Implementation and comparison of hash-function based multi-time digital signature protocols

Anatoly Anisimov[1] and Oleksii Bashuk[1,*]

[1] *Taras Shevchenko National University of Kyiv, 02000 Kyiv, Ukraine*

Abstract

In the rapidly advancing field of information technology, the arrival of portable, general-purpose quantum computers is on the horizon. While these quantum machines promise significantly higher efficiency compared to classical computers, they also introduce substantial vulnerabilities to current cryptographic and cybersecurity algorithms. To address this impending challenge, the development of post-quantum protocols has become a crucial area of research.

Digital signature schemes are particularly at risk, prompting the exploration of quantum-resistant methods based on hash functions. One-time signature schemes, such as the Lamport signature, offer security against quantum attacks but are limited by their single-use nature. To overcome this drawback, multi-time digital signature schemes have been developed, utilizing multiple one-time signatures to enable repeated use. Signature schemes like chain-based and tree-based schemes have become the most popular in this domain.

This work concentrates on the implementation and comparative analysis of various multi-time digital signature schemes. Through practical implementation and measurement, we clearly demonstrate the advantages and limitations of each scheme. Our findings reveal that tree-based schemes, especially those constructed sequentially, offer superior performance. However, in specific scenarios, fully pre-constructed trees may be more advantageous.

Given the relative novelty and limited exploration in this field, multi-time digital signatures offer ample opportunities for further research. Future work could investigate the use of N-ary trees and compare their efficiency and security properties with those of binary trees.

Keywords

hash function, digital signatures, post-quantum protocols, one-time digital signatures, multi-time digital signatures, chain-based schemes, tree-based schemes

1. Introduction

Technological progress continues to advance, and quantum computers are set to play a key role in the future. However, one of the greatest risks of their implementation is the vulnerability of current cryptographic algorithms, which are based on the computational limitations of classical binary computers. Traditional cryptosystems rely on problems like integer factorization or discrete logarithms, which can be easily solved by sufficiently large quantum computers. Considering this, research in post-quantum cryptography has become increasingly relevant in recent years. These protocols are independent of quantum computations and, thus, resistant to quantum attacks. For example, hash-based protocols may remain secure against quantum computing threats.

Digital signatures are no exception to this trend. One of the foundations of post-quantum signature protocols has become different one-time signatures (OTS), such as the Lamport OTS [1], the Merkle OTS [2], the Winternitz OTS [2], the Bleichenbacher-Maurer OTS [3], the BiBa OTS [4], and the HORS [5]. However, as the name suggests, these signatures are for one-time use only, which

Information Technology and Implementation (IT&I-2024), November 20-21, 2024, Kyiv, Ukraine

* Corresponding author.

a.v.anisimov@knu.ua (A. Anisimov); a.bashuk@knu.ua (O. Bashuk)
0000-0002-1467-2006 (A. Anisimov); 0009-0002-6778-2943 (O. Bashuk)

is their main drawback. A logical desire arises to combine multiple one-time signature keys into one. One solution is the so-called Chain-Based Signature (CBS)[6, pp. 465-468]. In these schemes, some one-time signature protocol is used as the basis. When signing a message, not only the message itself is signed but also the public key of a newly created one-time signature. During the next signing, this new signature will be used to sign a new message, along with the public key of yet another freshly created signature. In this way, a chain of signatures is built. To verify any signature, the entire chain, along with the corresponding public keys and the signed input messages starting from the initial signature to the current one, must be provided. Using the initial public key, the authenticity of each signature in the chain, including the target one, can be easily verified. Essentially, the "signature" of a message is considered not just the last one-time signature made in the chain, but the entire current state of the chain along with the last signature. However, this approach has an obvious drawback: with each new signature, both the size of the signature and the time required to verify it grow.

