P. Juaristi and I. Agudo and R. Rios and L. Ricci, "Benchmarking post-quantum cryptography in Ethereum-based blockchains", *8th International Workshop on Cryptocurrencies and Blockchain Technology (CBT 2024)*, In press.
NICS Lab. Publications: `https://www.nics.uma.es/publications`

Benchmarking post-quantum cryptography in Ethereum-based blockchains

Patxi Juaristi[1], Isaac Agudo[1], Ruben Rios[1], Laura Ricci[2]

[1] NICS Lab, Universidad de Málaga, Spain
[2] Università di Pisa, Italy

Abstract. Blockchain technology has significantly transformed various industries by enabling secure and tamper-resistant transactions. However, the rise of quantum computing threatens the cryptographic foundations of blockchain networks, making blockchain vulnerable to signature forgery and transaction manipulation. This raises concerns about the long-term viability of blockchain systems and highlights the need for post-quantum secure solutions.
This paper investigates the feasibility of quantum-resistant blockchain ecosystems. Our research focuses on estimating the cost of the integration of the post-quantum algorithms selected in the NIST standardization competition into Ethereum-based blockchains.

Keywords: Blockchain · post-quantum cryptography · digital signature · Ethereum · ECDSA · Dilithium · Falcon · SPHINCS$^+$.

1 Introduction

In recent years, blockchain technology has emerged as a revolutionary technology for secure and decentralized data management, largely due to its ability to provide an immutable and transparent data ledger, which cannot be manipulated. This technology has been adopted in various sectors, with finance being the main application scenario, but also extending to supply chain management and many others.

However, the advent of quantum computers poses a major threat to the foundations of blockchain technology as they could undermine the security of current cryptographic algorithms. Leveraging the principles of quantum mechanics, quantum computers can perform a large number of calculations simultaneously because their basic unit of information representation, the quantum bit or qubit, can exist in a superposition of states. This allows multiple states to be represented at the same time, greatly facilitating efficient parallel processing [1].

As such, quantum computers are capable of solving complex mathematical problems considerably faster than classical computers.

The potential of quantum computing is significantly enhanced by the application of specific algorithms, such as Shor's algorithm [2] for factoring large numbers, and Grover's algorithm [18] for speeding up searches in unstructured data.

These algorithms have direct implications for the security of current cryptographic algorithms, especially public-key algorithms. For example, the security of RSA is based on the impossibility of factoring large prime numbers, but with Shor's algorithm a 2048-bit RSA key could be factorized in approximately 8 hours using a quantum computer with 20 million qubits [13].

While this is a clear threat to security, the reality is that we are still far from building quantum computers with this number of qubits. To the best of our knowledge, the largest quantum computers to date have just over a thousand of qubits. IBM's Condor, for example, has 1.121 qubits [14]. On such a computer, the factorization of the above mentioned RSA key would take approximately 142.204 hours, or about 5915 days. This value was obtained without differentiating between physical and logical qubits, because otherwise the result would be much higher [3].

Although quantum computers are not an immediate problem, the rapid development of them in recent years, has led to the need for new cryptographic algorithms that can withstand quantum computers. These new cryptographic algorithms are referred to as post-quantum or quantum-resistant algorithms.

Since blockchains rely on public-key cryptography to operate, they are also vulnerable to quantum computers. The main mechanism for interacting with blockchains is through transactions, which are digitally signed to ensure their authenticity. Most blockchains, including Bitcoin and Ethereum-based blockchains, use the Elliptic Curve Digital Signature Algorithm (ECDSA) [15] for this purpose. The main reason is that ECDSA uses short keys and produces short signatures.

In this paper we investigate the threat that quantum computers pose to blockchain systems and whether post-quantum cryptography (PQC) can be integrated into Ethereum-based blockchains to mitigate the risk. The main contribution of this paper is a performance comparison of the PQC algorithms selected from the NIST standarization process against ECDSA using real-time transaction data. It has been focused only on the ECDSA signature algorithm used to sign the Ethereum transactions, not in the BLS signature. The reason for this has been because replacing BLS is much more changeling given that Ethereum consensus uses BLS signature aggregation to store the results of the consensus voting. In the end, the solution has been composed of a modular and scalable system for acquiring, comparing and visualizing the results.

