)$
Vector Commitment	$O(n^2)$	$O(n)$	$O(1)$
Verkle Tree	$O(\mathrm{w}n)$	$O(w\log_w n)$	$O(\log_w n)$

There has not been a lot of research performed on post-quantum vector commitment schemes, or ones that might be secure against quantum attacks. You can use Merkle trees that were constructed with a post-quantum hash function, but their updates are relatively costly and inherently stateful. Based on the SIS lattice problem, we only have Merkle tree-like construction that directly produces a stateless updatable VC scheme.

For us, it is important that resulting methods protect us from quantum computer attacks. Quantum computers could break our earlier vector commitments based on RSA. Our signature techniques employ Verkle trees, but we construct vector commitments using lattices. There are other Merkle algorithms, which are post-quantum, such as the Fractal Merkle algorithm [16].

In this case, the classical algorithm results are the following:

Key generation time—0.049351, Signature time—0.0002425, Verification time—0.0038651.

Thread-based algorithm:

Key generation time—0.013841, Signature time—0.0002425, Verification time—0.0038651.

Based on the post-quantum Short Integer Solution lattice problem, we developed a new construction of vector commitment. The protocol enables vector message verification and secure commitment. They start with a stateless updatable "base" VC construction. It is especially suitable for a relatively large d because of the quadratic dependence of the public parameters on d.

We provide a specialized tree transformation (unlike generic Merkle trees) of our SIS-based VC for larger dimensions d^{h} that preserves stateless updates. The proofs of construction are d-factor-concise, as the transformation relies on a VC instead of a hash function.

Our method's primary advantage is its theoretical security against quantum attacks. This has the drawback that commitment and proof sizes in the vector dimension d are logarithmic rather than constant. Despite the inherent trade-off of logarithmic commitment and proof sizes in vector dimension d, our lattice-based construction was rigorously tested against classical algorithms. The results, although slower, showcase a smaller digital signature compared to the Merkle tree version, emphasizing the practical advantages of our proposed scheme.

We tested our algorithm on the same machine, where we tested the digital signature based on the Merkle tree.

We have the following results:

Key generation time—0.049351, Signature time—0.00001520, Verification time—0.00048250.

Our lattice-based construction is of course slower, but in our case the digital signature is much better than the Merkle version.

Our novel vector commitment construction, rooted in the post-quantum Short Integer Solution lattice problem, presents a compelling alternative, especially in scenarios where

quantum security is paramount. The trade-offs in commitment and proof sizes are carefully balanced, and empirical results underscore the practical benefits of our approach.

10. Conclusions

This research explored the present tools that are accessible for both classical and quantum scenarios. Systems for post-quantum cryptography were covered. We covered hash-based one-way functions, their integration into Merkle, and their incorporation into Verkle. Vector commitment and lattice-based commitments were explored. We discussed the computation and integration of the powerful Merkle tree—Verkle tree improvement. The effectiveness of novel shames led to the creation of a new model, and its integration into Verkle. They do require more complicated cryptography to accomplish, but there is a chance for significant scalability advantages.

It is crucial for us that the resulting schemes defend against attacks from traditional and quantum computers. We received the systems that are integrated into Merkle after analyzing the work completed. Although the verification size is too large, Merkle trees, which are built with cryptographic hash functions, provide a strong defense against quantum attacks. Each child in a Merkle tree is represented by the hash of a parent node. A parent node in a Verkle tree is defined as the vector commitment of its children in our improved Verkle tree model. In order to implement the new technology, we discussed vector commitment and commitments based on hard lattice problems.

As a substantial improvement in the Merkle scheme, the Verkle scheme enables much smaller verifications. Rather than presenting all nodes at every level, verification only needs one proof to validate all parent–descendant relationships: all commits from each leaf node to the root. This allows the verification size to be reduced by about 6–8 times when compared to the conventional Merkle approach. Thus, a Verkle tree was considered in place of the Merkle tree in our improved approach. Vector commitment is all that is required in this situation to serve as the proof. To construct the Verkle tree, we relied on vector commitments based on the lattice's assumption.

It is vital for us that the resulting methods protect us from quantum computer attacks. Quantum computers could break our earlier vector commitments based on CDH and RSA. We have now enhanced the plan to make it more secure and efficient. We use lattices to build vector commitments, but our signature methods use Verkle trees. Our systems operate under post-quantum suppositions.

Author Contributions: Conceptualization, M.I.; Formal analysis, M.I. and T.K.; Methodology, T.K. and R.B.; Writing—original draft, T.K.; Writing—review and editing, R.B. All authors have read and agreed to the published version of the manuscript.

