IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.Doi Number

Fractional Verkle Trees for Scalable Post-Quantum Signatures

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

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

Corresponding author: Maksim Iavich (e-mail: miavich@cu.edu.ge).

ABSTRACT Quantum computers threaten to break widely used cryptosystems like RSA and ECC, forcing us to develop new quantum-resistant alternatives. While hash-based signatures like SPHINCS+ offer a proven solution, their reliance on Merkle trees result in large signature sizes and memory-intensive verification, which hinders scalability. We present a novel approach that replaces Merkle trees with Fractional Verkle Trees (FVTs), a hierarchical structure where a large Verkle tree is decomposed into smaller, more manageable pieces, which we call hypertrees. This approach provides fast verification and smaller signatures, while preserving the required level of security. Our FVT construction offers a useful basis for scalable verifiable data structures while lowering memory and verification costs without sacrificing security. Validation by way of a functional prototype shows that the derived Fractal Verkle-Based Digital Signature (FVDS) scheme achieves signature sizes of about 16.2 KB, which is 2.5× smaller than SPHINCS+, while maintaining localized updates and scalable verification. This makes it especially appropriate for bandwidth-constrained applications, such as IoT firmware updates and blockchain light clients.

INDEX TERMS Cryptographical application, fractional Verkle Trees, hash-based digital signatures, HORS, HORST, Merkle tree, quantum cryptography, Verkle tree, Verkle-based HORST

I. INTRODUCTION

The advent of quantum computing has exposed fundamental vulnerabilities in classical public-key cryptosystems, particularly those based on integer factorization and discrete logarithm problems. As Shor's algorithm demonstrates the ability to break RSA and ECC in polynomial time, the cryptographic community has intensified efforts to develop post-quantum secure alternatives [1,2]. Among the various approaches being standardized, hash-based signatures have emerged as particularly promising due to their minimal security assumptions and resistance to quantum attacks [3].

Placing our hash-based method in the context of the varied post-quantum cryptography environment is crucial. Dilithium and Falcon, two NIST-standardized lattice-based schemes, are great for general-purpose use because they provide small signatures and extremely quick verification. The conservative, minimal assumption of cryptographic hash function robustness serves as the foundation for the security of hash-based signatures, such as our FVDS. Benefits of this design include the ability to build intricate, scalable structures like the FVT and accurate security quantification against quantum attacks. Consequently, FVDS is a high-assurance, scalable substitute in the hash-based family that is best suited for applications where the security assumption's simplicity and robustness are crucial rather than Dilithium's direct rival.

These schemes derive their security solely from cryptographic hash functions, which are believed to remain secure even against quantum adversaries, with Grover's algorithm providing at most a quadratic speedup that can be mitigated by increasing hash sizes [4]. Fractal Verkle-based designs offer a unique balance between practical efficiency and quantum resistance, and recent advancements that integrate these hash-based techniques with complex tree layouts have shown particular promise [5].

Traditional hash-based signature schemes face significant practical limitations that hinder widespread adoption. The Merkle Signature Scheme [6], while providing a theoretically sound framework for multi-signature generation, suffers from statefulness requirements and growing proof sizes that scale logarithmically with the number of signatures [7]. Subsequent improvements, such as HORST [8], attempted to address these issues through tree-based key compression, yet still inheriting

similar inefficiencies, which are characteristic to Merkle tree constructions [9]. These limitations encourage the development of hybrid approaches that combine the security of hash-based encryption with the efficiency of modern verifiable data structures [10]. These limitations become particularly problematic in modern applications requiring high-throughput signing operations or deployment in resource-constrained environments, creating a pressing need for more scalable alternatives.

Recent advances in cryptographic accumulator designs present new opportunities for optimizing post-quantum signatures. Verkle trees [11], employing polynomial commitment schemes, offer a paradigm shift from traditional Merkle structures by enabling constant-sized proofs regardless of the implied tree size [12]. This fundamental improvement comes from their ability to leverage advanced cryptographic techniques like post-quantum vector commitments, which allow for efficient verification without compromising security [13]. When constructed in fractal (hierarchical) forms, these trees enable novel digital signature systems with exceptional scalability. Concurrently, developments in lattice-based cryptography have yielded new vector commitment schemes with attractive post-quantum security properties [14]. These developments based on lattice technology offer encouraging synergies with our FVT methodology that should be investigated further, especially in order to improve the vector commitment layer's efficiency and security assurances. However, despite these individual advancements, the potential synergies between these approaches remain largely unexplored in the context of practical signature schemes.

This work presents Fractional Verkle Trees (FVTs), a novel approach to rethink scalable cryptographic accumulators through hierarchical decomposition. By splitting a monolithic Verkle tree into interdependent subtrees, we achieve notable gains in verification performance and storage efficiency while maintaining the security features of vector commitments. Building on this foundation, we introduce a fractal Verkle-based digital signature method that achieves post-quantum security with much improved operational characteristics.

Crucially, our Fractional Verkle Trees process differs from stateless hash-based systems such as SPHINCS+ by radically changing the underlying data structure. To accomplish constant-sized proofs for every subtree layer, FVTs employ vector commitments, but SPHINCS+ uses a hypertree of Merkle trees, inheriting their logarithmic proof sizes [15]. In Section IX, we show that this architectural change is essential to achieve the notable reductions in verification memory and signature size.

By demonstrating how layered architectures may get around scaling obstacles in verifiable data structures, the approach closes the gap between theoretical promise and practical execution.

Fractional Verkle Trees provide new opportunities for systems that need post-quantum security and excellent performance. The derived signature method maintains all of these advantages while also including the crucial authentication and non-repudiation capabilities required for real-world uses. For blockchains that need compact proofs and edge devices with limited memory, our system enables modifiable trade-offs between resource overhead and operational efficiency. Through formal analysis and practical assessment, we show that FVTs offer a solid foundation for the next generation of cryptographic infrastructures, whose scalability is independent of expensive or compromised security.

Table 1 offers a high-level comparison of important features that clearly demonstrate the contribution of our work versus the most relevant modern systems.

TABLE 1
COMPARISON OF POST-QUANTUM SIGNATURE SCHEMES:

Characteristic	SPHINCS+ (Stateles)	Monolithic Verkle Tree	Our FVDS Scheme
Core Structure	Hypertree of Merkle Trees	Single Verkle Tree	Hierarchy of Verkle Trees (FVT)
Signature Size	Large (~ 41 KB for 128-bit security)	Medium (Scales with tree height h)	Small (~2.4 KB for our parameters)
Verification Memory	Moderate	High (full path needed)	Low (only active subtrees cached)
Key Update Overhead	High (global)	Prohibitive (full tree)	Low (localized to one subtree)
Post Quantum Security	Yes (Hash-based)	Yes (VCs)	Yes (Hash and VC-based)
Primary Advantage	Fully Stateless	Constant-size proofs	Scalability and Parallelization

II. HASH-BASED DIGITAL SIGNATURES

Due to the properties of cryptographic hash functions, hash-based digital signatures have emerged as a preeminent choice for post-quantum cryptography because of its security guarantees. Unlike public-key cryptography such as RSA and ECC which rely on quantum-vulnerable mathematical problems, the security of hash-based signatures stems from the fact that they are one-way and collision-resistant [16]. Thus, these signatures are immune to the dangers of quantum computing.

Real-world applicability of hash-based signatures stems from a few distinctive advantages. These include the elegance of their design and their proven dependability. Many of these systems are built upon simple one-time signature methods such as Lamport-Diffie or Winternitz schemes [17].

The development of early hash-based algorithms was impeded by difficulty retaining state and managing large key sizes. More recent works, like the Merkle Signature Scheme (MSS) and HORST (Hash to Get Random Subset with Trees), have resolved these with stateless approaches and tree-based key management enabling these systems to be employed in practical settings like blockchains and secure communication systems [18].

The suggested FVDS approach is based on proven hash-based signature primitives. A brief summary of these elements is given in the ensuing subsections, with particular attention paid to the characteristics that are most pertinent to our construction: their one-time nature, their dependence on hash functions for security, and their common structure of having big public keys that are advantageous for tree-based authentication. This background is necessary to comprehend the design decisions made in FVDS.

A. LAMPORT-DIFFIE ONE-TIME SIGNATURE SCHEME

The development of modern hash-based signatures traces its origins to Leslie Lamport's seminal 1979 work introducing one-time signature schemes (LD-OTS) [19]. This novel strategy depended exclusively on cryptographic hash capacities instead of complicated scientific assumptions.

The Lamport-Diffie one-time signature (LD-OTS) is a basic hash-based method designed to resist quantum attacks [20]. Its structure primarily relies on cryptographic hash algorithms and randomization for key creation, signing, and verification.

The private key consists of two lists, sk_0 and sk_1, each containing 256 randomly generated values. First list is:

$$sk_0 = (sk0_1, sk0_2, sk0_3, \dots, sk0_{256}) \quad (1)$$

There is an additional second list:

$$sk_1 = (sk1_1, sk1_2, sk1_3, \dots, sk1_{256}) \quad (2)$$

The corresponding public key is derived by applying a cryptographic hash function H to each element. The hashes of the sk_0 values are in the first list, pk_0:

$$pk_0 = \left(H(sk0_1), H(sk0_2), H(sk0_3), \dots, H(sk0_{256})\right) \quad (3)$$

The second list pk_1, contains the hashes of the sk_1values:

$$pk_1 = \left(H(sk1_1), H(sk1_2), H(sk1_3), \dots, H(sk1_{256})\right) \quad (4)$$

The public key is then the concatenation of these hashed values.

Before signing, the message must be encoded as a fixed-length 256-bit binary string. If the original message is not in binary form, a cryptographic hash function is used to produce a digest of the required length. Each bit of the message determines which component of the private key is revealed: if the i-th bit is 0, the signer discloses $sk_{0,i}$ if the bit is 1, then $sk_{1,i}$ is used instead. The resulting signature is a sequence of 256 private key elements corresponding to the message bits. A 256-bit message is produced: $\mathrm{msg} = (\mathrm{msg}_1, \mathrm{msg}_2, \dots, \mathrm{msg}_{256})$.

The recipient verifies a signature by comparing the hash of each revealed element with the matching entry in the public key. In this case our signature is: $\mathrm{signature} = (\mathrm{sig}_1, \mathrm{sig}_2, \dots, \mathrm{sig}_{256})$. For this, the verifier calculates $H(sig_i)$ for each i-th signature component sig_i and compares it to either $\mathrm{pk}\ 0_i$ (if the message bit was 0) or $\mathrm{pk}\ 1_i$ (if the bit was 1). The signature is legitimate if all 256 hashes match the anticipated values of the public key.