The rest of this paper is organized as follows. Section 2 provides a comparative analysis with related works. Next, Section 3 introduces the main existing post quantum cryptography families. The benchmarking architecture is described in Section 4. Subsequently, Section 5 shows the results obtained from the evaluation of applying ECDSA and NIST selected algorithms to real-time transaction data from an actual blockchain network. Finally, Section 6 present the conclusion and outlines potential lines of future research.

2 Related work

Concerns about the threat of quantum computers to current cryptographic schemes have led to notable research in this area, generating several solutions and surveys.

Regarding survey papers, Buser et al. [4] focus on exotic signature schemes for post-quantum blockchains, exploring the challenges associated with their implementation and proposing research directions. More recently, Yang et al. [21] presented a comprehensive survey and comparison of post-quantum and quantum blockchains, which highlighted the current state of research and identifying possible future directions.

Some papers focus on investigating the application of post-quantum blockchains to particular scenarios. For example, Chen et al. [5] concentrate on studying the practical implications of integrating post-quantum cryptographic schemes into blockchain systems designed for the Internet of Things (IoT) and other smart city infrastructures. Similarly, Yi et al. [22], discuss the application of post-quantum blockchain technology to secure the social Internet of Things (SIoT), proposing a framework that leverages post-quantum cryptography to ensure data integrity and privacy in these scenarios.

Additionally, some authors proposed the use of lattice-based cryptography to make blockchain networks resistant to quantum attacks. The focus of [11] is the creation of a cryptocurrency based on a post-quantum blockchain while [16] proposes a new signature scheme and describes how to apply it to secure blockchain transactions. None of them present a practical implementation.

To the best of our knowledge, the paper that is most similar to ours is [9]. This paper analyses the feasibility of post-quantum algorithms for the blockchain. However, their study does not use the finalists of the PQC competition organized by NIST. Furthermore, their evaluation is not done with real blockchain data.

In contrast, in this paper we provide a modular and scalable tool to facilitate the incorporation of novel post-quantum algorithms as they are developed. In addition, our solution uses real-time transaction data from a blockchain network, making the evaluation results more accurate.

3 Post-quantum cryptography families

The threat of quantum computing to today's cryptographic schemes has prompted the US National Institute of Standards and Technology (NIST) to launch a standardization process for post-quantum schemes, which aims to develop and standardize cryptographic algorithms that are resistant to quantum attacks [17]. The competition started in 2016 with 69 candidate algorithms and has progressed through rounds of evaluation. The candidates algorithms can be classified into different families, which are briefly described below.

Lattice-based cryptography relies on the mathematical properties of lattices, geometric structures formed by points in n-dimensional space. It involves solving problems such as the Shortest Vector Problem (SVP) and the Closest

Vector Problem (CVP) in these high-dimensional spaces, making brute-force attacks impractical. Lattices are often represented as arrays for easier matrix operations. The main cryptographic schemes include NTRUEncrypt and NTRUSign, which leverage SVP and CVP, and Ring-LWE-based schemes, which use the Ring Learning with Errors (RLWE) problem to perform computations on polynomial rings.

Code-based cryptography is one of the oldest and most studied approaches to post-quantum cryptography. Its security is based on the difficulty of solving the mathematical problems associated with error-correcting codes. These codes ensure reliable data transmission over noisy channels by introducing redundancy, which allows the receiver to detect and correct errors, while attackers without secret knowledge cannot decode the data. The McEliece cryptosystem, created in 1978, is the most famous code-based algorithm. It encrypts messages by encoding them in codewords and adding random errors to make decryption impossible without the private key. Although efficient and secure, the McEliece cryptosystem has large public keys and computationally intensive key generation and management processes.

Multivariate polynomial-based cryptography (MPKC) is a cryptographic scheme that exploits the hardness of solving systems of multivariate polynomial equations, which are computationally very difficult to solve, and have been proven to be NP-Complete, like some other lattice or code problems. These problems are of complexity class NP (non-deterministic polynomial time), which means that if a polynomial-time algorithm exists for solving any NP-complete problem, then polynomial-time algorithms exist for solving all problems in NP, making them essentially equivalent in difficulty.

