Digital Signature Design Using Verkle Tree

Maksim Iavich [1], Tamari Kuchukhidze [2]

[1] *Department of Computer Science, Caucasus University, 0102, Georgia*
[2] *International Black Sea University, 0131, Georgia*

Abstract
A significant amount of research has been done on quantum computers recently. Many of the current public key cryptosystems can be broken if humanity ever develop a powerful quantum computer. Such cryptosystems are used in a lot of commercial items nowadays. Solutions have been developed that appear to defend us from quantum attacks, but they cannot be applied in real life due to safety and efficiency issues. Hash-based digital signature methods are discussed in the article. Based on the Merkle tree, digital signatures are examined. The paper analyzes the new concepts utilizing vector commitments and Verkle tree.
The authors of this article provide novel technology of creating a digital signature scheme employing cutting-edge technology, Verkle tree. The authors offer the drawbacks of the scheme. The authors also provide the concepts of post-quantum signature design using Verkle Tree.

Keywords
quantum, quantum cryptography, signature schemes, Merkle tree, vector commitments, Verkle tree.

1. Introduction

Quantum computing will take over and become more widespread as time goes on. Quantum encryption, also known as post-quantum cryptography, is a cryptographic method for conventional computers that can fend off attacks from quantum computers. Computers will be able to carry out complex calculations considerably more quickly than traditional computers if they can take advantage of the special capabilities of quantum mechanics. It seems obvious that a quantum computer might do certain types of sophisticated calculations in a couple of hours. It is notable that a classical computer takes several years to accomplish these computations [1].

Most, if not all, currently in use traditional cryptosystems will likely be rendered useless by quantum computers. Cryptosystems based on the integer factorization problem (RSA) are commonly employed in reality nowadays, yet they do not resist quantum computer assaults. The RSA cryptosystem is utilized in a wide range of products and applications. This cryptosystem is now implemented into a growing number of commercial products. Since the RSA method is mostly used in encryption technology, it can be regarded as one of the most prevalent public key cryptosystems that develops with technology [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. Designing and putting into practice secure and effective post-quantum cryptosystems takes a lot of work.

The world's top scientists are working to create and refine quantum computers, yet even improved systems are vulnerable to powerful attacks [3]. The development of a cryptosystem that is suitable with both regular and post-quantum cryptography is our key objective. Yet it is equally critical that our systems are efficient, requiring less computational power and taking up little space on servers.

As quantum computing takes over, RSA and other asymmetric algorithms won't be able to keep our personal information secure. We are working to develop post-quantum systems because of this [4,5].

INFORMATION SOCIETY AND UNIVERSITY STUDIES (IVUS 2023), 12 May 2023, Kaunas, Lithuania
EMAIL: miavich@cu.edu.ge (M. Iavich); tamari.kuchukhidze@gmail.com (T. Kuchukhidze)
ORCID: 0000-0002-3109-7971 (M. Iavich); 0000-0003-1997-465X (T. Kuchukhidze)
© 2023 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-3575/Paper9.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

In practice, traditional digital signature schemes are vulnerable to quantum computer assaults. We are aiming to create RSA substitutes that are impervious to quantum computer assaults. Digital signature schemes based on hashes are one of the options. These programs employ a cryptographic hash function. The collision resistance of the hash algorithms used in these digital signature systems is what makes them secure. The hash-based Digital Signature Schemes are one RSA substitute. These systems' safety is dependent on how secure their cryptographic hash functions are.

2. Literature Review

Quantum computers can quickly break the encryption techniques now in use. As a result, assaults using quantum computers can now defeat conventional encryption techniques. Digital signature techniques that can withstand attacks from quantum computers are presented in this article [1]. The study [2] covers one-way functions as well as one-time signature techniques. In the paper [3], a McEliece public-key encryption system implementation with algorithmic and parameter choices is covered, along with the state of cryptanalysis at the time.

Researchers are looking towards quantum computers, according to article [4]. Cryptosystems based on the integer factoring problem can be cracked by quantum computers. It suggests that the RSA system, one of the best-known public-key cryptosystems, is vulnerable to attack by quantum computers. In [5,] various QRNG integration techniques are presented. The authors of articles [6–9] go over different quantum number generator-based hash-based digital signature techniques. The Merkle scheme is detailed in article [10]. The use of vector commitment is described in papers [11–13]. Verkle trees are described in this study [11].