To reduce the size of the signature, a combination of the chain-based signature and Merkle tree [6, pp. 183-184] ideas was proposed. So, to construct a so-called Tree-Based Signature (TBS) [6, pp. 468-473], it has to be built a binary tree, where each node contains signatures of its child nodes. For this purpose, key pairs are generated for both child nodes (if not already created), and then the public keys of these child nodes are signed. Additionally, a binary string "a" consisting of 0s and 1s is assigned to each node, such that the child nodes of the current node will have strings "a0" and "a1," while the root corresponds to an empty string. Nodes, where the length of the string equals the bit-length of the input messages, are considered as leaves and are used to sign these input messages. However, the entire tree of height 256 is too large to store in memory, so this tree is not fully built from the beginning, and it grows over time. The public key of this signature tree is the public key of the root node. When signing a new input message, the algorithm goes through the tree from the root, ensuring that the string of each node is a prefix of the binary representation of the input message. If a node does not have children and is not a leaf, key pairs for one-time signatures are created for the node, and the public keys of these child nodes are signed using the current node's private key. If the node is a leaf, the corresponding input message is signed using the leaf's private key. If the message has already been signed, the stored value is reused. As a result, the signature in such a tree consists of the actual one-time signature of the input message in the leaf, all one-time signatures in the nodes along the path from the leaf to the root, and the public keys of the corresponding child nodes. For verification, it is enough just to check that all signatures on the path are valid and that the signature of the input message is correct.

With this construction, all signatures will have a fixed length corresponding to the bit length of the input messages. Therefore, after a certain point, the length of one tree-based signature and the time required for its verification will be better than those of chain-based signatures. It's also important to note that with this structure, previously signed input messages do not appear in the signature of new input messages, which is more secure. Moreover, unlike a Merkle tree, there is no need to build the entire tree upfront, which eliminates the explicit limit on the number of one-time signatures.

However, storing this tree requires more memory than storing a chain-based signature and leads to the storage capacity limitation of the device to store the current state of the tree. But this design was proposed for a specific reason. With this construction, the input message is not actually used in signing – it is used to choose which one-time signatures must be provided. Even to sign the leaf, the input message is not needed as it is equal to the leaf's node string. At the same time, all key generations are pseudorandom and can be predicated with the known seed. According to this, it is not needed to store the state of the tree, as each path to any leaf with all signings and keys can be constructed 'on the fly' from the known seed.

But if the memory usage is not a high priority, (If the inclusion of previous input messages in the current message signature is not an issue,) an alternative tree structure with theoretically better productivity can be built. While the previously proposed tree is constructed similarly to a depth-first search algorithm, reaching the leaves from the root of a fixed-size tree, it is also possible to build a tree in a manner akin to a breadth-first search algorithm. This approach, which can be called the Sequential Tree-Based Signature (STBS), constructs nodes sequentially based on their distance from the root. In this scheme, with each new message signature, the algorithm chooses the next node in the queue, creates two child nodes with new one-time signatures for each, and adds those to the end of the queue. The public keys of the newly created child nodes, along with the input message, are signed using the private key of the current node. Consequently, the signature in this tree consists of all the signatures along the path from the current node to the root, along with the public keys of the child nodes on the way and the corresponding previous input messages. This construction significantly reduces the number of nodes involved in a single signature, resulting in fewer one-time signature verifications. This should positively impact both the signature and verification times, compared to the previously proposed tree. However, the signature will also include previously signed input messages, increasing each signature segment's size.

If the inclusion of previous input messages in the current message signature is an issue, there is another alternative approach that is similar to the previous method but solves the issue. In this approach, instead of having two child nodes, each node will have three, creating a Wide Sequential Tree-Based Signature (WSTBS). The left and right child nodes maintain their usual functionality and are used for further tree construction, while the central node serves as a leaf and is exclusively used to sign the current message. Thanks to this additional leaf generation for signing messages in every regular node of the tree, there is no longer a need to include corresponding intermediate messages in the signature verification process. Instead, the public keys of these leaves are provided to verify intermediate signatures. As a result, previously signed messages are excluded from the signatures, but 50% more one-time signatures are generated, which impacts both the signature time and the overall tree size. This method removes the need to reference past messages during verification, simplifying the signature, but at the cost of generating a larger tree and consuming more computational resources for the increased number of one-time signatures.