Hash-based cryptography is a method of encrypting and securing data using hash functions. Before the process begins, it is necessary to determine which values are to be signed, and then to generate a long random string of characters for each of them. This resulting random string will be the private key used to sign the data. Once the private key is ready, this string is hashed using typical hash functions such as SHA-1, SHA-3, SHA-256 or BLAKE2, which produce strings between 256 and 512 bits, which are used as public keys to verify the signature. This is done by hashing the signature (the private key) and comparing it with the public key it has.

Isogeny-based cryptography creates cryptographic systems over finite fields using isogeny graphs of elliptic curves, which are mappings that preserve algebraic structures. An isogeny is a morphism between two elliptic curves, characterized by its homomorphism between the groups of points on the curves. Key generation involves choosing two elliptic curves over a finite field and generating a secret isogeny between them. The private key is the isogeny, while the public key is made of the starting and resulting curves. While the Supersingular Isogeny Diffie-Hellman (SIDH) protocol and its successor, Supersingular Isogeny Key Encapsulation (SIKE), were initially prominent, they have been found vulnerable to cryptographic attacks. As a result, newer isogeny-based schemes like SQISign and its variants have emerged. SQISign, which has been featured in recent NIST

post-quantum cryptography evaluations, represents a more robust approach, addressing the vulnerabilities of earlier protocols and offering promising security in the evolving landscape of post-quantum cryptography.

4 Benchmarking solution

This section describes the benchmarking tool developed to evaluate the performance of both current and post-quantum cryptographic algorithms in blockchain.

The tool consists of a set of interconnected functional blocks that result in a modular and scalable environment for the evaluation of cryptographic algorithms against real-time transactions from an Ethereum-based blockchain. The tool also provides an interface for the visualization and analysis of the results.

In the following, we present the overall architecture of the benchmarking solution and its building blocks in detail.

4.1 System architecture

The proposed solution consists of four main components as illustrated in Figure 1. The first component is devoted to the deployment of a blockchain network. The second component provides all necessary cryptographic algorithms, which will be applied to actual blockchain data in the third component. The fourth component provides a mechanism to display the results.

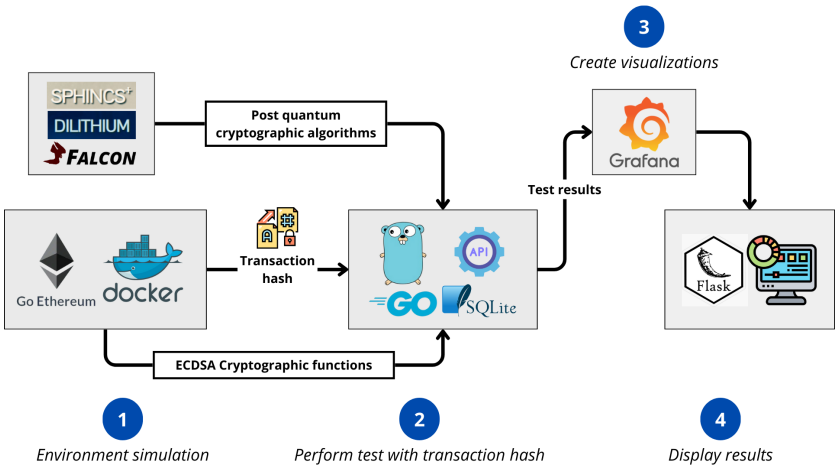

Fig. 1: Benchmarking solution architecture

As shown in the figure, our solution incorporates an actual blockchain network capable of deploying and running a number of Go-Ethereum nodes using a

Docker environment. All transactions sent in the blockchain environment are intercepted and their actual payloads are passed to our benchmarking API. The benchmarking component evaluates the performance of both the cryptographic algorithms currently provided by the Go-Ethereum client and the post-quantum cryptographic schemes selected by NIST. The results are stored in a Sqlite database that is read by Grafana, the component in charge of generating graphics for comparing the evaluation results. These graphics are finally included in a web application developed with Flask to facilitate the visualization and analysis of the results.