3. Hash-based one-time signature schemes

Hash-based one-time signature methods offer particularly promising possibilities for the post-quantum era. We take into consideration signature schemes whose security is dependent solely on the cryptographic hash function's ability to resist collisions. The Lamport-Diffie one-time signature (LDOTS) system is an example [6]. While constructing randomized algorithms and protocols, it is believed that computers have access to a stream of truly random bits (that is a sequence of independent and unbiased coin tosses). In actual implementations, a sample is drawn from a "source of randomness" to generate this sequence [7].

We assume that Lamport-Diffie one-time signature's security parameter n is a integer. LDOTS uses a one-way function $f : \{0,1\}^n \rightarrow \{0,1\}^n$and cryptographic hashing function $g : \{0,1\}^n \rightarrow \{0,1\}^n$ to generate an LDOTS key pair, the LDOTS signature key X consists of a string of 2n bits of length n. X is chosen randomly:

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

The LDOTS verification key is Y

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

To calculate the key, we use the one-way function f:

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

So LDOTS key generation requires 2n evaluations of f. The signature and verification keys are n-length 2n-bit strings. In case of LDOTS signature generation, document $M \epsilon \{0,1\}^*$ is signed using LDOTS with signature key X. let's $g(M) = d = (d_{n-1}, \dots, d_0)$ is the message digest of M. The LDOTS signature is $sign = (x_{n-1}[d_{n-1}], \dots, x_1[d_1], x_0[d_0]) \epsilon \{0,1\}^{(n,n)}$.

This signature is made up of n bit strings of length n. As message digest function d, they are chosen. In most cases, hashes per second are used to measure how many cryptographic operations a processor can do at any one moment [8]. The i-th bit string of this signature is $x_i[0]$, but if the i-th bit in d is 0, i-th bit string of this signature is $x_i[1]$. The signature does not require the evaluation of f. The signature length is n^2.

In the case of LDOTS verification, if we want to verify the signature of M, $sign = (sign_{n-1}, \dots, sign_0)$, verifier calculates the message digest $d = (d_{n-1}, \dots, d_0)$. Then, it is determined whether it is or not:

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

LDOTS generates keys and signatures fairly quickly, however the signature size is quite huge. To decrease the size of signatures, the Winternitz one-time signature scheme (WOTS) is suggested. The concept is to sign multiple bits in a message digest using a single string, or a single string in a one-time signature key. WOTS utilizes a cryptographic hashing function and a one-way function, just like LDOTS.

One-time signature approaches are insufficient for most real-world scenarios since each key pair can only be used once for a signature. Ralph Merkle presented a solution to this problem. He suggests using a whole binary hash tree. The idea is to use a full binary hash tree to restrict the validity of an arbitrary but fixed number of one-time verification keys to a single public key, the hash tree's root.

4. Merkle tree authentication scheme

One-time signature schemes are very inconvenient to use, because to sign each message, we need to use a different key pair. The problem with such schemes are that they require to store n digests. For everyday use it is impractical, and we would like a scheme that allows us to store a uniform-sized digest, no matter how many files we have. Merkle Tree was proposed to solve this problem. By using a binary tree as the root, this approach can replace a large number of verification keys with a single public key. A cryptographic hash function and a one-time Lamport or Winternitz signature scheme are used in this system.

Merkle signature scheme (MSS) works for any cryptographic hash function and any one-time signature scheme. We assume that $g : \{0,1\}^* \rightarrow \{0,1\}^n$ is a cryptographic hashing function and the one-time signature scheme is already selected.

When generating the MSS key pair, the signer chooses $H \in \mathbb{N}, H \geq 2$. Then the key pair is generated. Using them, it will be possible to sign/verify 2^H documents. Note that this is a significant difference from signature schemes such as RSA and ECDSA, where many documents can potentially be signed/verified with just one key pair. However, in practice this number is also limited by the devices with which the signature is created or by certain policies [9].