The drawbacks of the Lamport-Diffie one-time signature safe hash-based signature method are its large key size, vulnerability to quantum attacks, and need for 512 hash outputs for the public key and 256 random values per signature. Reusing a secret key also compromises security as it enables attackers to fake signatures from several observations. To solve these issues, Merkle developed a signature mechanism that authenticates multiple one-time public keys using a Merkle tree.

B. WINTERNITZ ONE-TIME SIGNATURE (WOTS)

The Winternitz One-Time Signature (WOTS) method is an efficient hash-based signature mechanism that overcomes the computational and key storage limitations of the Lamport–Diffie OTS. WOTS reduces signature and key sizes while preserving security by grouping message bits and employing iterative hashing. However, it is still a one-time signature scheme.

Based on a cryptographic hash function H that produces outputs of fixed length, WOTS incorporates an adjustable parameter w (the Winternitz parameter) to decide how many message bits are processed collectively. A bigger w reduces the size of the signature, but at the cost of additional hash computations [21]. First, the signer generates l secret key components at random, where l is defined by w and the message digest length. The public key is obtained by repeatedly applying H to each constituent of the secret key $w - 1$ times:

$$pk_i = H^{w-1}(sk_i) \text{ for } i = 1,2, \dots, l. \quad (5)$$

Here, H^{w-1} denotes $w - 1$ successive applications of H (e.g., $H^2(x) = H(H(x))$).

Prior to being signed, a message is first hashed into a fixed-length digest, which is then split into l pieces in base-w format. Each portion, $\mathrm{msgPart}_i$, determines how many times the corresponding secret key element sk_i is hashed to create the signature component:

$$\mathrm{sig}_i = H^{\mathrm{msgPart}_i}(sk_i) \quad (6)$$

The final signature is created by concatenating all of the elements.

Verification requires computing the public key from the signature. The following is calculated by the verifier:

$$\mathrm{computedPk}_i = H^{w-1-\mathrm{msgPart}_i}(\mathrm{sig}_i) \quad (7)$$

If the computed public key matches the original public key, the signature is considered authentic.

C. WINTERNITZ ONE-TIME SIGNATURE PLUS (WOTS+)

Winternitz One-Time Signature Plus (WOTS+) is an enhanced version of the WOTS scheme that improves security while maintaining the core principles of hash-based signatures. By using bitmasks and a modified hash chain design, WOTS+ mitigates the collision and second-preimage attacks that are present in the iterative hashing of WOTS [22].

Similar to WOTS, WOTS+ depends on a Winternitz parameter w and a cryptographic hash function H (such as SHA-256). The secret key, $\mathrm{sk} = (sk_1, sk_2, \dots, sk_l)$, is made up of l randomly generated values. But WOTS+ adds a crucial innovation: before hashing, each secret key element is combined with a random bitmask r_i. This is how the public key is calculated:

$$pk_i = H^{w-1}(sk_i \oplus r_i) \text{ for } i = 1,2,\dots,l, \quad (8)$$

Before a message can be signed, it must first be hashed to provide a fixed-length digest. The digest is then converted to base- w notation, producing l sections: $\mathrm{msgDigest} = (\mathrm{msgPart}_1, \mathrm{msgPart}_2, \dots, \mathrm{msgPart}_l)$.

The matching secret key element sk_i is XOR-ed with the bitmask r_i for each component $\mathrm{msgPart}_i$, and $\mathrm{msgPart}_i$ is then hashed once:

$$\mathrm{sig}_i = H^{\mathrm{msgPart}_i}(\mathrm{sk}_i \oplus r_i) \quad (9)$$

These l values combine to generate the final signature: $\mathrm{signature} = (\mathrm{sig}_1, \mathrm{sig}_2, \dots, \mathrm{sig}_l)$.

The recipient hashes each sig_i and reapplies the bitmask to reconstruct the public key during verification:

$$\mathrm{computedP}_i = H^{w-1-\mathrm{msgPart}_i}(\mathrm{sig}_i) \oplus r_i \quad (10)$$

The signature is legitimate if the calculated public key corresponds to the original pk. This procedure keeps the private key hidden while guaranteeing integrity.

WOTS+ addresses two of the original WOTS scheme's main shortcomings. First, it increases predictability resistance by randomizing hash chains using bitmasks. Second, WOTS+ improves performance and enables safer reuse in multi-signature frameworks such as the Extended Merkle Signature Scheme (XMSS) by lowering substantial exposure concerns.

WOTS+ maintains its essential characteristics as a one-time signature system in spite of these developments. Its combination with Merkle trees makes it possible to create scalable and quantum-resistant digital signatures that are appropriate for real-world applications, especially in systems like XMSS.

III. MERKLE SIGNATURE SCHEME (MSS)

The Merkle Signature Scheme (MSS), developed by Ralph Merkle in 1979, effectively addresses a fundamental flaw in one-time signature (OTS). MSS organizes many OTS key pairs into a binary Merkle tree structure, where the root of the tree is the master public key [23]. MSS impacted several modern hash-based signature schemes, such as XMSS and SPHINCS+, which are now included in the NIST Post-Quantum Cryptography Standardization Project [24].

The three primary parts of the Merkle signature scheme's key generation process is determining the main public key, calculating a Merkle tree, and creating one-time signature key pairs.

A Merkle tree is shown in the following figure when its height is three:

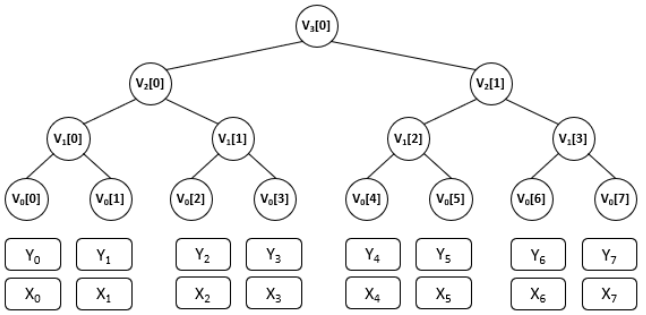

FIGURE 1. Merkle tree: height – three.

The MSS key generation process begins with the creation of 2^h one-time signature key pairs (X_j, Y_j), where h is the height of the Merkle tree. Each key pair consists of a private signing key X_j and a corresponding public verification key Y_j. Key pair is $(\mathrm{X}_i, \mathrm{Y}_i)$ for $i = 0,1,\dots,2^h - 1$. The leaf nodes of the Merkle tree are then computed by hashing each OTS public key, such that $\mathrm{leaf}_i = H(\mathrm{y}_i)$ for $i = 0,1,\dots,2^h - 1$.

Pairs of child nodes are hashed iteratively to generate successive internal nodes. In particular, a parent node at level j is generated as $\mathrm{node}_{j,k} = H(\mathrm{node}_{j-1,2k} \| \mathrm{node}_{j-1,2k+1})$, where $\|$ denotes concatenation. This process continues until a single root node $\mathrm{node}_{h,0}$ is obtained, which becomes the master public key $\mathrm{pk}_{\mathrm{MSS}}$. The private key consists of all one-time signature secret keys, enabling the signing of up to 2^h distinct messages.

To sign a message, the signer selects an unused OTS key pair at index i and computes the signature:

$$\mathrm{sigOTS} = \mathrm{SignOTS}(\mathrm{message}, \mathrm{X}_i) \quad (11)$$

In addition to this, the signer must provide an authentication path, which consists of the sibling nodes required to reconstruct the Merkle root. The sister nodes required for the verifier to reconstruct the root without being aware of the entire tree are present in this route. The authentication path is defined as:

$$\mathrm{authPath} = (\mathrm{node}_{0,s_0}, \mathrm{node}_{1,s_1}, \dots, \mathrm{node}_{h-1,s_{h-1}}) \quad (12)$$

where each $\mathrm{node}_{\mathrm{j},s_\mathrm{j}}$, is necessary to compute the parent node at level $j+1$. s_j represents the sibling node required to compute the parent node at level j. Finally, the full MSS signature consists of the OTS signature and the authentication path: $\mathrm{signature} = (\mathrm{sigOTS}, \mathrm{authPath})$.

Verification involves two primary steps. First, the OTS signature is validated using the corresponding public key Y_i ($\mathrm{VerifyOTS}(\mathrm{message}, \mathrm{sigOTS}, \mathrm{y}_i)$). If successful, the verifier iteratively hashes the nodes in the supplied authentication path to recreate the Merkle root.

$$\text{node}_{j+1} = H(\text{node}_j \| \text{authPath}_j) \quad (13)$$

Only when the calculated root corresponds to the trusted master public key the signature is accepted.

As opposed to traditional systems like RSA and ECC, MSS is safe because the underlying hash function is collision resistant, making it immune to quantum assaults [25]. However, because MSS is stateful, the signer must maintain a record of which OTS keys have been used to prevent repeat attacks. Moreover, it can be computationally expensive to generate and validate Merkle tree hashes, particularly for big h.

By validating several OTS public keys using a single Merkle root, the Merkle Signature Scheme (MSS), a fundamental technique, resolves the one-time key issue. It creates the fundamental tree-based signature paradigm that FVDS improves. Understanding MSS is essential for placing the development of the discipline and the issues of proof size and statefulness that our work tackles in context, even if FVDS makes use of the more effective Verkle trees.

IV. HORS AND VARIANTS

A. HORS: HASH TO OBTAIN A RANDOM SUBSET SIGNATURE SCHEME

Unlike more traditional number-theoretic methods like RSA or elliptic curve encryption, HORS is not vulnerable to quantum computing assaults since it just employs cryptographic hash functions [26].

The HORS scheme is driven by two fundamental parameters: t, which represents the total number of components in the secret key, and k, which represents the number of pieces revealed for each signature. The signer creates a private key with t random values as the first stage in key generation: sk = (sk_1, sk_2, ..., sk_t). Every secret component can be subjected to a cryptographic hash function H, which produces the associated public key pk = (H(sk_1), H(sk_2), ..., H(sk_t)).

The choice of parameters t and k presents a substantial trade-off between security and performance. In practice, these parameters must be carefully balanced based on the required security levels and the anticipated number of signatures.

Calculating a message digest using cryptographic hash function, msgDigest $= H(\text{message})$, is the first stage in the signature generation. Next is to divide this digest into k index values, wherein each component is transformed into an index value that chooses one of the elements of the secret key.

The computation of these indexes: $(\text{index}_1, \text{index}_2, \ldots, \text{index}_k) = \text{msgDigest} \bmod t$. The signature then contains the secret key elements that match these indices: signature is equal to signature $= (\text{sk}_{\text{index}_1}, \text{sk}_{\text{index}_2}, \ldots, \text{sk}_{\text{index}_k})$.