4.2 Cryptographic component

This section presents the cryptographic algorithms and libraries incorporated in our benchmarking solution. Currently, it includes the three post-quantum signature algorithms selected by NIST after their standardization competition:

- **CRYSTALS-Dilithium** [7]: Dilithium is a digital signature algorithm based on the hardness of lattice problems over module lattices. It ensures that even with access to a signing oracle, an adversary cannot fabricate a new signature for an unsigned message or a different signature for an already signed message. Dilithium utilizes rejection sampling of Fiat-Shamir with Aborts and the uniform distribution for signature generation, leading to secure yet larger signatures. Despite this, its key size is reduced post-creation with an optimizaiton process, making it the algorithm with the smallest combined public key and signature size among lattice-based schemes. Dilithium offers three modes (Dilithium 2, 3, and 5), with increasing security and resource needs, and includes AES encryption.
  As there is a library that implements this algorithm in GO [6], the necessary test functions have been included directly in the simulation program that has been carried out in GO.
- **Falcon** [10]: As Dilithium, it is a lattice-based digital signature scheme, but uses the Gentry, Peikert, and Vaikuntanathan framework [12], deploying NTRU lattices with a fast Fourier sampling trapdoor sampler. This scheme addresses the short integer solution (SIS) problem over NTRU lattices, which remains computationally challenging even for quantum computers. Falcon offers two modes, Falcon-512 and Falcon-1024, with this benchmark focusing on Falcon-1024 due to its higher security.
  To implement of Falcon, since there is currently no native library for GO, it was necessary to use the CGO library to combine C and GO code, calling Falcon cryptographic functions from C [19] for analysis in GO.
- **SPHINCS$^+$** [20]: Stateless hash-based digital signature scheme using Merkle tree structures. It offers high security with relatively short signatures and fast verification times. SPHINCS$^+$ includes three variants based on the underlying hash functions: SHAKE256, SHA-256, and Haraka, with this benchmark focusing on SHA-256.

The GO library for SPHINCS$^+$ [8] supports 12 modes per hashing method, varying in hash length (128, 192, 256) and mode type (simple, robust). In this case, six modes have been analyzed, covering both simple and robust options for each hashing type.

Although this component only incorporates libraries for the three finalists of the NIST competition, it is designed to facilitate the inclusion of new cryptographic algorithms in the future.

4.3 Measurement component

The measurement component is one of the core elements of the proposed solution. This component is responsible for receiving, in real time, the transactions of all blocks generated by the Ethereum-based blockchain.

Every time a signature is generated or verified in the blockchain network, an API call is made with the hash of the transaction. This hash is then used as payload to perform the comparisons of cryptographic functions.

The tests, which are performed in every single API call, simulate a key generation, signature generation and signature verification process. These tests are first carried out with the cryptographic code extracted from Go-Ethereum, i.e. with ECDSA, and then compared with the post-quantum cryptographic schemes explained above: Dilithium, Falcon and SPHINCS$^+$.

The metrics measured have been the execution time and the memory usage of the corresponding cryptographic function. All measurements have been carried out directly from GO, with native libraries: `time` for time measurement and `runtime/pprof` for CPU profiling. Both are started just before calling the function to which the parameters are going to be measured.

However, instead of measuring the time directly in the blockchain node, we produce also an ECDSA signature in the backend to ensure that all benchmarked algorithms are run in the same environment. As the backend does not know the private keys used for signatures, we need to generate a new pair in each test iteration. In addition to that, key and signature sizes have also been measured apart from the tests.

Execution times The simulations have been carried out several times in a loop and the average execution time of all of them has been calculated, since there have been functions that took extremely low time that were considered as 0ms by the measurement function. This loop was 50 times for the tests performed by API call.

However, the need to perform iterations to obtain more accurate results, increased considerably the time required to complete each test. This has caused another problem, which was that API calls were being received with new block information, while tests were still running with hashes from previous blocks. In short, it took longer for the GO program test to complete than for the private network to validate a new block.