This work is aimed at developing implementations of various types of multi-time signatures based on one-time signatures, as well as comparing their performance.

2. Implementation results

All measurements and research were conducted on a 2021 MacBook Pro with an Apple M1 Pro processor and 32 GB of RAM. As a basic hash function was chosen SHA-256 [7], and as a basic one-time signature scheme for all multi-time signature schemes was chosen the Lamport signature scheme. You can find the link to the code implemented during the experiments in the Online Resources section.

To measure the performance of different schemes, 2560 bits size messages were generated and, after that, were signed and verified by different schemes. The measurements include signature time, signature and sign copying time, verification time, signature size, and overall memory usage. Due to the varying asymptotic complexity of the functions of different schemes, the number of messages used for the measurements varied. Specifically, CBS was tested with 1000 messages, TBS with 10259, STBS with 83550, and WSTBS with 59052. All collected data can be seen in the following graphs, where CBS is in blue, TBS in orange, STBS in green, and WSTBS in red (Figure 1).

The memory usage metrics are as expected since with each signature (except TBS), the structure storing the current state of signatures (list or tree) grows linearly by the same amount. Let's examine other measurements in more detail.

2.1. Signature time

In both signature time graphs, there are noticeable sequential spikes that increase over time. However, the only difference between the graphs is the additional time spent on fully copying the signature, and the graph with copying shows significantly more spikes. This can be easily explained by the computer's internal processes on which the measurements were conducted, which caused this "noise." Specifically, most of these spikes are due to freeing up RAM for the structure that stores the signatures. After removing the noise, we will have (Figure 2).