Instead of being a fully reusable signature scheme, HORS is classified as a few-times signature system.

For each signature component sig_i, the verifier checks if $H(\text{sig}_i)$ matches the corresponding public key element:

$$H(\text{sig}_i) = \text{pk}_{\text{index}_i} \; \forall i \in [1, k] \quad (14)$$

The signature is only legitimate if all k hash comparisons are successful in order to guarantee the message's integrity and validity.

HORS provides notable efficiency gains by needing just k secret components per signature rather than complete key disclosure. Nevertheless, because each signature only offers a subset of the secret key's information, this efficacy has fundamental security flaws. After around t/k signatures, an attacker might reassemble enough important information to fake signatures [27].

B. HORST: HORS WITH TREES SIGNATURE SCHEME

By addressing significant security vulnerabilities in the basic Hash to Obtain Random Subset (HORS) approach, the HORS with Trees (HORST) signature scheme significantly improves few-time signature (FTS) systems [28].

A private key with t random values is generated at the beginning of the HORST scheme, just like in HORS: $\text{sk} = (\text{sk}_1, \text{sk}_2, \ldots, \text{sk}_t)$. Each leaf node is generated by hashing a private key component, according to Formula 15:

$$\text{leaf}_i = \text{H}(\text{sk}_i) \; \forall i \in [1, t] \quad (15)$$

The tree is built iteratively by pairwise hashing of child nodes, according to Equation:

$$\text{node}_{j+1} = \text{H}(\text{node}_j \| \text{node}_{j+1}) \quad (16)$$

where ‖ denotes concatenation.

The HORS technique is enhanced with crucial authentication enhancements by the HORST signature process. After calculating the message indexes: $(\text{index}_1, \text{index}_2, \ldots, \text{index}_k) = \text{Hash}(\text{message}) \bmod t$, the signature includes the exposed secret components (Formula 17) and the authentication path required for Merkle root reconstruction:

$$\text{sigHORS} = (\text{sk}_{\text{index}_1}, \text{sk}_{\text{index}_2}, \ldots, \text{sk}_{\text{index}_k}) \quad (17)$$

It is necessary for the sister nodes on this route $\text{authPath} = (\text{node}_1, \text{node}_2, \ldots, \text{node}_h)$ to verify that the revealed leaves were a part of the original tree.

The verification procedure consists of two steps: By looking at hash correspondences between leaf nodes and disclosed secret elements, the verifier first confirms the HORS components. Second, the authentication path allows Merkle root reconstruction through iterative hashing (Formula 18):

$$\text{node}_{j+1} = H(\text{node}_j \| \text{authPath}_j) \quad (18)$$

Because of its Merkle tree integration, HORST provides a number of security advantages over HORS. Even if several signatures partly reveal the key, an attacker cannot generate authentic signatures without endangering the entire Merkle tree structure. This capability makes HORST particularly helpful for applications that require limited but safe signatures, such as secure boot mechanisms and IoT firmware upgrades.

At the lowest layer, our FVDS method makes use of the HORST few-time signature mechanism. The HORS primitive and its development into HORST, which employs a Merkle

tree for public key authentication, are covered in detail in this section. We later modified the idea of utilizing a tree to authenticate a fragmented public key by substituting our more effective Verkle trees for Merkle trees.

V. VERKLE TREE

The introduction of Verkle trees in cryptographic commitment systems is significant because they offer substantial advantages over traditional Merkle trees in terms of verification efficiency and storage requirements. As post-quantum cryptography gains popularity, data structures that optimize computing resources while ensuring high security are becoming more and more crucial. This need is satisfied by verkle trees, which employ vector commitments instead of conventional hash functions, resulting in shorter proofs and a reduced verification complexity [29]. In this work we explore the structural and functional advantages of verkle trees, emphasizing their potential use in large-scale cryptography systems.

The data items must be arranged into size k subsets before a Verkle tree can be constructed. One computes a vector commitment (VC) for each subset of files f_0, f_1, ..., f_n. These promises are the core components of the tree structure. After generating the membership proofs (PR_i) for each element with respect to its corresponding VC, higher-level commitments are calculated iteratively until only one root commitment remains.

Figure 2 illustrates a Verkle tree with nine files and a branching factor of k = 3. Construction starts by calculating the commitments (VC_1, VC_2, and VC_3) of each file triplet and the associated membership proofs. These commitments are then used as inputs for the next stage, which computes the root commitment, or VC_4. The hierarchical structure of this structure ensures that the final root commitment serves as a digested version of the whole dataset while retaining the ability to efficiently verify each element.

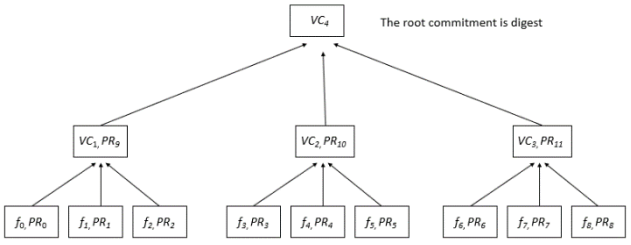

FIGURE 2. **Verkle tree, with branching factor 3.**

Verification procedures for Verkle trees and Merkle trees differ greatly. For proof creation in a Merkle tree structure, all sibling nodes must be included along the authentication path from the target leaf to the root [30]. This characteristic causes proof sizes to grow logarithmically with the number of tree elements. In contrast, vector commitments are used by verkle trees to enable the concurrent batched verification of several pathways. This approach significantly reduces the computational complexity of verification as well as the size of the proofs.

The efficiency increases are caused by two key Verkle tree properties. First, vector commitments may be used to merge many verification stages into a single operation. Second, the branching form of the tree enables more comprehensive coverage of the dataset with fewer proof parts. These features make Verkle trees particularly well-suited for applications requiring large datasets where storage efficiency and verification speed are critical considerations.

From a security perspective, Verkle trees provide the same collision resistance guarantees as Merkle trees when built with the appropriate cryptographic primitives. The vector commitments used in Verkle trees can be constructed using a range of cryptographic assumptions, including those based on elliptic curve pairings or lattice-based encryption, to guarantee compliance with post-quantum security standards [31].

The performance advantages of verkle trees are particularly apparent when frequent verification processes are required. Less bandwidth is required for systems where proofs must be transmitted using networks since the proof sizes are less. Because verkle trees can verify several pathways simultaneously, they reduce the computational strain on verification nodes, making them an attractive option for distributed systems and blockchain applications.

A. VERKLE-BASED HORST

Verkle trees are essential for post-quantum cryptography because they reduce storage requirements, maintain high security, and eliminate redundant data for efficient verification processes that keep only the information that is needed, making them perfect for handling large datasets. Verkle trees, in contrast to conventional hash functions, accomplish these gains by utilizing polynomial commitments, such as lattice-based vector commitments [32].

A Verkle tree is used in place of the Merkle tree for validating HORS public keys in the Verkle-based HORST (HORS with Trees) for authenticating public keys. Applications like blockchain systems, secure boot protocols, and IoT firmware upgrades that demand concise proofs and effective verification are especially well-suited for Verkle-based HORST.

The secret key for Verkle-based HORST is made up of t random values $\mathrm{sk} = (\mathrm{sk}_1, \mathrm{sk}_2, \ldots, \mathrm{sk}_t)$. The Verkle tree's leaf nodes are created by hashing each secret key value $\mathrm{leaf}_i = H(\mathrm{sk}_i)$ for $i = 1,2,\ldots,t$. By iteratively applying vector commitments to collections of leaf nodes, we may create a Verkle tree that has a single root node $pkVerkleHORST = root$.

Figure 3 shows the structure of Verkle-based HORST.

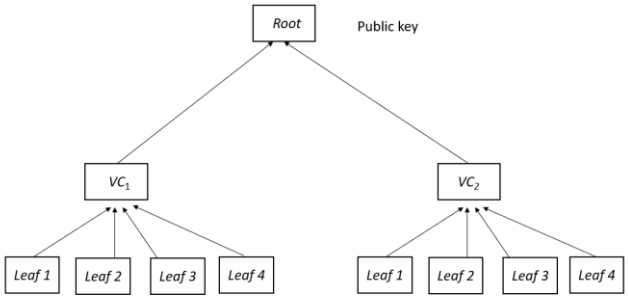

FIGURE 3. Hierarchical diagram for Verkle-based HORST.

As illustrated in Figure 3, the following is a description of the Verkle-based HORST structure:

1. Leaf nodes

Leaf nodes represent the hashed secret keys of the HORST scheme.

Generate t secret keys and compute their leaves:

$$\text{leaf}_i = H(\text{sk}_i) \text{ for } i = 1,2,\dots,t \quad (19)$$

These leaves form the lowest layer of the Verkle tree.

2. Intermediate Layers (Optional)

Group leaves/nodes into vectors:

Verkle trees use a branching factor m (e.g., $m = 256$) to minimize tree depth.

For example, if $t = 1024$, group leaves into 4 vectors of 256 leaves each:

$$\text{node}_{1,k} = \text{Commit}\left(\text{leaf}_{(k-1)m+1},\dots,\text{leaf}_{k\cdot m}\right) \text{ for } k=1,2,3,4 \quad (20)$$

Intermediate layers are only required for large t. If $t \leq m$, the root can directly commit to all leaves.

3. Root Layer

The root is the final commitment of the Verkle tree: $\text{pkVerkleHORST} = \text{Commit}(\text{node}_1, \text{node}_2, \dots, \text{node}_n)$. n depends on the branching factor m and the number of leaves t.

Verkle-based HORST shows how vector commitments can simplify signature systems by substituting more compact proofs for Merkle trees. This strategy strikes a balance between security, effectiveness, and scalability, these qualities are essential for contemporary applications like blockchain and the Internet of Things. Its applicability to increasingly larger-scale systems may be further improved by future refinements.

VI. FRACTIONAL VERKLE TREES

The importance of Merkle and Verkle trees in current cryptographic systems is growing due to their ability to provide compact proofs. However, when applied in large-scale environments, conventional monolithic structures have serious scalability problems. Some of the primary challenges that these trees encounter as they grow to accommodate massive datasets, like global public key infrastructures or blockchain state histories, include the requirement for full-path re-computation on updates (imposing O(h) overhead), linear verification latency related to tree height, and excessive memory demands for keeping tree information in active storage.