Therefore, it has been essential to implement asynchronous calls in the blockchain network, so that it has not need to wait for the API response and continues its normal operation. At the same time, and more importantly, a queuing system has had to be implemented in the GO program. This has been achieved using GO channels and GO routines. When the application launches, the channel is initialized and a GO routine is started to handle incoming requests. Then for each API call received, the hash information has been added to the channel queue, to be executed one at a time on a first-come, first-served basis.

Memory usage In addition to measuring the execution time of cryptographic functions, in order to assess whether an algorithm is suitable or not, it has been very interesting to quantify the memory usage during executions. For this purpose, four metrics have been evaluated:

- **Alloc**: Amount of memory allocated by the Go runtime for live objects, including all reachable objects in the heap, as well as some additional memory used by the garbage collector and other runtime structures.
- **Total alloc**: Cumulative amount of memory allocated since the program started. It covers all allocations, even those that have been freed by the garbage collector.
- **Sys**: Total memory obtained from the operating system, including both the Go heap and any memory allocated by the Go runtime for other purposes (such as stack space, memory-mapped files, and so on).
- **Num GC**: Number of garbage collection cycles that have occurred since the program started. Garbage collection is the process of reclaiming memory that is no longer in use by the program, and each cycle involves scanning the heap to identify and free unreachable objects.

5 Experimental results

The test results have been divided into three parts: the key and signature sizes, and the execution times and memory usage of the functions under test.

5.1 Key and signature sizes

Table 1 shows the comparison of private and public key and signature sizes for each type of cryptographic algorithm. It is worth mentioning that the size of the public key is only the size of the key itself, which in reality would have to be added 1 bit for the signature of the y-coordinate.

Comparing the key sizes of the various cryptographic schemes, there have been notable differences that reflect their security objectives and underlying algorithms. The best values have been obtained by ECDSA and 128-bit SPHINCS$^+$, which with the sum of the public and private key sizes achieve the same size. ECDSA has a 64-byte public key and a 32-byte private key, while SPHINCS$^+$ has a 32-byte public key and a 64-byte private key.

Table 1: Key and signature sizes of different algorithms

Algorithm	**Public key**	**Private key**	**Signature**
ECDSA	64	32	64
Dilithium2	1312	2528	2420
Dilithium3	1952	4000	3293
Dilithium5	2592	4864	4595
Falcon 1024	1793	2305	1231
SPHINCS$^+$ SHA256 128bit - Robust	32	64	17088
SPHINCS$^+$ SHA256 192bit - Robust	48	96	35664
SPHINCS$^+$ SHA256 256bit - Robust	64	128	49856

In any case, although ECDSA could be considered the best in this aspect, it should be noted that the set of all SPHINCS$^+$ schemes have values very close to ECDSA and therefore, values that could be competitive to it in this aspect. Among SPHINCS$^+$ versions, the higher the security level, the more bits are used and therefore, the key sizes increase. However, using the simplest version, the 128 bit one, it is possible to use the same key sizes as ECDSA.

In contrast, both Dilithium and Falcon have been excessively far from the ECDSA or SPHINCS$^+$ key size values. They should be used in scenarios where security is paramount and large key size is not an issue.

In case of the signatures, it can be seen that the result changes radically compared to the key sizes comparison. Although ECDSA has still been the best, SPHINCS$^+$, which for the keys was the second best option, becomes the worst with a huge difference with the rest.

In this case, the second best option has been Falcon 1024. However, the size have fallen far above of the ECDSA signature size, since the size of a Falcon 1024 signature is equivalent to just over 19 ECDSA signatures.

The size of the signatures generated using Dilithium has also turned out to be considerably larger, doubling the size of Falcon 1024 using Dilithium2, and more than tripling if Dilithium5 is employed.

In general, it can be concluded that there is no scheme that globally (adding the three sizes) comes close to ECDSA, since the second best is Falcon and exceeds it by 33 times. Especially analyzing the sizes of the signatures, it can be clearly stated that all post-quantum schemes are extremely far from the ECDSA values. However, SPHINCS$^+$, in the aspect of the keys obtains a very good result, being even better in the size of the public keys and not much worse in the private ones. In any case, it is clear that using these algorithms, the increase in the size of the keys and especially of the signatures is an inevitable consequence of the improvement in security they offer.