The signer will generate 2^H unique key pairs $(X_j, Y_j), 0 \leq j < 2^H$. Here, X_j is the signature key and Y_j is the verification key. Both of them are bit strings. The leaves of the Merkle tree are $g(Y_j)$, $0 \leq j < 2^H$. The internal nodes of a Merkle tree are calculated according to the following rule: a parent node is the hash value of the concatenation of its left and right children. The MSS public key is the root of the Merkle tree. The MSS secret key is a sequence of 2^H one-time signature keys [10].

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

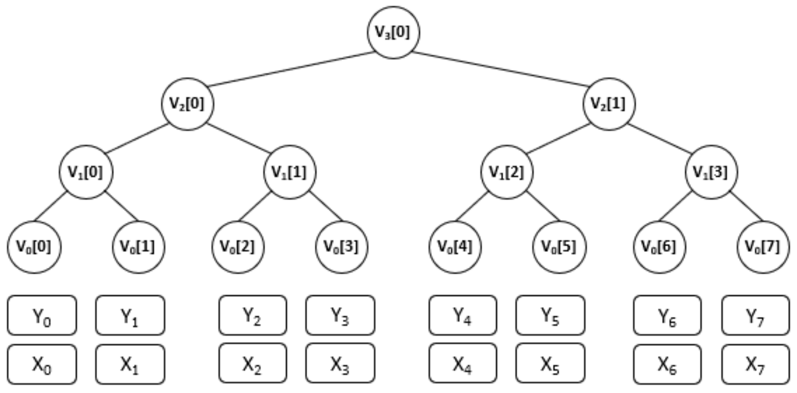

Fig. 1. Merkle tree height H=3

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

It is not necessary to store the full hash tree to compute the root of a Merkle tree. Instead, the tree hash algorithm is used. The basic idea of this algorithm is to sequentially compute the leaves and when we can compute their parents as well.

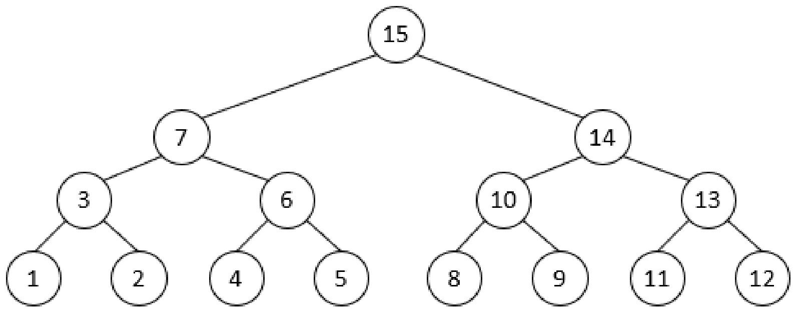

Fig. 2. Merkle tree, tree hash algorithm

This shows the order in which Merkle tree nodes are computed by the tree hash algorithm. In this example, the maximum number of nodes stored on the stack is 3. This happens after node 11 is created and pushed onto the stack. To compute the root of a Merkle tree of height H, the tree hash algorithm requires 2^H calls, and $2^H - 1$ evaluations of the hash function.

MSS successfully uses one-time signing keys for signature generation. To sign a message on M, we must first compute the n-bit $d = g(M)$. The signer then generates a one-time signature $sign_{\mathrm{OTS}}$ using the s-th one-time signature key $X_s, s \in \{0, \dots, 2^H - 1\}$. A Merkle signature contains this one-time signature and the corresponding one-time verification key Y_s. To prove the authenticity of Y_s, the signer also appends the index s and the authentication path to the verification key Y_s. This index and the authentication path allow the verifier to construct a path from the leaf $g(Y_s)$ to the root of the Merkle tree. A node h in the authorization path is a sibling node of height h, which is the path from the leaf $g(Y_s)$ to the root of the Merkle tree.

For $h = 0, \dots H - 1$ figure 3 shows an example of s=3. So, the s-th merkle signature is

$$sign_s = \left(s, sign_{\mathrm{OTS}}, Y_{s,}, (sign_0, \dots, sign_{H-1})\right) \tag{5}$$

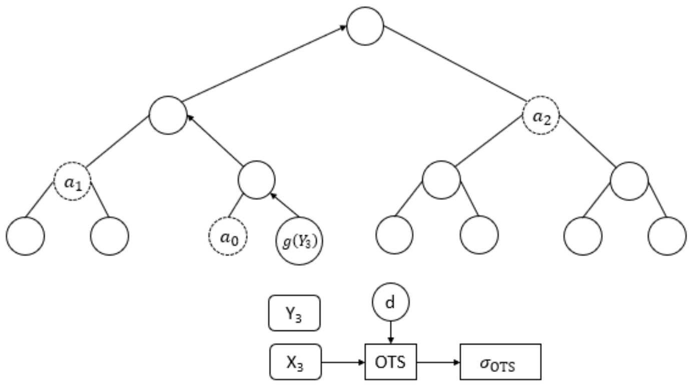

Fig. 3. Generation of Merkle's signature when $s = 3$.