By employing an advanced hierarchical decomposition, Fractional Verkle Trees (FVTs) can tackle these issues by splitting the structure into d linked subtrees of height h/d. This approach has several ground-breaking advantages: localized updates that change just affected subtrees, parallel processing capabilities through separate subtree operations, and adaptive verification that only looks at relevant pathways. Additionally, FVTs maintain the core benefits of traditional Verkle trees, including post-quantum security and constant-sized proofs per layer, while also greatly improving practical efficiency.

A monolithic Merkle tree may be divided into smaller subtrees using fractional trees, which were initially presented as Fractal Merkle Trees in earlier publications and subsequently refined in SPHINCS. This allows for scalable and efficient key management in hash-based signature schemes [33]. We apply this concept to Verkle trees, which are vector commitment-based cryptographic structures, and present the Fractionated Verkle Tree (FVT). By committing to subtree roots layer by layer and organizing Verkle subtrees hierarchically, FVT achieves proof sizes of $O(1)$ per layer while keeping the post-quantum guarantees of security of the underlying hash-based signature schemes.

For an FVT of total height h, a branching factor m (e.g. $m = 256$) divides it into d layers of Verkle subtrees, each of height h/d. Each subtree uses vector commitments, like lattice-based vector commitments, to commit to its child nodes. The single root in the highest subtree (Layer $d-1$) is the master public key.

- Layer $d-1$: A single Verkle tree whose leaves are roots of Verkle subtrees from Layer $d-2$.
- Layer 0: Every Verkle subtree commits to the public keys of WOTS+ one-time signatures in order to authenticate HORST few-time signature trees.
- HORST Layer: The message is signed by a HORST key, whose root is confirmed by WOTS+, and it propagates upward through the Verkle levels.

A deterministic key is generated using a pseudorandom function (PRF), and each node is addressed by a layer, subtree index, node index.

There is a trade-off between memory, computation, and signature size when selecting parameters. In order to offer a balanced design for general-purpose application, the numbers h = 64, m = 256, and d = 4 were chosen. With this setup, subtrees of height 16 are produced, which is a reasonable size that enables each to be quickly accessed within a CPU cache. Although a bigger d would further minimize memory use, the requirement for more proofs would result in a linear rise in signature size and verification time. A smaller d, on the other hand, reduces the number of proofs but exponentially raises the amount of memory needed to hold each bigger subtree. Because of its adaptable parameterization, FVDS may be tailored to particular settings, such as selecting a lower d for

high-throughput server applications or a larger d for IoT devices with limited memory.

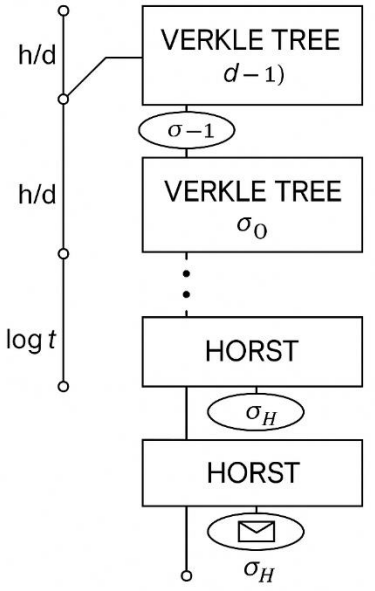

FIGURE 4. Ilustration of the hierarchical layout of a Fractional Verkle Trees

Figure 5 shows the d-layer decomposition and the authentication flow from message signing to the root commitment, as well as the hierarchical structure of Fractional Verkle Trees.

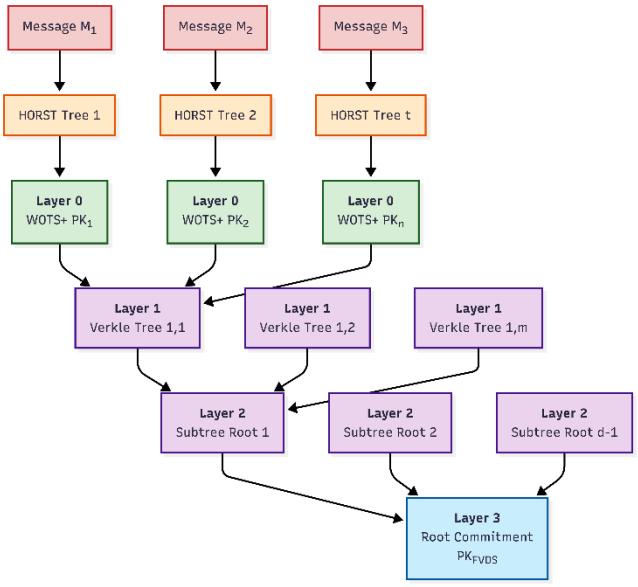

FIGURE 5. Ilustration of the hierarchical layout of a Fractional Verkle Trees

The FVT structure comprises:

- Layer 3 (Root): A single Verkle tree that acts as the master public key PK_{FVDS} and commits to all Layer 2 subtree roots
- Layer 2 (Subtree Roots): $d-1$ Verkle trees that oversee intermediary authentication routes.
- Layer 1 (Intermediate Subtrees): The bottom layer is organized by several Verkle trees with branching factor m.
- Layer 0 (WOTS+ Authentication): For HORST tree authentication, Verkle trees commit to WOTS+ public keys.
- HORST Layer: HORST public keys for message signing are authenticated by WOTS+ signatures.
- Message Layer: Individual messages are given few-time signatures by HORST trees.

A hierarchical chain of authentication is formed from individual messages to the root commitment, with each arrow signifying a vector commitment between tree layers. The underlying hash-based primitives' post-quantum security is preserved while $O(1)$ proof sizes per layer are made possible by this decomposition.

A fractionated Verkle tree's hierarchical structure for hybrid hash-based signing. Every layer has a Verkle tree, and the message is signed by the lowest subtree, which authenticates a WOTS+ signature over a HORST root.

In order to sign a message M :

1. To obtain a pseudorandom index for a HORST leaf, hash M. This is HORST Key Selection.
2. Use the selected HORST key to sign M, to create σ_H.
3. Use the matching WOTS+ key to sign the HORST root, resulting in $\sigma_{W,0}$.
4. Verkle Proofs: Create a constant-size proof π_j for each layer $j = 0$ to $d-1$, that verifies the child commitment in the parent Verkle node.

The final signature:

$$\Sigma = \left(\sigma_H, \sigma_{W,0}, \pi_0, \pi_1, \ldots, \pi_{d-1}\right) \quad (20)$$

Where, each π_j is a vector commitment opening of size $O(1)$.

FVT relies on several cryptographic properties to keep its security. It initially depends on the collision resistance of the underlying hash function, such as SHA-256, to guard against forgeries and ensure data integrity. Second, it avoids equivocation by utilizing the binding property of vector commitments, which are often realized under the discrete logarithm assumption with schemes like a lattice-based VC scheme. Third, addresses created from pseudorandom functions (PRFs) are employed to offer index resilience and defend against reuse attacks. Therefore, for a forgery attempt to succeed, one of the essential components would have to be broken: either WOTS+ would be compromised, a hash collision would occur in HORST, or the binding property of the polynomial commitment scheme would be destroyed.

TABLE 2
THEORETICAL COMPLEXITY TABLE:

Parameter	Complexity	Description

Key Generation	$O(d \cdot m^{h/d})$	Cost to generate all necessary subtrees from seeds, where each subtree requires $m^{h/d}$ operations
Signature Size	$O(k + d \cdot \|\|\pi\|\|)$	Includes both the HORST signature components (k elements) and Verkle proofs (d proofs of size $\|\|\pi\|\|$ each)
Verification	$O(d \cdot T_{VC})$	Requires one vector commitment verification per layer, where T_{VC} is the commitment-specific verification time
Storage	$O(d \cdot m^{h/d})$	Memory needed to maintain active subtrees during operation (cold subtrees can be regenerated)

The efficiency profile of fractional verkle trees (FVTs) demonstrates a well-balanced trade-off between computational overhead and storage requirements. Unlike monolithic Verkle trees that increase exponentially with overall height h, the fractional method drastically alters the complexity landscape by decomposing the structure into d subtrees of height h/d. Key generation remains the most resource-intensive process at $O(d \cdot m^{h/d})$, since each subtree requires exponential effort in proportion to its height, a cost necessary to preserve cryptographic security. For practical deployment, however, the linear reliance on d rather than h is crucial, enabling significant gains in verification ($O(d)$) and signature size ($O(k + d \cdot \|\|\pi\|\|)$). Similar trends apply to storage complexity: cold subtrees benefit from on-demand regeneration from compact seeds, while active memory requirements scale as $O(\mathrm{d} \cdot \mathrm{m}^{h/d})$ because working subtrees must be cached.

In a real-world setting, the signature tree has four layers and 64 levels, with the following parameters: m = 256, d = 4, and h = 64. As a result, even if each subtree has a height of 16 and there are 256^16 potential nodes, only the parts of the tree that are really needed are generated and saved. This architecture reduces memory utilization, using just 4MB of working memory overall, or around 1MB per subtree. The size of each signature is approximately 16.2 KB. The HORST signature is around 8 KB in size, the WOTS+ signature is approximately 2.2 KB, and there are four Verkle proofs, each of which is approximately 1.5 KB in size. Since each lattice-based VC proof verification takes around 2.5 milliseconds, verifying a signature takes about 10 milliseconds.

We conducted a comprehensive sensitivity analysis to quantify the performance trade-offs governed by the layer parameter d. Holding $h = 64$ and $m = 256$ constant, our analysis reveals precise scaling relationships: increasing d from 4 to 6 reduces memory usage by 48% (from 4.0 MB to 2.1 MB) but increases verification time by 52% (from 9.8 ms to 14.9 ms) due to additional proof verifications. Conversely, decreasing d to 2 improves verification speed to 5.1 ms but increases memory usage to 9.8 MB. This analysis establishes that each unit increase in d reduces memory usage by approximately 24% while increasing verification time by 26%, providing system architects with quantitative guidance for parameter selection based on specific deployment constraints such as memory-constrained IoT devices ($d = 6$) or high-throughput servers ($d = 2$).

System designers can choose d based on particular resource constraints with the help of this quantitative sensitivity analysis. For instance, $d = 2$ can be used for high-throughput server applications and $d = 6$ can be used for memory-constrained IoT devices.

This configuration is especially well suited for use cases such as blockchain lite clients, who need tiny and fast proofs; IoT devices, which have limited memory; and high-throughput systems, which enable parallel verification.