5.2 Execution times

One of the most important factors when choosing one cryptographic algorithm over another is the time required for the algorithm to execute the cryptographic

functions. Therefore, much emphasis has been placed on measuring and analyzing in a precise and detailed way the execution times required by each cryptographic scheme.

The final average results after more than 24 hours of running the blockchain network have been those summarized in the Table 2 and also visible in the historical data dashboards of the project.

Table 2: Average execution times by algorithms (ms)

Algorithm	**Key generation**	**Signature generation**	**Signature verification**
ECDSA	90.7	58.6	71.4
Dilithium2	80.5	149	17.3
Dilithium2-AES	102	124	16.5
Dilithium3	144	222	22.7
Dilithium3-AES	171	191	19.5
Dilithium5	201	251	33.6
Dilithium5-AES	254	207	29.6
Falcon 1024	90434	13162	166
SPHINCS$^+$ SHA256 128bit - Robust	5612	120583	7845
SPHINCS$^+$ SHA256 128bit - Simple	3376	72760	4569
SPHINCS$^+$ SHA256 192bit - Robust	8318	198359	11856
SPHINCS$^+$ SHA256 192bit - Simple	4977	118773	6777
SPHINCS$^+$ SHA256 256bit - Robust	26333	483158	14568
SPHINCS$^+$ SHA256 256bit - Simple	13074	240571	6957

Firstly, evaluating the differences in key generation times, it can be concluded that there have been two algorithms that clearly stand out above the rest: ECDSA and Dilithium2 (both in its normal version and in the AES version, which hardly varies). However, it should be noted that all versions of Dilithium have a relatively good execution time that did not deviate that much from the ECDSA times. In fact, the Dilithium2 version improves on the ECDSA result and achieves the shortest execution time of all the algorithms. However, both SPHINCS$^+$, in all its versions, and Falcon, differ exaggeratedly compared to the times of the rest, especially the latter.

Secondly, evaluating the signature generation times, there is no doubt that ECDSA is clearly above the rest of the algorithms, as it is slightly more than twice as fast as the second, Dilithium2-AES. Unlike the key generation, in this section Falcon 1024 (224 times the ECDSA time) has obtained a better result than SPHINCS$^+$ (2057 times the ECDSA time in the simplest mode), but they are still times that are not at all competitive compared to the times of ECDSA or even Dilithium.

However, at this point it must be taken into account the large difference that existed between the signature size of ECDSA and the signature sizes of the post-quantum algorithms, which greatly influences the time of signature generation.

Finally, as far as signature verification times are concerned, it is interesting to note that ECDSA did not obtain the best score on this point, being clearly outperformed by Dilithium in all its versions.

Dilithium2 has been more than four times faster than ECDSA, and Dilithium5, the version with the highest level of security, 2.1 times faster too. Falcon 1024, on the other hand, although it has been more than twice as slow as ECDSA, was not far behind. SPHINCS$^+$, in all its modes, has been the scheme that took an exorbitant amount of time compared to the rest, being 47.3 times slower than Falcon 1024 for example (128bits robust mode). However, although it is not a very considerable difference, a clear difference in times could be observed when using the robust or simple mode of SPHINCS$^+$, regardless of the number of bits used.

Summarizing the results, on the one hand, in key generation, Dilithium obtains a result very close to ECDSA, even improving its time in its Dilithium2 version. The rest of the algorithms need an extremely higher time and are not competitive at all. The same happens in the generation of signatures, although in this case Dilithium does not improve the time of ECDSA in any version, it is very close. The worst here is SPHINCS$^+$, while Falcon improves compared to key generation. The most remarkable thing happens in signature verification, where Dilithium is exceptionally better than the rest, including ECDSA, being between 2.1 and 4.1 times faster than ECDSA depending on the versions used.

Therefore, in the sum of all times, it can be concluded that Dilithium needs a similar amount of time as ECDSA, being practically the same time in the Dilithium2 version. Comparing it with the rest of the post-quantum schemes, it obtains an excellent result, being the only one that could compete in this aspect with the current cryptographic methods.

5.3 Memory usage

In addition to measuring the execution time of cryptographic functions, in order to assess whether an algorithm is suitable or not, it has been very interesting to quantify the memory usage during executions. Nevertheless, obtained results have not been as interesting and analyzable as those of the execution time, which has led to more extensive conclusions.