The dashed nodes denote the authentication path of the leaf $g(Y_3)$. Arrows indicate the path from the leaf $g(Y_3)$ to the root.

Merkle's signature verification involves two steps. In the first step, the verifier uses the one-time verification key Y_s to verify d's signature $sign_{\mathrm{OTS}}$ by means of the corresponding one-time signature scheme verification algorithm. At the second stage, the verifier checks the reliability of the one-time verification key Y_s.

Merkle Trees are computationally fast, and a Merkle Tree over n nodes can be constructed in O(n) time. Merkle Tree that contains many nodes can have Merkle proofs that are then prohibitively large. To sign 2^n messages the height of the tree must be n. The Merkle Proof itself might put a significant and expensive strain on our local storage.

5. Merkle Tree vs Verkle Tree

Verkle trees are a powerful upgrade to Merkle trees that allow for much smaller verifications and are more efficient. The structure of the Verkle tree is very similar to the Merkle Patricia tree [11].

The main idea of the Verkle Tree is that we can build a Merkle Tree but substitute Cryptographic Hash functions with Vector Commitments. First, we decide how many pieces to divide our tree into, k. Then compute a Verkle Tree across a number of files, including f_0, f_1, ..., f_n. Then, after dividing our files into k subgroups, we compute a Vector Commitment, or *VC*, over each of the subsets of files. Additionally, we determine each Vector Commitment membership proofs PR_i with regard to *VC* for each file fi in the subset. Following that, we compute Vector Commitments across previously computed commitments up the tree until we compute the root commitment [12].

In Fig 4, we have 9 files and branching factor k = 3. We divide the files into subsets of size $k = 3$, a Vector Commitment is computed over each subset along with the corresponding membership proofs. This leaves us with the commitments VC_1, CV_2, and VC_3. We compute the Vector Commitment VC_4 over these three commitments along with the membership proofs PR_9, PR_{10}, and PR_{11} for the commitments VC_1, VC_2, and VC_3 respectively with respect to the commitment VC_4. The digest of the Verkle Tree is the root commitment, which is VC_4 in this case.

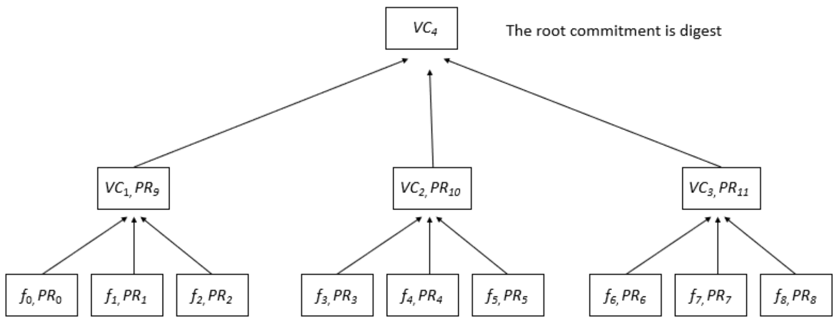

Fig. 4. A verkle tree when $K = 3$

As previously stated, the Verkle tree is an improved variant of the Merkle tree. Both sorts of trees have unique characteristics, especially when it comes to offering Merkle and Verkle proofs. The whole set of sister nodes in a Merkle tree, including Merkle Patricia trees, constitutes the evidence of a value. The proof must include all nodes in the tree that have any parent node in common with the node you are attempting to prove.

Verkle tree differs from Merkle tree in that you simply need to provide the path plus a small amount of additional information as proof - you don't even need to include sibling nodes. This is why Verkle trees profit from larger width while Merkle Patricia trees do not: in both situations, a tree with greater width leads to shorter routes, but in a Merkle Patricia tree, this effect is overcome by the increased cost of needing to provide all the width -1 sister node per level in a proof. This cost is absent in a Verkle tree.