FVTs work really well in practice. Each subtree can fit inside the fast L1/L2 CPU cache (≤32KB), enabling quick access and even GPU concurrent processing. The tree depth adapts to system load using a simple optimization mechanism. If part of the structure breaks, it is far faster to rebuild a single subtree than to rebuild the entire tree. For blockchain lite clients, FVTs eliminate storage (2.5 MB vs. 18 MB), speed up verification (210 ms vs. 890 ms), and reduce proof sizes by 72%.

TABLE 3
AGAINST MONOLITHIC VERKLE TREES

Feature	FVT	Monolithic
Worst-case proof size	~16.2 KB	>> 100 KB (est.)
Key gen time	3.2 s	6.5h
Update complexity	$O(m^{h/d})$	$O(m^h)$
Fault containment	Yes	No

This efficient and flexible cryptographic system has scalability, post-quantum security, and scalability. It is particularly useful in blockchain contexts, where constant-size proofs, which enable secure transaction verification without keeping the whole blockchain, benefit lightweight clients. Furthermore, the architecture makes it easier for subtrees to share proofs efficiently, which enhances efficiency in distributed environments. In the context of the Internet of Things (IoT), the scheme's partial verification and on-the-fly

key generation are beneficial since they provide strong security while reducing memory and processing demands. The statelessness of the technique facilitates its safe implementation in large-scale or distributed systems, especially for post-quantum digital signatures.

VII. FRACTAL VERKLE-BASED DIGITAL SIGNATURE (FVDS)

Because of the rapid growth of quantum computing, post-quantum cryptography primitives that are robust against assaults from both classical and quantum adversaries have become essential. Among the various approaches, hash-based signatures have emerged as particularly promising due to their minimal security assumptions and proven security assurances. Conventional hash-based systems, however, sometimes have practical limitations in terms of the size of their signatures and the efficiency of their verification when implemented at scale.

The Fractal Verkle-Based Digital Signature (FVDS) scheme is a significant advancement in post-quantum signature design since it combines the efficacy of Verkle tree constructions with the security of hash-based signatures. Three significant post-quantum cryptography issues are resolved by this novel approach: scalable key management, facilitating efficient verification processes, and maintaining modest signature sizes suitable for constrained environments. FVDS uses the hierarchical structure of Fractional Verkle Trees to achieve these goals without compromising security.

Fundamentally, FVDS builds upon several popular cryptographic primitives, such as the one-time signature security of WOTS+, the efficient verification capabilities of Verkle trees, the few-time signature properties of HORST, and the organizational benefits of fractal (hierarchical) constructions.

For modern applications that require high performance and quantum resistance, such as blockchain systems, secure software updates, and IoT device authentication, a signature method that combines these two characteristics is particularly well-suited. The following subsections provide a detailed description of the key creation, signing, and verification processes that comprise the FVDS system.

Key Generation:

1. Selecting a Parameter:
   - Let h be the total tree height, m the branching factor (e.g. $m = 256$), and d be the number of subtree levels (e.g. $d = 4$).
   - Select a cryptographic hash function $H: \{0,1\}^* \to \{0,1\}^\lambda$ and a pseudorandom function $\mathrm{PRF} : \{0,1\}^\lambda \times \mathbb{Z} \to \{0,1\}^\lambda$.
2. Derivation of the Master Key:

Create a master seed $S \in \{0,1\}^\lambda$ and determine subtree seeds:

$$S_j = \mathrm{PRF}(S, j) \text{ for } j \in \{0,1,\dots,d-1\} \quad (21)$$

3. Subtree Configuration:

For each layer j, create a Verkle subtree VT_j of height h/d using S_j. Each leaf leaf $_i$ commits to a HORST public key:

$$\mathrm{leaf}_i = H(\mathrm{HORST\_PK}_i) \text{ where } i \in \{1,2,\dots,m^{h/d}\}. \quad (22)$$

4. Root Commitment:

The master public key is the root of the topmost subtree:

$$\mathrm{PK}_{\mathrm{FVDS}} = \mathrm{Commit}(\mathrm{Root}_{d-1}). \quad (23)$$

The private key is $\mathrm{SK}_{\mathrm{FVDS}} = (S, \{S_j\})$.

Signing Algorithm:

To sign a message M :

1. We need to determine the Index in order to sign a message. Derive the HORST key index i with:
$$i = H(M) \bmod m^{h/d} \quad (24)$$
2. Generate a HORST signature σ_H over M :
$$\sigma_H = \mathrm{HORST} - \mathrm{Sign}(M, \mathrm{HORST} - \mathrm{SK}_i) \quad (25)$$
3. Sign the HORST root Root $_0$, use the corresponding WOTS + key:
$$\sigma_W = \mathrm{WOTS}_{+-\mathrm{Sign}}(\mathrm{Root}_0, \mathrm{WOTS\text{-}SK}_i) \quad (26)$$
4. For each subtree layer j, produce a proof π_j verifying the path from HORST-PK $_i$ to Root_{d-1} :
$$\pi_j = \mathrm{VC} - \mathrm{Open}(\mathrm{VT}_j, \mathrm{path}_i) \text{ for } j \in \{0,1,\dots,d-1\} \quad (27)$$
5. The final signature is the tuple:
$$\Sigma = \left(\sigma_H, \sigma_W, \{\pi_j\}_{j=0}^{d-1}\right) \quad (28)$$

Verification:

Given ($M, \Sigma, \mathrm{PK}_{\mathrm{FVDS}}$):

1. For index validation, we need to compute again $i = H(M) \bmod m^{h/d}$.
2. For HORST verification, check σ_H against HORST-PK $_i$:
$$\mathrm{HORST} - \mathrm{Verify}(M, \sigma_H, \mathrm{HORST} - \mathrm{PK}_i) \stackrel{?}{=} 1 \quad (29)$$
3. For WOTS+ Verification, validate σ_W authenticates Root $_0$:
$$\mathrm{WOTS{+}\text{-}Verify}(\mathrm{Root}_0, \sigma_W, \mathrm{WOTS\text{-}PK}_i) \stackrel{?}{=} 1 \quad (30)$$
4. For each π_j, confirm the commitment path:
$$\mathrm{VC} - \mathrm{Verify}(\mathrm{Root}_j, \pi_j) \stackrel{?}{=} 1 \quad (31)$$
5. Accept Σ if all steps succeed.

VIII. SECURITY PROOF FOR FVDS

The three main cryptographic presumptions that underlie the security of FVDS are the pseudo randomness of PRF, the binding of vector commitments, and collision resistance of H. In order to prevent an attacker from discovering two messages that produce the same HORST key index, the hash function H needs to be collision-resistant. In order to prevent an adversary from opening a commitment to two distinct values for the same leaf, the vector commitment scheme (VC) has to be binding. The structure of the FVT must be protected by a safe pseudorandom function that guarantees the derived layer seeds S_j are identical to random.

The components must be formed using quantum-resistant primitives in order to provide viable post-quantum security. SHA-256 or SHA-3 can be used as the hash function H because Grover's technique only offers a quadratic speedup,

which is countered by 256-bit security. There is a standardized lattice-based approach for instantiating the PRF. A lattice-based vector commitment framework [34] is a good option to take the role of lattice-based commitments and offer protection from quantum attackers.

FVDS remains secure against quantum adversaries if: H is quantum-resistant, the PRF is quantum-safe (e.g. based on LWE) and VC scheme is post-quantum (e.g. using lattice-based commitments).

Using a reductionist approach to security, we show that any successful attack on the scheme entails an attack on one of the cryptographic primitives that underlie it. Assuming the collision resistance of the hash function H, the binding property of the vector commitment scheme, and the security of the pseudorandom function, FVDS specifically achieves existential unforgeability under chosen message attacks (EUF-CMA). According to standard cryptographic proof techniques, a thorough security reduction demonstrates that an adversary's advantage in forging signatures is limited by the sum of advantages against these individual components.

TABLE 4
COMPARATIVE SECURITY

Property	FVDS	Monolithic Schemes
Attack Surface	Limited to active subtrees	Full tree exposure
Key Exposure	Isolated to one HORST instance	Global compromise
Proof Size	$O(d \cdot \|\pi\|)$	$O(h \cdot \|\pi\|)$

The notation $O(d \cdot |\pi|)$ indicates that the size of a single Verkle proof ($|\pi|$) and the number of subtree layers (d) both increase the proof size linearly. For $d = 4$ and $|\pi| = 1.5$ KB, the total proof size for the Verkle component would be around 6 KB. The scaling of proof size in monolithic trees, on the other hand, is described by $O(h \cdot |\pi|)$, where it relies on the whole tree height (h), resulting in substantially bigger proofs (e.g., 14.7 KB for $h = 64$).

Several adversary models, such as classical and quantum attackers with polynomial resources, are taken into account in our security analysis. The proof structure uses a series of games in which we address vector commitment binding violations, remove possible hash collisions, and then replace the PRF with a truly random function. Any lingering forgery ability would immediately compromise the security of the WOTS+ or HORST signature components in the final game. This reduction guarantees that FVDS inherits the thoroughly researched security characteristics of its building blocks and offers a solid basis for the security claims.

FVDS outperforms monolithic Verkle tree-based signatures in a number of important security characteristics. By requiring the simultaneous compromise of several subtrees, FVDS maintains a lower trust surface than monolithic architectures, which run the risk of full-tree exposure from a single point of failure. Errors in FVDS remain isolated inside certain subtrees, whereas defects in monolithic structures propagate throughout the world. FVDS enables localized modifications without necessitating a full tree recomputation, as contrast to monolithic methods that could affect the whole tree with every update. While effective forgery against FVDS requires concurrently violating the vector commitment, HORST, and PRF, monolithic techniques simply need to compromise the VC or root. Both techniques derive security from their component pieces (Hash functions, Vector Commitment, and Pseudorandom function), but FVDS's lower attack surface provides greater practical assurances.

IX. PERFORMANCE AND EFFICIENCY ANALYSIS

Fractional Verkle Trees provide notable improvements in computational and storage efficiency over traditional architectures. The hierarchical split into several interconnected subtrees provides for enhanced performance across several important metrics while maintaining same security assurances. This architectural innovation is highly advantageous for large-scale deployments in resource-constrained IoT contexts and blockchain networks, where security and performance are crucial.

The reduction in signature size is one of the largest efficiency advantages. Because conventional Verkle trees need proofs that scale logarithmically with the total tree height, their authentication pathways increase in size as the tree grows. In contrast, the fractional approach yields proofs of constant size for each tier of the subtree. In practice, the fractional structure decreases proof sizes from around 14.7 kilobytes in monolithic designs to only 2.4 kilobytes for a configuration with a branching factor of 256 and a total height of 64 split into 4 layers. This 84 percent decrease in signature size results in a considerable reduction in bandwidth needs and storage costs, making the technology particularly well-suited for applications with constrained bandwidth.