The values of total alloc, sys and num GC, have concluded to be very similar between all the executions of the different algorithms in each test. The metric of interest for the analysis has been the alloc, the amount of memory allocated by the Go runtime for live objects, which has led to detect greater variation among the different algorithms.

First, in the memory allocation levels of the key generation, a clear difference has been observed between ECDSA and Falcon compared to Dilithium and SPHINCS$^+$. Although ECDSA achieved the lowest result, it is less than 1% better than Falcon, which is a negligible difference. The difference to consider exists when compared to the other two post-quantum algorithms, which need about 55-60% more memory than the first two.

Next, the results of the same metric have been analyzed for the signature generation process, and it was observed that in this aspect, Dilithium improves and approaches the values of ECDSA and Falcon, which are still slightly better in that order. SPHINCS$^+$, on the other hand, still requires about 50% more memory compared to the other three algorithms.

Finally, signature verification follows the same pattern as signature generation, with all the algorithms being quite close to each other, except SPHINCS, which continues with much higher values. However, in this case, it can be seen that there is a notable difference between the different modes that has SPHINCS. The robust 192-bit mode gives the best result, while curiously, the simple mode of the same bits is the one that gives the worst result of all, surpassing it by just over 10%. The rest of the algorithms are in between, but except for the 128-bit one, in the other two, it is the simple mode which needs less allocated memory than the robust mode.

6 Conclusions and future work

One of the main conclusions was obtained during the research and study process. At present, quantum computers are still a long way from being able to break current encryption schemes easily and quickly. Current quantum computers are not necessarily large enough to break the schemes fast, nor are they anywhere near the size needed at the moment.

The test execution times have been where the most interesting information has been extracted, especially because a similar performance has been seen between ECDSA and Dilithium, and a huge difference has been observed with the other two analyzed schemes.

Analyzing the memory usage, it can be said that even though there is a difference between the schemes, it is not as big as in the case of the execution times. Falcon equals ECDSA in terms of results, followed by Dilithium, which needs more resources in key generation, but in the other two processes it is practically the same as ECDSA. However, SPHINCS$^+$ does require slightly more memory in all processes.

In summary, analyzing all the results, it can be clearly concluded that in terms of performance, Dilithium is currently the best post-quantum cryptographic scheme, being quite close to the performance levels of ECDSA, which has been the present scheme used as a comparison. In the case of the simplest version of Dilithium (Dilithium2), summing all the values of time and memory, it would only need 12% more execution time in the three processes evaluated and 20% more memory. These values, which although on a large scale could make a considerable difference, are values that are not too far off current levels. Therefore, the increase in security level that the integration of this algorithm could entail, could be totally understandable in exchange for the slight decrease in performance.

From the results presented and also during the development, new ideas, questions and improvements have arisen, which will be detailed below.

The main point for improvement would be to integrate the post-quantum cryptographic methods used in the tests into the blockchain network directly, i.e. to include them into the source code of the used Ethereum client, in this case Go-Ethereum. This way, more detailed tests could be performed, with real transactions, enabling the analysis of more factors such as network load for example. It would be possible to analyze the behavior of the methods in a real network, including latency, node distribution, volume of transactions...

Secondly, while current tests only include post-quantum digital signature algorithms selected by NIST, many other post-quantum algorithms could be tested in the future.

Finally, current tests were performed on a local computer with its own computational limitations and concurrent processes, which might affect results. Moving the system to an external server dedicated to testing, free from other tasks, would provide more accurate performance measurements. Additionally, creating a network of multiple systems to run the same tests and share a common database would allow for calculating average values across different systems, offering a broader perspective on performance results.

Acknowledgements

This work has been partially supported by project PID2022-139268OB-I00, financed by MCIN/AEI /10.13039/501100011033 / FEDER, UE and project TED2021-129830B-I00, financed by MCIN/AEI /10.13039/501100011033/Next-GenerationEU/PRTR.