In a verkle tree, an inner node is computed from its children using a hash function other than a standard hash. A vector commitment is used instead. This small parameter is what we'll use as proof. The primary proposition of the Verkle tree is that a Merkle tree can be obtained by replacing the cryptographic hash functions with vector commitments. Similar to a Merkle tree, a Verkle tree accomplishes the same thing. The main difference is that they are substantially more efficient in terms of size in bytes.

6. Vector Commitments

Instead of committing to individual messages, Vector Commitment (VC) allows users to commit to an ordered list of q values (i.e., a vector). This is done in such a way that it will later be feasible to open the commitment with respect to particular positions (for example, to prove that m_i is the i-th committed message). To be more specific, vector commitments are necessary to meet position binding. An adversary should not be able to open a commitment to two separate values at the same location, according to the concept of position binding. We require VCs to be concise, which means that the length of the commitment string and the size of each opening must be independent of the vector length [13].

Vector commitments may additionally need to possess the "hiding property," which states that even after viewing certain openings, it should be impossible to tell if a commitment was made to the vectors $(m_1, \dots, m_q)$ or to $(m'_1, \dots, m'_q)$. The implementation of vector commitments does not, however, depend much on the hiding property.

Furthermore, required is the ability to update Vector Commitments. They are provided with two algorithms to update the commitment and the accompanying openings, to put it crudely. The first approach enables the committer to acquire a (modified) Com' containing the updated message when updating a commitment Com by altering the i-th message from m_i to m'_i. Holders of an opening for a message at position j with respect to Com may update their evidence according to the second method in order to make it legitimate with respect to the new Com'.

We can make advantage of the conventional and well-established RSA assumption to realize Vector Commitment. Compact and effective solutions made possible by vector commitment significantly outperform earlier studies in terms of the effectiveness of the resulting solutions, the "quality" of the underlying assumption, or both [14].

7. Vector Commitments based on RSA

If an integer N is the product of two different prime numbers p, q, it is known as the RSA modulus. We have a random number $z \in \mathbb{Z}_N$, a public RSA modulus N, a public exponent e, and god $(\epsilon, \phi(N)) = 1$. To solve the RSA problem, one must determine the one and only $y \in \mathbb{Z}_N$ such that $z = y^e \mathrm{mod}\, N$. The public exponent e can be chosen based on multiple distributions, and different distributions result in different variants of the problem. We take into account the RSA problem where e is picked at random from a $(l + 1)$-bit prime (for some parameter l). Formally speaking, the related RSA assumption reads as follows.

We have RSA Assumption. Let N be a random RSA modulus of length k, z be a random element in $\mathbb{Z}_N$, and e be a $(l + 1)$-bit prime (for some parameter l), $k \in \mathbb{N}$ is security parameter. The RSA assumption is then considered to be valid if for each PPT adversary A, the probability

$$\Pr\,[y \leftarrow A(N, e, z) \colon y^e = z \bmod N] \tag{6}$$

is negligible in k.

We offer vector commitment realization based on the RSA assumption. It can be described with following algorithms:

VC.KeyGen $(1^k, l, q)$ Randomly choose two $k/2$-bit primes p_1, p_2, $N = p_1 p_2$, and then choose q $(l + 1)$-bit primes $\epsilon_1, \dots, \epsilon_q$ that do not divide $\phi(N)$. For $i = 1$ to q set

$$S_i = a^{\Pi} \Pi_{j=1, j \neq i}^{e} e_j \tag{7}$$

The public parameters pub is $(N, a_1 S_1, \dots, S_q, e_1, \dots, e_q)$. The message space is $M = \{0,1\}^{\ell}$

VC.Com_{pub}$(m_1, \dots, m_q)$ Compute

$$C \leftarrow S_1^{n_1} \cdots S_q^{m_q} \tag{8}$$

and output C and the auxiliary information aux $= (m_1, \dots, m_q)$.

VC.Open $_{pub}(m, i,$ aux), Compute

$$A_i \leftarrow \sqrt[5]{\prod_{j=1, j \neq i}^{q} S_j^{m_j}} \bmod N \tag{9}$$

Notice that knowledge of pp allows to compute A_i efficiently without the factorization of N.