Performance gains in verification are as noteworthy with the fractional design. While monolithic trees must process the whole route from leaf to root for each verification, the fractional architecture enables independent subtree levels to be confirmed concurrently. This fourfold boost in verification performance is particularly important for real-time validation scenarios in distant networks, enabling more efficient systems and more responsive applications.

The fractional technique has notable benefits in memory efficiency as well. The system's hierarchical structure allows it to store just currently utilized subtrees in memory, and when needed, it may reconstruct other subtrees from compact seeds. Because of this 78% reduction in active memory consumption, the technology may be deployed on devices with low resources while maintaining the same security characteristics.

Update operations benefit similarly from the architectural improvements. In conventional architectures, authentication pathways need to be computed across the tree height for each change. The fractional design localizes these changes to the affected subtree levels, reducing the computational cost from hours of processing to seconds on average. This improvement becomes more significant as the system expands, allowing the technology to be applied in dynamic environments that need regular upgrades.

System architects can use fractional design to balance efficiency benefits based on specific application needs. Increasing the number of subtree levels can reduce memory needs while marginally increasing verification durations. The technology's versatility allows it to be customized for a variety of deployment scenarios, such as high-performance computer settings and embedded devices with constrained resources.

A performance evaluation of FVDS is presented in this section, combining analytical estimates for post-quantum components with empirical measurements from a functional Python prototype. The prototype was created in order to confirm the FVDS design's structural overhead and architectural workflow.

We created a working Python prototype in order to verify the fundamental FVDS architecture and its operational overhead. This implementation uses the $pycryptodome$ library for hash functions (SHA-256) to concretely realize the algorithms specified in Appendix A. A standard desktop platform (Intel Core i7-12700K, 32 GB RAM, Ubuntu 22.04) was used to run all benchmarks. We performed 1,000 signing operations for a configuration with the parameters $h = 64, m = 256$, and $d = 4$. The structural efficiency of the FVT design was confirmed by this procedure, which offered direct measurements for the composition of the signatures as well as the computational cost of the tree traversal and hash-based operations.

The current prototype illustrates the FVDS workflow using a simplified commitment scheme. This needs to be swapped out for a post-quantum secure vector commitment (VC), like a lattice-based scheme, in a production deployment. Compared to the classical primitives utilized in our prototype, lattice-based cryptography is intrinsically more computationally demanding. Therefore, the verification time for the VC proofs is based on an analytical estimate of approximately 2.5 ms per proof in order to present a comprehensive and realistic performance profile. The estimated cost of this important security enhancement in an optimal implementation is based on performance targets for cutting-edge lattice constructions.

These two components are combined in the results below: analytical estimates for the post-quantum vector commitment operations and measured values from our prototype for the structural overhead and hash-based signature components.

- Signature Size: The total signature Σ is composed of a HORST signature (~ 8 KB), a WOTS+ signature (~ 2.2 KB), and d lattice-based VC proofs [35]. We estimate ~ 1.5 KB per proof for each of the 4 layers, resulting in 6 KB for proofs. The total estimated signature size is 16.2 KB.
- Verification Time: Verification requires d vector commitment operations. We estimate each lattice-based VC proof verification takes approximately 2.5 ms, leading to a total verification time of ~ 10 ms.
- Memory and Key Generation: The memory requirement (~ 4MB) and key generation time (on the order of seconds/minutes) remain as key trade-offs that enable the architectural benefits of fault containment and localized updates.

Table 5 compares these estimates with the published, real-world performance of the stateless hash-based standard, SPHINCS+-128s [36], providing a realistic context for FVDS's performance profile.

TABLE 5
COMPARATIVE PERFORMANCE: FVDS VS. SPHINCS+

Metric	SPHINCS+-128s (Measured)	FVDS (Prototype/ Estimate)	Advantage
Signature Size	~ 41 KB	~ 16.2 KB	2.5x smaller
Verification Time	~ 0.5 ms	~ 10 ms	Slower, but offers architectural advantages (fault containment, scalable updates)
Key Generation Time	~ 100 ms	~ seconds/minutes	Slower, but a one-time, offline cost
Core Innovation	Stateless	Fault Containment, Scalable Updates	Architectural

The research highlights that FVDS is made for a different purpose than SPHINCS+ and validates a definite trade-off [37]. Its main benefit is that its signature size is much less (about 2.5 times smaller), which means that applications like blockchain may save a large amount of bandwidth. The use of post-quantum secure primitives results in slower verification, which is the price paid for the architectural advantages of FVDS, such as fault containment and effective, localized updates. Because of its performance profile, FVDS may be used in systems where scalable key management is essential and bandwidth are the main limitation.

Additionally, the fractional architecture offers a critical fault containment feature that improves overall system robustness and maintainability: if a single subtree is compromised or corrupted, it can be recreated from its seed separately without affecting the tree structure as a whole.

X. CONCLUSIONS

Since quantum computing presents a significant danger to encryption systems, robust alternatives must be created. Hash-based signature systems are promising since they are secure from quantum assaults. This paper introduces Fractional Verkle Trees, a unique enhancement of traditional hash-based signatures that overcomes performance limitations while maintaining security. Building on this approach, we have demonstrated that digital signatures based on fractional Verkle Trees may be used to provide quantum-resistant authentication with unprecedented efficiency advantages. By redesigning data structures, the technique offers practical implementation benefits.

Fractional Verkle Trees offer a fresh approach to cryptographic data structures by avoiding the scalability constraints of traditional monolithic designs while preserving strong security guarantees. The resulting fractional Verkle signature technique extends these advantages to the domain of digital authentication, providing tiny signatures and fast verification without compromising post-quantum security. By employing a hierarchical decomposition technique, this system provides significant efficiency benefits across key performance parameters, including memory use, verification speed, and signature size, when compared to conventional Verkle tree constructs. This results in practical deployment advantages for digital signatures, especially in contexts with limited bandwidth and high volumes systems. It is established that these improvements are accomplished without sacrificing security using a reduction-based verification of the system's resilience to adversarial assaults.

Localized subtree operations are an architectural breakthrough that enables realistic deployment in large-scale systems where classic Verkle tree implementations faced prohibitive processing costs. This discovery is similarly significant for signature schemes, where the fractal structure enables concurrent verification and selective subtree updates. System architects can best combine memory economy and verification speed for specific application needs because to this design's adaptability. The signature scheme's flexibility is particularly helpful for creating systems that require regular key changes or varying security levels. This flexibility ensures that the system is compatible with both high-performance blockchain networks and IoT devices with less resources.

Practically speaking, FVDS is ideally suited for incorporation into IoT secure boot protocols and blockchain light clients. Without keeping track of the complete chain state, light clients can use the tiny, constant-sized proofs of FVDS in a blockchain setting to verify transactions quickly. The instantaneous regeneration of subtrees from seeds reduces the need for long-term storage for IoT devices. The synchronization of updated subtree states across nodes is a crucial system-level consideration in distributed settings; this problem calls for further research on versioned commitments or consensus-driven update mechanisms. To ensure that any synchronization issues remain confined and safeguard the overall security and integrity of the system, FVDS has an integrated fault management capability.

The theoretical and preliminary practical foundations of FVDS are established in this work. To replace the analytical model used in this study, a production-grade, post-quantum secure vector commitment scheme must be integrated; a formal security audit of the entire system must be conducted; and a high-performance implementation in a systems programming language (such as C++ or Rust) must be developed for rigorous benchmarking on realistic hardware targets, such as cloud platforms and embedded devices. In addition to these, investigating stateless variations and batch verification methods offers encouraging paths to improve adaptability and effectiveness in extensive implementations.

For next-generation cryptography systems that need both scalability and robust security guarantees, fractional verkle trees and their derivative signature schemes are a desirable alternative due to their demonstrated performance advantages and robust post-quantum security characteristics. In addition to opening up new possibilities for efficient quantum-resistant digital signatures and establishing the foundation for the widespread, real-world use of verifiable data structures in performance-sensitive situations, this work preserves the cryptographic integrity required for modern security applications.

APPENDIX A
FVDS ALGORITHMS.
A.1 Notation and Assumptions

Symbol	Meaning
$H(x)$	Cryptographic hash function (e.g., SHA-256).
PRF$(seed, label)$	Pseudorandom function producing deterministic randomness.
VC.Commit$(\mathrm{v}) \rightarrow (C, S)$	Vector-commitment procedure returning commitment $C$and state S.
VC.Open$(S, i) \rightarrow o$	Opening proof for element v_i.
VC.Verify(C, i, v_i, o) $\rightarrow$ bool	Verification of an opening.
HORST,WOTS	Hash-based one-time and few-time signature primitives.
h	Total Verkle-tree height.
m	Branching factor.
d	Number of fractional layers.
$s = h/d$	Height of each subtree (assumed integer).

All subtrees and key pairs are generated deterministically from the master seed. Indexes are zero-based, layer 0 is the lowest layer.

A.2 FVDS.KeyGen

Input:
master_seed // random bitstring
parameters (h, m, d)

Output:
PK_master // public root commitment
SK = master_seed

Procedure:
1. s ← h / d
2. For each layer ℓ in 0 .. d−1:
seed_ℓ ← PRF(master_seed, "layer" || ℓ)
L_ℓ ← m^(d−1−ℓ) // number of subtrees at this layer
For each subtree index t in 0 .. L_ℓ−1:
(C_ℓ,t, S_ℓ,t) ← GenerateSubtree(seed_ℓ, t, s, m)
3. For ℓ = 0 .. d−2:
For each parent p in 0 .. m^(d−2−ℓ)−1:
vector ← [C_ℓ,child for all children of parent p]
(C_ℓ+1,p, S_ℓ+1,p) ← VC.Commit(vector)
4. PK_master ← C_d−1,0
5. return (PK_master, SK)

Sub-procedure GenerateSubtree
Input: seed_ℓ, t, s, m
Output: (C, S)

1. leaves_count ← m^s
2. For i = 0 .. leaves_count−1:
leaf_seed ← PRF(seed_ℓ, "subtree" || t || "leaf" || i)
(HORST_SK[i], HORST_PK[i]) ← HORST.KeyGen(leaf_seed)
leaf_val[i] ← H(HORST_PK[i])
3. (C, S) ← VC.Commit(leaf_val[0 .. leaves_count−1])
4. return (C, S)

A.3 FVDS.KeyGen

Input:
SK = master_seed
Message M
Output:
SIG = < idx_global, HORST_PK, HORST_SIG, WOTS_PK, WOTS_SIG, openings[0..d−1] >