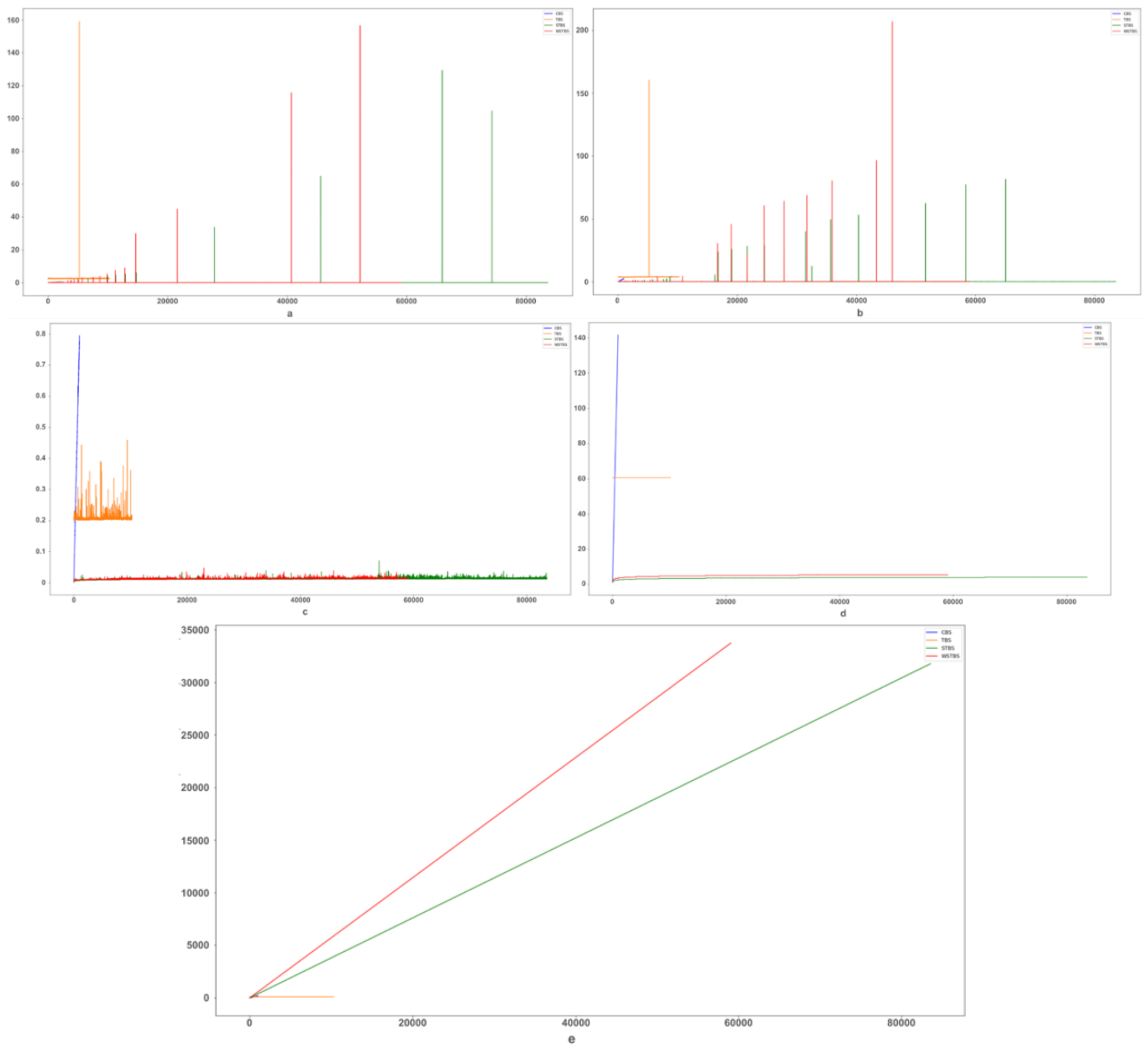

Figure 1: graphs of all collected measurements that include a) signature time (sec.), b) signature and sign copying time (sec.), c) verification time (sec.), d) signature size (MB), and e) overall memory usage (MB).

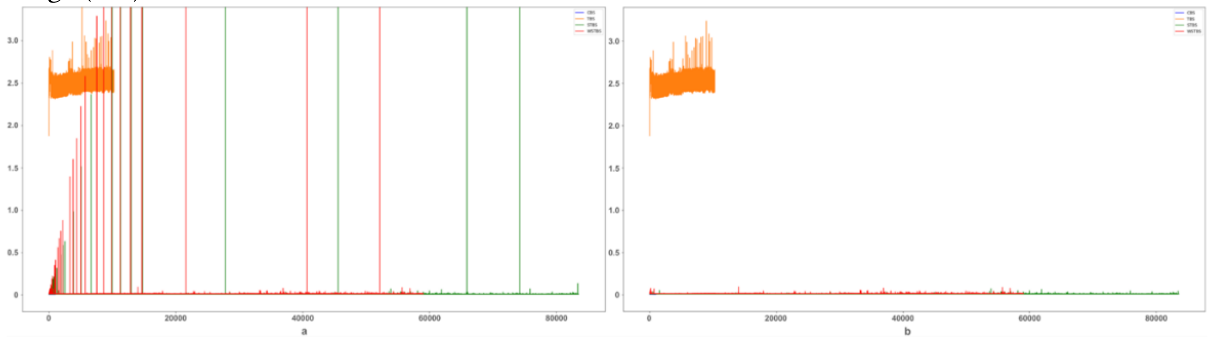

Figure 2: Noise remove from signature time (sec.) data, where a) before, and b) after.