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

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest. The funder had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Chen, L.; Chen, L.; Jordan, S.; Liu, Y.K.; Moody, D.; Peralta, R.; Perlner, R.A.; Smith-Tone, D. *Report on Post-Quantum Cryptography*; US Department of Commerce, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2016; Volume 12.
2. Buchmann, J.; Dahmen, E.; Szydlo, M. Hash-based Digital Signature Schemes. In *Post-Quantum Cryptography*; Bernstein, D.J., Buchmann, J., Dahmen, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2009. [CrossRef]
3. Bhaskar, B.; Sendrier, N. McEliece cryptosystem implementation: Theory and practice. In *Post-Quantum Cryptography, Proceedings of the Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, 17–19 October 2008*; Proceedings 2; Springer: Berlin/Heidelberg, Germany, 2008.
4. Yin, X.; He, J.; Guo, Y.; Han, D.; Li, K.-C.; Castiglione, A. An Efficient Two-Factor Authentication Scheme Based on the Merkle Tree. *Sensors* **2020**, *20*, 5735. [CrossRef] [PubMed]

5. Chen, Y.-C.; Chou, Y.-P.; Chou, Y.-C. An Image Authentication Scheme Using Merkle Tree Mechanisms. *Future Internet* **2019**, *11*, 149. [CrossRef]
6. Lamport, L. Constructing Digital Signatures from a One Way Function. 1979. Available online: https://www.microsoft.com/en-us/research/publication/constructing-digital-signatures-one-way-function/ (accessed on 5 December 2023).
7. Iavich, M.; Bocu, R.; Arakelian, A.; Iashvili, G. Post-Quantum Digital Signatures with Attenuated Pulse Generator. Volume 2698. 2020. Available online: https://www.researchgate.net/profile/Maksim-Iavich/publication/346971219_Post-Quantum_Digital_Signatures_with_Attenuated_Pulse_Generator/links/5fd63e2845851553a0b26923/Post-Quantum-Digital-Signatures-with-Attenuated-Pulse-Generator.pdf (accessed on 5 December 2023).
8. Koo, D.; Shin, Y.; Yun, J.; Hur, J. Improving Security and Reliability in Merkle Tree-Based Online Data Authentication with Leakage Resilience. *Appl. Sci.* **2018**, *8*, 2532. [CrossRef]
9. Sim, M.; Eum, S.; Song, G.; Yang, Y.; Kim, W.; Seo, H. K-XMSS and K-SPHINCS+: Enhancing Security in Next-Generation Mobile Communication and Internet Systems with Hash Based Signatures Using Korean Cryptography Algorithms. *Sensors* **2023**, *23*, 7558. [CrossRef] [PubMed]
10. Merkle, R.C. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology—CRYPTO '87. CRYPTO 1987*; Pomerance, C., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1988; Volume 293. [CrossRef]
11. Chen, H.; Liang, D. Adaptive Spatio-Temporal Query Strategies in Blockchain. *ISPRS Int. J. Geo-Inf.* **2022**, *11*, 409. [CrossRef]
12. Wang, W.; Ulichney, A.; Papamanthou, C. BalanceProofs: Maintainable vector commitments with fast aggregation. In Proceedings of the 32nd USENIX Conference on Security Symposium (SEC '23), Berkeley, CA, USA, 9–11 August 2023; USENIX Association: Berkeley, CA, USA, 2023. Article 247. pp. 4409–4426.
13. Kurosawa; Kaoru; Hanaoka, G. (Eds.) Public-Key Cryptography—PKC 2013. In Proceedings of the 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, 26 Feburary–1 March 2013; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7778.
14. John, K. Verkle Trees. 2019. Available online: https://math.mit.edu/research/highschool/primes/materials/2018/Kuszmaul.pdf (accessed on 5 December 2023).
15. Papamanthou, C.; Shi, E.; Tamassia, R.; Yi, K. Streaming authenticated data structures. In *Advances in Cryptology–EUROCRYPT 2013, Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, 26–30 May 2013*; Proceedings 32; Springer: Berlin/Heidelberg, Germany, 2013.
16. Iavich, M.; Gnatyuk, S.; Arakelian, A.; Iashvili, G.; Polishchuk, Y.; Prysiazhnyy, D. Improved Post-quantum Merkle Algorithm Based on Threads. In *Advances in Computer Science for Engineering and Education III 3*; Springer International Publishing: Berlin/Heidelberg, Germany, 2021; pp. 454–464.