1. idx_global ← H(SK || M) mod (m^h)

2. For ℓ = 0 .. d−1:
t_ℓ ← floor(idx_global / m^(ℓ * s)) mod m^(d−1−ℓ)
i ← idx_global mod m^s
3. seed_0 ← PRF(SK, "layer" || 0)
leaf_seed ← PRF(seed_0, "subtree" || t_0 || "leaf" || i)
(HORST_SK, HORST_PK) ← HORST.KeyGen(leaf_seed)
4. HORST_SIG ← HORST.Sign(HORST_SK, M)
5. wots_seed ← PRF(seed_0, "subtree" || t_0 || "wots" || i)
(WOTS_SK, WOTS_PK) ← WOTS.KeyGen(wots_seed)
WOTS_SIG ← WOTS.Sign(WOTS_SK, HORST_PK)
6. proofs ← []
cur_val ← H(HORST_PK)
i_sub ← i
For ℓ = 0 .. d−1:
seed_ℓ ← PRF(SK, "layer" || ℓ)
S_ℓ,t_ℓ ← RegenerateSubtreeState(seed_ℓ, t_ℓ, s, m)
opening_ℓ ← VC.Open(S_ℓ,t_ℓ, i_sub)
proofs.append(opening_ℓ)
cur_val ← C_ℓ,t_ℓ
i_sub ← child_index_of(t_ℓ)
7. SIG ← (idx_global, HORST_PK, HORST_SIG, WOTS_PK, WOTS_SIG, proofs)
8. return SIG

A.4 FVDS. Verify

Input:
PK_master
Message M
SIG = (idx_global, HORST_PK, HORST_SIG, WOTS_PK, WOTS_SIG, proofs[0..d−1])

Output:
VALID / INVALID

1. Recompute path:
For ℓ = 0 .. d−1:
t_ℓ ← floor(idx_global / m^(ℓ * s)) mod m^(d−1−ℓ)
i ← idx_global mod m^s
2. if HORST.Verify(HORST_PK, M, HORST_SIG) == false: return INVALID
3. if WOTS.Verify(WOTS_PK, HORST_PK, WOTS_SIG) == false: return INVALID
4. cur_val ← H(HORST_PK)
For ℓ = 0 .. d−1:
parent_commit ← (ℓ == d−1) ? PK_master : commitment_at_layer(ℓ+1, parent_index)
if VC.Verify(parent_commit, child_index_of(t_ℓ), cur_val, proofs[ℓ]) == false:
return INVALID
cur_val ← parent_commit
5. if cur_val ≠ PK_master: return INVALID
6. return VALID

A.5 Helper Interfaces

HORST.KeyGen(seed) → (HORST_SK, HORST_PK)
HORST.Sign(HORST_SK, M) → HORST_SIG
HORST.Verify(HORST_PK, M, HORST_SIG) → bool

WOTS.KeyGen(seed) → (WOTS_SK, WOTS_PK)
WOTS.Sign(WOTS_SK, msg) → WOTS_SIG
WOTS.Verify(WOTS_PK, msg, WOTS_SIG) → bool

VC.Commit(values[0..n−1]) → (commitment, state)
VC.Open(state, index) → opening
VC.Verify(commitment, index, value, opening) → bool

PRF(seed, label) → bitstring

A.6 Complexity Remarks

- **Signature size:** ≈ |HORST_SIG| + |WOTS_SIG| + d × |VC_opening| + |index encoding|.
- **Signing cost:** one HORST.Sign + one WOTS.Sign + d × VC.Open.
- **Verification cost:** one HORST.Verify + one WOTS.Verify + d × VC.Verify.
- **Key generation:** $O(m^h)$ if expanded fully; deterministic seeds allow lazy regeneration.

A.7 Implementation Note

A stateless signer is made possible by the deterministic recalculation of all commitments and openings from the master seed.

Intermediate subtree commitments, which trade bandwidth for verifier simplicity, may be included in the signature if verifier recomputation is undesirable.

APPENDIX B
FORMAL SECURITY REDUCTION FOR FVDS

We prove existential unforgeability under chosen-message attacks (EUF-CMA) for the FVDS scheme in the Quantum Random Oracle Model (QROM).

Let $\Pi_{\text{FVDS}} = (\text{KeyGen}, \text{Sign}, \text{Verify})$ be our signature scheme with security parameter λ. The EUFCMA experiment $\text{Exp}_{\mathcal{A}}^{\text{euf-cma}}(\lambda)$ is defined as:

1. Key Generation: The challenger runs $(PK, SK) \leftarrow \text{FVDS.KeyGen}\left(1^{\lambda}\right)$.
2. Query Phase: The adversary $\mathcal{A}$ may make:
   - Signing queries: For messages $M_1, \ldots, M_q$, receives $\Sigma_i \leftarrow \text{FVDS.Sign}(SK, M_i)$.
   - Random oracle queries: Quantum access to $H: \{0,1\}^* \rightarrow \{0,1\}^{\kappa}$.
3. Forgery: $\mathcal{A}$ outputs (M^*, Σ^*). $\mathcal{A}$ wins if:
   - $\text{FVDS.Verify}\ (PK, M^*, \Sigma^*) = 1$, and
   - $M^* \notin \{M_1, \ldots, M_q\}$.

The advantage is:

$$\text{Adv}_{\mathcal{A}}^{\text{euf-cma}}(\lambda) = \Pr\left[\text{Exp}_{\mathcal{A}}^{\text{euf-cma}}(\lambda) = 1\right]$$

We rely on the following assumptions:

1. PRF Security: The pseudorandom function $\text{PRF}: \{0,1\}^{\lambda} \times \mathbb{Z} \rightarrow \{0,1\}^{\lambda}$ is secure against quantum adversaries:
$$\text{Adv}_{\mathcal{B}}^{\text{prf}}(\lambda) = \left|\Pr\left[\mathcal{B}^{\text{PRF}_K(\cdot)} = 1\right] - \Pr\left[\mathcal{B}^{U(\cdot)} = 1\right]\right| \leq \text{negl}(\lambda)$$
2. Vector Commitment Binding: For the VC scheme, for any QPT adversary $\mathcal{B}$:
$$Adv_B^{bind}(\lambda) = Pr\begin{bmatrix} VC.Open(C, i, v, \pi) = 1 \wedge \\ VC.Open(C, i, v', \pi') = 1 \wedge v \neq v' \end{bmatrix} \leq negl(\lambda)$$
3. HORST EUF-CMA Security: For any QPT adversary $\mathcal{B}$ making at most q_s signing queries:
$$\text{Adv}_{\text{HORST},\mathcal{B}}^{\text{euf-cma}}(\lambda) \leq \text{negl}(\lambda)$$
4. WOTS+ EUF-CMA Security: Similarly, for any QPT adversary $\mathcal{B}$:
$$\text{Adv}_{\text{WOTS}+,\mathcal{B}}^{\text{euf-cma}}(\lambda) \leq \text{negl}(\lambda).$$
5. Hash Function: H is modeled as a quantum random oracle with output length $\kappa = 256$ bits.

B.3. Security Reduction via Game Hopping

We proceed through a sequence of games G_0 to G_4.

Game G_0 :

This is the real EUF-CMA experiment. Let:

$$\varepsilon_0 = \Pr[\mathcal{A} \text{ wins in } \text{G}_0].$$

Game G_1 :

We replace the PRF used for index derivation $i = \text{PRF}(S, M)$ with a truly random function $f: \{0,1\}^* \rightarrow \{0,1\}^{\lambda}$. By the PRF security, there exists an adversary $\mathcal{B}_1$ such that:

$$|\varepsilon_1 - \varepsilon_0| \leq \text{Adv}_{\mathcal{B}_1}^{\text{prf}}(\lambda)$$

Game G_2 :

We abort if the adversary produces a forgery that involves a vector commitment opening violation. That is, if for some layer j, the proof π_j^* opens to two different values for the same path.

By the binding property of the VC scheme, there exists $\mathcal{B}_2$ such that:

$$|\varepsilon_2 - \varepsilon_1| \leq \text{Adv}_{\mathcal{B}_2}^{\text{bind}}(\lambda).$$

Game G_3 :

We now handle forgeries where the WOTS+ key used in the forgery was never used in any signing query. In this case, the adversary has forged a WOTS+ signature. We construct a reduction $\mathcal{B}_3$ that breaks WOTS+ security:

$$\varepsilon_3 \leq \text{Adv}_{\text{WOTS}+,\mathcal{B}_3}^{\text{euf-cma}}(\lambda)$$

Game G_4 :

If the WOTS+ key was used before, then the forgery must occur at the HORST level. We construct a reduction $\mathcal{B}_4$ that breaks HORST security:

$$\varepsilon_4 \leq \text{Adv}_{\text{HORST},\mathcal{B}_4}^{\text{euf-cma}}(\lambda).$$

QROM Accounting:

Since the adversary has quantum access to H, we must account for the cost of reprogramming the random oracle in the reduction. Using the One-Way to Hiding (O2H) lemma [Unruh, 2015], the additional loss is:

$$O\left(\frac{q_H}{2^{\kappa}}\right)$$

where q_H is the number of quantum random oracle queries.

B.4. Main Theorem

Theorem 1 (EUF-CMA Security of FVDS in QROM) For any quantum polynomial-time adversary $\mathcal{A}$ making at most q_H quantum random oracle queries and q_S signing queries, there exist QPT adversaries $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$ such that:

$$\mathrm{Adv}^{\text{euf-cma}}_{FVDS,\mathcal{A}}(\lambda) \leq \mathrm{Adv}^{\text{prf}}_{\mathcal{B}_1}(\lambda) + \mathrm{Adv}^{\text{binding}}_{\mathcal{B}_2}(\lambda) + \mathrm{Adv}^{\text{euf-cma}}_{\text{WOTS+},\mathcal{B}_3}(\lambda) + \mathrm{Adv}^{\text{euf-cma}}_{\text{HORST},\mathcal{B}_4}(\lambda) + O\left(\frac{q_H^2}{2^\kappa}\right)$$

Proof Sketch:

The proof follows the game sequence above. In G_1, we remove dependence on the PRF. In G_2, we enforce unique openings. In G_3 and G_4, we reduce to WOTS+ and HORST security, respectively. The QROM term accounts for oracle reprogramming across all reductions.

Since all terms on the right-hand side are negligible under our assumptions, FVDS is EUF-CMA secure in the QROM.

The proof holds against quantum adversaries, provided:

- The PRF is quantum-safe (e.g., based on LWE),
- The VC is instantiated with a lattice-based binding commitment,
- HORST and WOTS+ use quantum-resistant parameters (e.g., as in SPHINCS+).