VC.Ver_{pub}(C, m, i, Λ_i) The verification algorithm returns 1 if $m \in M$ and

$$C = S_i^m \Lambda_i^{e_i} \bmod N \tag{10}$$

Otherwise it returns 0.

VC.Update $e_{pub}(C, m, m', i)$ Compute the updated commitment $C^t = C \cdot S_i^{m^r - m}$. Finally output C' and $U = (m, m', i)$.

VC.ProofUpdate $_{\mathrm{pUB}}(C, \Lambda_j, m', i, U)$ A client who owns a proof Λ_j, that is valid with respect to to C for some message at position j, can use the update information U to compute the updated commitment C' and to produce a new proof Λ'_j which will be valid with respect to C'.

We distinguish two cases:

1. $i \neq j$. Compute the updated commitment as $C' = CS_i^{m'-m}$ while the updated proof is $\Lambda'_j = \Lambda_j \sqrt[e]{S_i^{m'-m}}$ (notice that such e_j-th root can be efficiently computed using the elements in the public key).
2. $i = j$. Compute the updated commitment $C' = C \cdot S_i^{m'-m}$ while the updated proof remains the same A_i.

In order for the verification process to be correct, notice that one should also verify (only once) the validity of the public key by checking that the S_i's are correctly generated with respect to a and the exponents $e_1, \dots, e_q$.

A drawback of this scheme is that the size of the public parameter pp is $O(q^2)$, the size of the public parameters is linear in q. This can be important in scenarios where big datasets are used with vector commitment. Also, this construction can be easily optimized in such a way that the verifier stores only a constant number of elements, instead of the entire public parameters. The signature is computed on (i, S_i, e_i).

8. Novel scheme using Verkle Tree

Because a different key pair must be used to sign each message, one-time signature schemes are particularly demanding to employ. These systems have the drawback of requiring n digests to be stored. We would like a system that enables us to save a uniform-sized digest regardless of the number of files we have because it is prohibitive for regular use. The Merkle Tree was suggested as a solution to this issue. This method can replace numerous verification keys with a single public key by employing a binary tree as the root.

A Merkle Tree with n nodes can be built in O(n) time because Merkle Trees are computationally quick. A Merkle tree with many nodes can result in prohibitively huge Merkle proofs. To sign 2^n messages the height of the tree must be n. Our local storage may experience a large and costly strain as a result of the Merkle Proof itself.

Merkle proofs can be greatly improved by Verkle trees, which enable significantly reduced proof sizes. The verifier simply needs to offer a single proof that demonstrates all parent-child ties between all commitments along the paths from each leaf node to the root, as opposed to having to present all "brother nodes" at every level. When compared to optimal Merkle trees, proof sizes can be reduced by a factor of 6–8, and when compared to Merkle Patricia trees, proof sizes can be reduced by a factor of more than 20–30.

Instead of the Merkle tree, we use the Verkle tree.

When generating the key pair, the signer chooses $H \in \mathbb{N}, H \geq 2$. Then the key pair is generated. Using them, it will be possible to sign/verify 2^H documents. The signer will generate 2^H unique key pairs $(X_j, Y_j), 0 \leq j < 2^H$. Here, X_j is the signature key and Y_j is the verification key. Both of them are bit strings. The leaves of the Verkle tree are $g(Y_j)$, $0 \leq j < 2^H$. They are computed and used as the leaves of the tree and each node in the tree is a hash value of of its children's concatenation. The public key of the Verkle crypto system is the root commitment. To generate public key 2^H pairs of keys must be computed.

We can use one-time signing keys signature generation. To sign a message on M, we must first compute the n-bit $d = g(M)$. Firstly, arbitrary size message m is transformed into size n by means of the hash function. Document's signature will be the concatenation of: one-time signature, one-time verification key, index s for the proof and the root commitment.

In Verkle's signature verification, verification is done as follows: the one-time signature of $sign$ should be verified using Y_s. After this, if it is true, the commitments VC_i are verified. If the root of the tree is equal to root commitment, the signature is verified. In verkle tree root commitment is digest.