The time required for signing in the TBS scheme is expected to be close to constant. If we examine all other schemes in more detail and slightly average the data, we get:

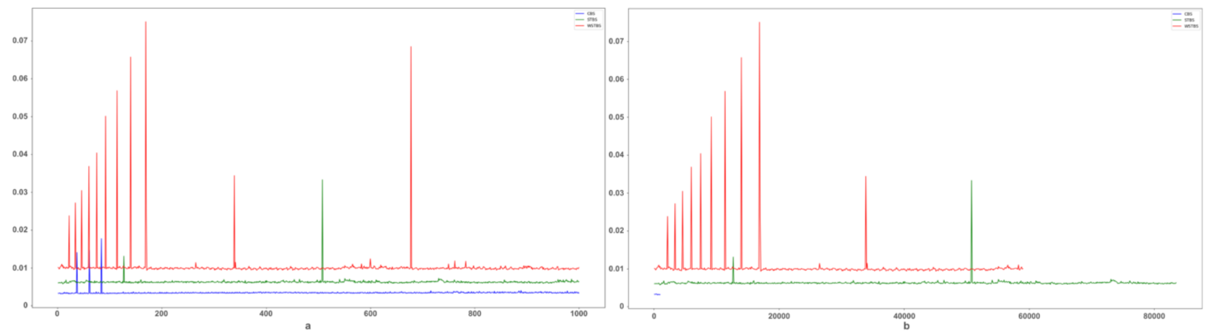

Figure 3: signature time (sec.) of CBS, STBS, and WSTBS on a) CBS interval and b) full interval.

The graphs still show some "noise", but except that, the time remains close to constant, and maintains a relationship according to the number of key generations for a single signing operation.

2.2. Verification time

Let's look at the verification time:

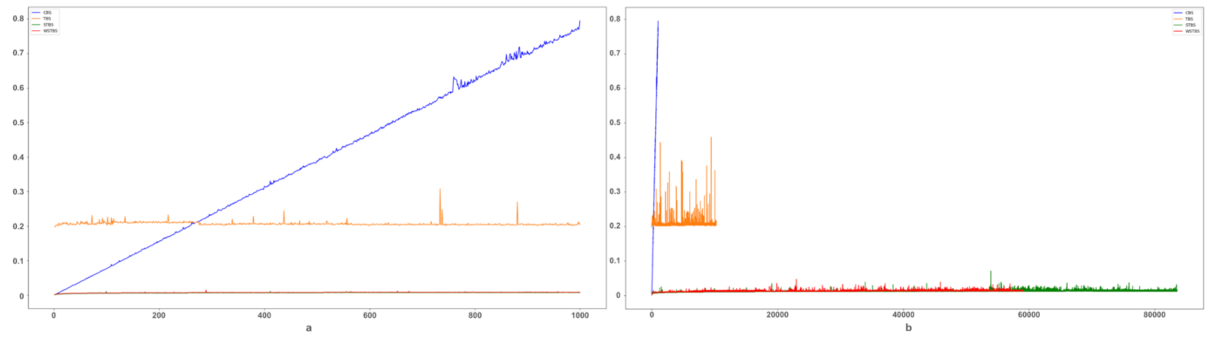

Figure 4: Verification time (sec.) on a) CBS interval, and b) full interval.

In the CBS scheme, the verification time grows linearly, while in the TBS scheme, the time remains close to constant, which fully aligns with the expected asymptotic. When examining the STBS and WSTBS schemes separately, we observe:

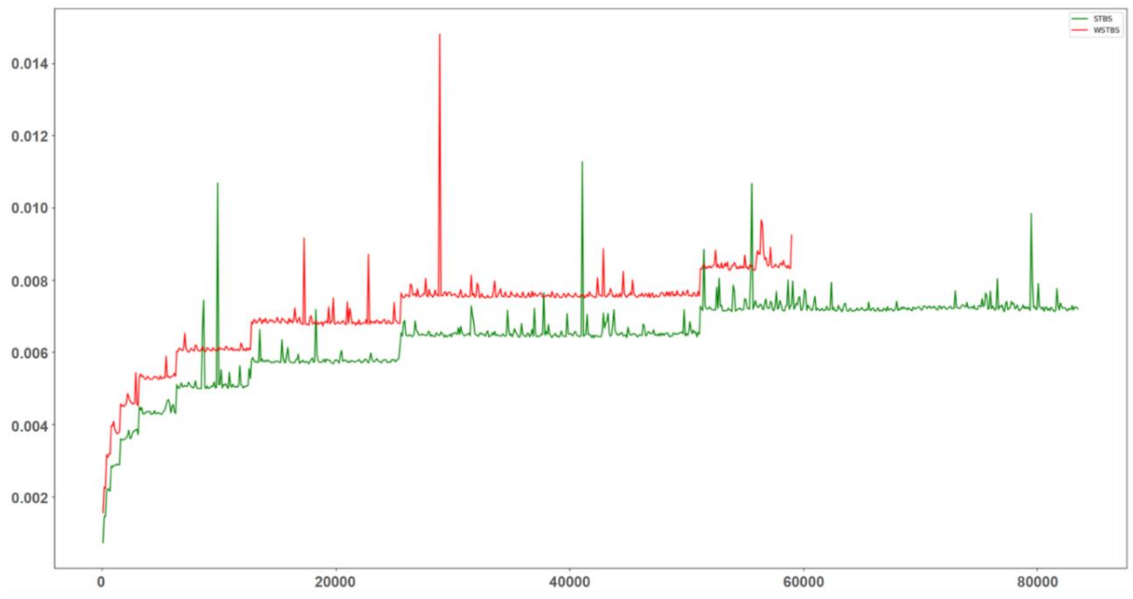

Figure 5: Verification time (sec.) of STBS and WSTBS.

Both graphs exhibit logarithmic asymptotics with clearly defined "steps" of equal height, corresponding to the increase in signature length by one link due to new layers in the signature tree states.

2.3. Signature size

Let’s look at the signature size:

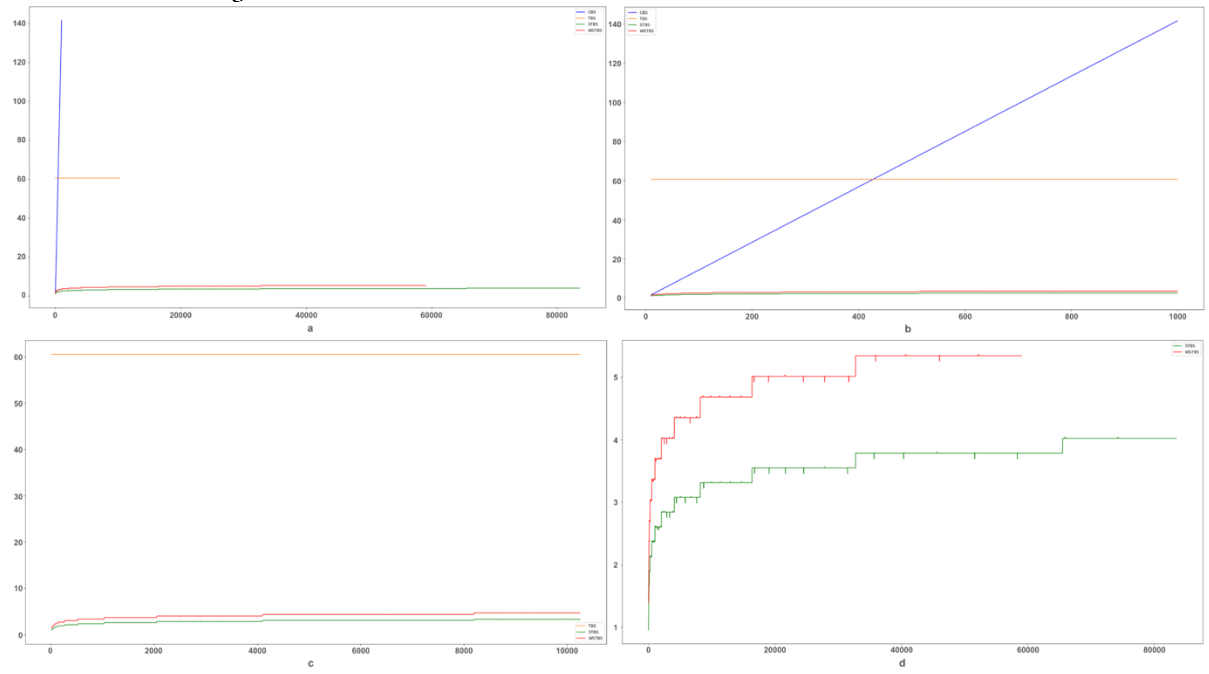

Figure 6: Signature size (MB) of a) all signature schemes on the full interval, b) all signature schemes on the CBS interval, c) different tree-based signature schemes, and d) STBS and WSTBS.

Again, there is full alignment with expected asymptotics. The last graph also shows a small amount of "noise" as well as "steps" of equal height, corresponding to the increase in signature length by one link.

2.4. General analysis

Based on the graphs, there is a clear advantage of different tree-based signatures over chain-based signatures. Furthermore, the STBS and WSTBS schemes clearly outperform the basic TBS scheme by several orders of magnitude across almost all measured metrics. However, it's important to consider the potential of the TBS scheme, even though it mainly affects the memory required to maintain the state of the signatures.

3. Conclusion

This work explored post-quantum digital signature methods. Specifically, various methods of constructing multi-time signature schemes based on the Lamport signature scheme as a one-time signature scheme were examined.

The results indicate that the best methods for constructing multi-time signatures among those considered are the schemes based on binary tree constructions. However, these schemes have their own advantages and disadvantages. For instance, if time efficiency and small signature size are not primary goals, it is more advantageous to use the TBS scheme. Conversely, if those factors are critical, it is better to use the STBS or WSTBS schemes.

4. Further investigation

The consideration of alternative data structures, such as ternary and N-ary trees, is indeed a logical next step. While these may not change the asymptotic complexity, they could potentially enhance the final results by optimizing performance or reducing overhead in specific cases. Exploring these options could provide additional insights into improving the efficiency and effectiveness of multi-time signature schemes.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] D. Boneh, V. Shoup, A Graduate Course in Applied Cryptography, 2017, pp. 569-579. URL: https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_4.pdf.

[2] Ralph C. Merkle, A certified digital signature, In Conference on the Theory and Application of Cryptology, Springer, 1989, pp. 218–238. URL: doi:10.1007/0-387-34805-0_21.

[3] Daniel Bleichenbacher and Ueli M. Maurer, Directed acyclic graphs, one-way functions and digital signatures, In Advances in Cryptology - CRYPTO '94, volume 839 of Lecture Notes in Computer Science, 1994, pp, 75–82.

[4] Adrian Perrig, The BiBa one-time signature and broadcast authentication protocol, In ACM Conference on Computer and Communications Security, 2001, pp. 28–37.

[5] Leonid Reyzin and Natan Reyzin, Better than BiBa: Short one-time signatures with fast signing and verifying, In Lynn Batten and Jennifer Seberry, editors, Information Security and Privacy, volume 2384 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2002, pp. 1–47.

[6] J. Katz, Y. Lindell, Introduction to Modern Cryptography, Second Edition, 2015. URL: https://eclass.uniwa.gr/modules/document/file.php/CSCYB105/Reading%20Material/%5BJonathan_Katz%2C_Yehuda_Lindell%5D_Introduction_to_Mo%282nd%29.pdf.

[7] National Institute of Standards and Technology (NIST), FIPS PUB 180-4: Secure Hash Standard (SHS), 2012. URL: https://luca-giuzzi.unibs.it/corsi/Support/papers-cryptography/fips-180-4.pdf.

A. Online Resources

Code implementation can be found at https://github.com/albashuk/Dissertation/tree/master/Experiments/Python.