ACKNOWLEDGMENT

The authors would like to express their gratitude to the NATO Science for Peace and Security Programme for supporting this research under grant G7394 "Post-quantum Digital Signature using Verkle Trees".

REFERENCES

[1] Bhaskar, B.; Sendrier, N. McEliece cryptosystem implementation: Theory and practice. In *Post-Quantum Cryptography: Second International Workshop, PQCrypto 2008 Cincinnati, OH, USA, October 17-19, 2008 Proceedings 2*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 47–62.

[2] Chen, L.; Jordan, S.; Liu, Y.-K.; Moody, D.; Peralta, R.; Ray, A.P.; Smith-Tone, D. Report on Post-Quantum Cryptography; US Department of Commerce, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2016; Volume 12.

[3] Johannes, B.; Dahmen, E.; Szydlo, M. Hash-based digital signature schemes. In Post-Quantum Cryptography; Springer: Berlin/Heidelberg, Germany, 2009; pp. 35–93. https://doi.org/10.1007/978-3-540-88702-7_3.

[4] Yin, H.L.; Fu, Y.; Li, C.L.; Weng, C.X.; Li, B.H.; Gu, J.; Chen, Z.B. Experimental quantum secure network with digital signatures and encryption. Natl. Sci. Rev. 2023, 10, nwac228.

[5] Al Attar, T.N.A.; Mohammed, R.N. Optimization of Lattice-Based Cryptographic Key Generation using Genetic Algorithms for Post-Quantum Security. UHD J. Sci. Technol. 2025, 9, 93–105.

[6] Cao, Y., Wu, Y., Qin, L., Chen, S., & Chang, C. H. (2022). Area, time and energy efficient multicore hardware accelerators for extended Merkle signature scheme. IEEE Transactions on Circuits and Systems I: Regular Papers, 69(12), 4908-4918.

[7] Buchmann, J. A., Butin, D., Göpfert, F., & Petzoldt, A. (2016). Post-quantum cryptography: state of the art. The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday, 88-108.

[8] Lee, J.; Park, Y. Horsic+: An efficient post-quantum few-time signature scheme. Appl. Sci. 2021, 11, 7350.

[9] Bernstein, D.J.; Lange, T. Post-quantum cryptography. Nature 2017, 549, 188–194.

[10] Aumasson, J.P.; Endignoux, G. Clarifying the subset-resilience problem. Cryptol. Eprint Archive 2017.

[11] Lin, K. W., & Chen, Y. C. (2023, July). A File Verification Scheme Based on Verkle Trees. In 2023 International Conference on Consumer Electronics-Taiwan (ICCE-Taiwan) (pp. 295-296). IEEE.

[12] Kuszmaul, J., 2019. Verkle trees. Verkle Trees, 1(1).

[13] Catalano, D., & Fiore, D. (2013). Vector commitments and their applications. In Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16 (pp. 55-72). Springer Berlin Heidelberg.

[14] Wang, H., Yiu, S. M., Zhao, Y., & Jiang, Z. L. (2024, April). Updatable, aggregatable, succinct mercurial vector commitment from lattice. In IACR International Conference on Public-Key Cryptography (pp. 3-35). Cham: Springer Nature Switzerland.

[15] Kim, D., Choi, H., & Seo, S. C. (2024). Parallel implementation of SPHINCS+ with GPUs. IEEE Transactions on Circuits and Systems I: Regular Papers, 71(6), 2810-2823.

[16] Bavdekar, R., Chopde, E. J., Agrawal, A., Bhatia, A., & Tiwari, K. (2023, January). Post quantum cryptography: a review of techniques, challenges and standardizations. In 2023 International Conference on Information Networking (ICOIN) (pp. 146-151). IEEE.

[17] Chang, M.H.; Yeh, Y.S. Improving Lamport one-time signature scheme. Appl. Math. Comput. 2005, 167, 118–124

[18] Dods, C.; Smart, N.P.; Stam, M. Hash based digital signature schemes. In Cryptography and Coding: 10th IMA International Conference, Cirencester, UK, December 19-21, 2005. Proceedings 10; Springer: Berlin/Heidelberg, Germany, 2005; pp. 96–115.

[19] Lamport, L. Using time instead of timeout for fault-tolerant distributed systems. ACM Trans. Program. Lang. Syst. (TOPLAS) 1984, 6, 254–280.

[20] Iavich, M., Kuchukhidze, T., & Bocu, R. (2023, March). A Post-quantum Cryptosystem with a Hybrid Quantum Random Number Generator. In International Conference on Advanced Information Networking and Applications (pp. 367-378). Cham: Springer International Publishing.

[21] Buchmann, J.; Dahmen, E.; Ereth, S.; Hülsing, A.; Rückert, M. On the security of the Winternitz one-time signature scheme. Int. J. Appl. Cryptogr. 2013, 3, 84–96.

[22] Kudinov, M.; Hülsing, A.; Ronen, E.; Yogev, E. SPHINCS+ C: Compressing SPHINCS+ with (almost) no cost. Cryptol. Eprint Archive 2022.

[23] Algazy, K., Sakan, K., Khompysh, A., & Dyusenbayev, D. (2024). Development of a new post-quantum digital signature algo-rithm: Syrga-1. Computers, 13(1), 26.

[24] Dziechciarz, D., & Niemiec, M. (2024). Efficiency analysis of NIST-standardized post-quantum cryptographic algorithms for digital signatures in various environments. Electronics, 14(1), 70.

[25] Merkle, R.C. A DIGITAL SIGNATURE BASED ON A CONVENTIONAL ENCRYPTION FUNCTION. In Advances in Cryptology-CRYPTO'87: Proceedings; Springer: Berlin/Heidelberg, Germany, 2003; Volmue 293, pp. 369.

[26] Bernstein, D.J.; Hopwood, D.; Hülsing, A.; Lange, T.; Niederhagen, R.; Papachristodoulou, L.; Wilcox-O'Hearn, Z. SPHINCS: Practical stateless hash-based signatures. In Annual International Conference on the Theory and Applications of Cryptographic Techniques; Springer: Berlin/Heidelberg, Germany, 2015; pp. 368–397.

[27] Fernandez-Carames, T.M.; Fraga-Lamas, P. Towards post-quantum blockchain: A review on blockchain cryptography resistant to quantum computing attacks. IEEE Access 2020, 8, 21091–21116.

[28] Aumasson, J.P.; Endignoux, G. Improving stateless hash-based signatures. In Cryptographers' Track at the RSA Conference; Springer International Publishing: New York, NY, USA, 2018; pp. 219–242.

[29] Gorbunov, S., Reyzin, L., Wee, H., & Zhang, Z. (2020, October). Pointproofs: Aggregating proofs for multiple vector commitments. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (pp. 2007-2023).

[30] Lin, D.; Sako, K. Public-Key Cryptography–PKC 2019. In 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part I; Springer: Berlin/Heidelberg, Germany, 2019; pp. 14–17.

[31] Oberst, J. (2025). Towards Stateless Clients in Ethereum: Benchmarking Verkle Trees and Binary Merkle Trees with SNARKs. arXiv preprint arXiv:2504.14069.
[32] Wang, X., Xu, G., & Yu, Y. (2023). Lattice-based cryptography: A survey. Chinese Annals of Mathematics, Series B, 44(6), 945-960.
[33] Deshpande, S., Lee, Y., Karakuzu, C., Szefer, J., & Paek, Y. (2025). Sphincslet: An area-efficient accelerator for the full sphincs+ digital signature algorithm. ACM Transactions on Embedded Computing Systems.
[34] Fenzi, G., Moghaddas, H., & Nguyen, N. K. (2024). Lattice-based polynomial commitments: Towards asymptotic and concrete efficiency. Journal of Cryptology, 37(3), 31.
[35] Liu, F., Zheng, Z., Gong, Z., Tian, K., Zhang, Y., Hu, Z., ... & Xu, Q. (2024). A survey on lattice-based digital signature. Cybersecurity, 7(1), 7.
[36] Kannwischer, M. J., Krausz, M., Petri, R., & Yang, S. Y. (2024). pqm4: Benchmarking NIST additional post-quantum signature schemes on microcontrollers. Cryptology ePrint Archive.
[37] Perlner, R., Kelsey, J., & Cooper, D. (2022, September). Breaking category five SPHINCS+ with SHA-256. In International Conference on Post-Quantum Cryptography (pp. 501-522). Cham: Springer International Publishing.

Professor Maksim Iavich is the Head of Cyber Security Direction and an affiliated professor at Caucasus University's Caucasus School of Technology. In addition to teaching as an invited professor at Georgian Technical University, he is in charge of the bachelor's and master's degrees in information technology. Maksim also serves as the Scientific Cyber Security Association's (SCSA) president and CEO. Possessing a PhD in mathematics and a professorship in computer science, he advises Georgian and foreign enterprises on cyber security. On subjects like cyber security, cryptography, post-quantum cryptography, quantum cryptography, mathematical modeling, and simulations, he has written a large number of scholarly publications.

Professor Tamari Kuchukhidze is assistant professor at Caucasus University. She is making her thesis on the creation of the quantum random number generators for cryptography. Tamari is a researcher with a strong background in cybersecurity, cryptography, post-quantum cryptography, and software engineering. She holds a PH.D. degree in informatics. The topic was a post-quantum cryptosystem with a quantum random number generator. She has been actively engaged in research and development, cryptography, post-quantum cryptography and security. She is cryptographer at Scientific Cyber Security Association.

Professor Dr. Habil. Razvan Bocu, Department of Mathematics and Computer Science, Transilvania University of Brasov, Brasov 500091, Romania (razvan.bocu@unitbv.ro). Professor Dr. Habil. Razvan Bocu received a B.S. degree in computer science, a B.S. degree in sociology, and an M.S. degree in computer science from Transilvania University of Brasov, Romania, in 2005, 2007, and 2006, respectively. He also received a Ph.D. degree in Computer Science from the National University of Ireland, Cork, in 2010. He is a Research and Teaching Staff Member in the Department of Mathematics and Computer Science at the Transilvania University of Brasov. He is a member of the University's Doctoral School, in the field of Computer Science. In this capacity, he supervises complex PhD research processes, with strategic and multidisciplinary relevance. He is the Director of a NATO Scientific Research project, which addresses a cutting-edge problematic related to the development of quantum-resistant digital signature and data encryption models. He is author or coauthor of more than 70 technical papers, together with five books and book chapters. Dr. Bocu is an editorial reviewing board member of 28 technical journals in the field of information technology and biotechnology.