9. Conclusion

We discussed currently available random number generation tools for both classical and quantum cases. The post-quantum cryptography systems were covered. We discussed hashing-based one-way functions, their integration into Merkle. We discussed vector commitment and commitments based on RSA assumption. We discussed powerful upgrade of the Merkle tree - Verkle tree, how it is computed and integrated. Novel shames, their efficiency, created a new model and integrated it into Verkle. They do require more complex cryptography to implement, but they present the opportunity for large gains to scalability.

It is important for us that the resulting schemes protect us from both classical and quantum computer attacks. After reviewing the tasks performed, we got the systems that are integrated into Merkle. Merkle Trees, which are built using cryptographic hash functions, are a good solution to protect against quantum attacks, but the verification size is too large. A parent node in a Merkle tree is the hash of its children. We have an improvement Verkle Tree, where a parent node in a Verkle Tree is the vector commitment of its children. To implement the new technology, we discussed vector commitment and commitments based on RSA assumption.

The Verkle scheme is a powerful upgrade of the Merkle scheme that allows for much smaller verifications. Instead of providing all nodes at each 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 compared to the classical Merkle scheme.

In our updated schemes we used Verkle tree instead of Merkle tree. In this case, only a small parameter, vector commitment, is needed to use as proof. We used vector commitments based on the RSA assumption to build the Verkle tree.

As previously stated, it is critical that the resulting methods defend us from attacks by quantum computers. Unfortunately, so far Vector commitments based on RSA can be broken by quantum computers. We are improving the scheme to make it more secure and efficient. In the future, we plan to create electronic signature schemes that will use verkle trees, but we will use lattices to build vector commitments. Our schemes will be based on post-quantum assumptions.

10. Acknowledgement

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

11.References

[1] Chen, Lily, et al. Report on post-quantum cryptography. Vol. 12. Gaithersburg, MD, USA: US Department of Commerce, National Institute of Standards and Technology, 2016.

[2] Buchmann, J., Dahmen, E., Szydlo, M. (2009). Hash-based Digital Signature Schemes. In: Bernstein, D.J., Buchmann, J., Dahmen, E. (eds) Post-Quantum Cryptography. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88702-7_3

[3] Biswas, Bhaskar, and Nicolas Sendrier. "McEliece cryptosystem implementation: Theory and practice." Post-Quantum Cryptography: Second International Workshop, PQCrypto 2008 Cincinnati, OH, USA, October 17-19, 2008 Proceedings 2. Springer Berlin Heidelberg, 2008.

[4] Gagnidze, Avtandil, Maksim Iavich, and Giorgi Iashvili. "Novel version of merkle cryptosystem." Bulletin of the Georgian National Academy of Sciences (2017).

[5] Iavich, Maksim, et al. "ADVANTAGES AND CHALLENGES OF QRNG INTEGRATION INTO MERKLE." Scientific and practical cyber security journal (2020).

[6] Lamport, Leslie. "Constructing digital signatures from a one way function." (1979).

[7] Post-Quantum Digital Signatures with Attenuated Pulse Generator; M. Iavich, R. Bocu, A. Arakelian, G. Iashvili; ceur-ws.org, Vol-2698, 2020.

[8] Iavich, Maksim, et al. "Improvement of Merkle signature scheme by means of optical quantum random number generators." Advances in Computer Science for Engineering and Education III 3. Springer International Publishing, 2021.

[9] Iavich, Maksim, Avtandil Gagnidze, and Giorgi Iashvili. "Hash based digital signature scheme with integrated TRNG." CEUR Workshop Proceedings. 2018.

[10] Merkle, Ralph C. "A digital signature based on a conventional encryption function." Advances in Cryptology—CRYPTO'87: Proceedings 7. Springer Berlin Heidelberg, 1988.

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

[12] Weijie Wang, Yale University Annie Ulichney, Yale University Charalampos Papamanthou, Yale University, BalanceProofs: Maintainable Vector Commitments with Fast Aggregation, Cryptology ePrint Archive, 2022

[13] Kurosawa, Kaoru, and Goichiro Hanaoka, eds. Public-Key Cryptography--PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, Feburary 26--March 1, 2013, Proceedings. Vol. 7778. Springer, 2013.

[14] Kuszmaul, John. "Verkle trees." Verkle Trees 1 (2019).