References

1. Arute, F., Arya, K., Babbush, R., et al.: Quantum supremacy using a programmable superconducting processor. Nature **574**(7779), 505–510 (10 2019). https://doi.org/10.1038/s41586-019-1666-5
2. Bhatia, V., Ramkumar, K.: An efficient quantum computing technique for cracking rsa using shor's algorithm. In: 2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA). pp. 89–94 (2020). https://doi.org/10.1109/ICCCA49541.2020.9250806
3. Brooks, M.: Quantum computing is taking on its biggest challenge: noise (01 2024), https://www.technologyreview.com/2024/01/04/1084783/quantum-computing-noise-google-ibm-microsoft/
4. Buser, M., Dowsley, R., Esgin, M., Gritti, C., Kermanshahi, S.K., Kuchta, V., Legrow, J., Liu, J., Phan, R., Sakzad, A., Steinfeld, R., Yu, J.: A survey on exotic signatures for post-quantum blockchain: Challenges and research directions. ACM Computing Surveys **55**(12) (3 2023). https://doi.org/10.1145/3572771
5. Chen, J., Gan, W., Hu, M., Chen, C.M.: On the construction of a post-quantum blockchain for smart city. Journal of Information Security and Applications **58**, 102780 (2021). https://doi.org/10.1016/j.jisa.2021.102780
6. Cloudflare: Circl: Cloudflare interoperable reusable cryptographic library (04 2024), https://pkg.go.dev/github.com/cloudflare/circl/sign/dilithium
7. CRYSTALS Team: Cryptographic suite for algebraic lattices (crystals) (08 2023), https://pq-crystals.org/

8. Daugaard, K.: Sphincsplus-golang (12 2023), https://github.com/kasperdi/SPHINCSPLUS-golang
9. Fernández-Caramès, T.M., Fraga-Lamas, P.: Towards post-quantum blockchain: A review on blockchain cryptography resistant to quantum computing attacks. IEEE Access **8**, 21091–21116 (2020). https://doi.org/10.1109/ACCESS.2020.2968985
10. Fouque, P., Hoffstein, J., Kirchner, P., Lyubashevsky, V., et al.: Falcon (11 2017), https://falcon-sign.info/
11. Gao, Y.L., Chen, X.B., Chen, Y.L., Sun, Y., Niu, X.X., Yang, Y.X.: A secure cryptocurrency scheme based on post-quantum blockchain. IEEE Access **6**, 27205–27213 (2018). https://doi.org/10.1109/ACCESS.2018.2827203
12. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions (2007), https://eprint.iacr.org/2007/432
13. Gidney, C., Ekerå, M.: How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. Quantum **5**, 433 (4 2021). https://doi.org/10.22331/q-2021-04-15-433
14. IBM: Ibm's roadmap for scaling quantum technology. IBM (12 2023), https://www.ibm.com/quantum/blog/ibm-quantum-roadmap
15. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). International Journal of Information Security **1**(1), 36–63 (08 2001). https://doi.org/10.1007/s102070100002
16. Li, C.Y., Chen, X.B., Chen, Y.L., Hou, Y.Y., Li, J.: A new lattice-based signature scheme in post-quantum blockchain network. IEEE Access **7**, 2026–2033 (2019). https://doi.org/10.1109/ACCESS.2018.2886554
17. National Institute of Standards and Technology (NIST): Post-quantum cryptography (04 2024), https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
18. Pati, C.: Search using grover's algorithm. National Institute of Technology Rourkela (11 2023). https://doi.org/10.13140/RG.2.2.25842.07369
19. Pornin, T.: Falcon source files (reference implementation) (11 2021), https://falcon-sign.info/impl/falcon.h.html
20. SPHINCS+ Team: Sphincs+ (08 2023), https://sphincs.org/
21. Yang, Z., Alfauri, H., Farkiani, B., Jain, R., Di Pietro, R., Erbad, A.: A survey and comparison of post-quantum and quantum blockchains. IEEE Communications Surveys & Tutorials **26**(2), 967–1002 (2024). https://doi.org/10.1109/COMST.2023.3325761
22. Yi, H.: Secure social internet of things based on post-quantum blockchain. IEEE Transactions on Network Science and Engineering **9**(3), 950–957 (2022). https://doi.org/10.1109/TNSE.2021.3095192