Enabling a Smooth Migration towards Post-Quantum Security for Ethereum

Xinxin Fan[1][0000−0002−4592−6603], Teik Guan Tan[2][0000−0003−3373−699X], Nicholas Ho[2], and Shi Hong Choy[2]

[1] IoTeX, Menlo Park, CA 94025, USA
xinxin@iotex.io
[2] pQCee Pte Ltd, Singapore 038987
teikguan,nicholasho,shihong@pqcee.com

Abstract. Digital signatures based on Elliptic Curve Digital Signing Algorithm (ECDSA) are widely used in Ethereum to secure transactions and Proof-of-Stake (PoS) consensus protocols. However, those digital signatures are vulnerable to quantum computing and therefore endanger the security of Ethereum and its millions of users' crypto assets. Based on the previous work in [16, 17], we present two proposals for a smooth migration of Ethereum towards post-quantum security in this paper. While the first proposal introduces a new Ethereum transaction type to encapsulate a quantum-safe zero-knowledge proof, the second one further improves system scalability via proof aggregation and zero-knowledge rollups. Our proposals only introduce minimal changes to the software running on Ethereum validators and clients, thereby achieving great backward compatibility. We report our initial evaluation results of the two proposals on Microsoft's Azure cloud platform and highlight the key observations, in the area of improving proof generation timing and proof sizes, for deploying our solutions in practice.

Keywords: Ethereum · ECDSA · Post-Quantum Security · Migration

1 Introduction

With the possibility of large-scale quantum computing on the horizon, blockchain systems (e.g., Ethereum) that are currently secured by elliptic curve cryptography could become vulnerable to a quantum attacker. As shown in Fig. 1, ECDSA has been extensively used in Ethereum for users signing transactions, validators verifying transactions in a PoS consensus process, among other use cases. Once a user has made a transaction with his/her Ethereum wallet, a quantum attacker is able to recover the user's private key using Shor's algorithm [15] and steal all the assets from the user's wallet. As a result, migrating Ethereum towards post-quantum security is a critical step for protecting users' crypto assets in the near future.

The National Institute of Science and Technology (NIST) has been working on the Post-Quantum Cryptography project [12] during the past seven years

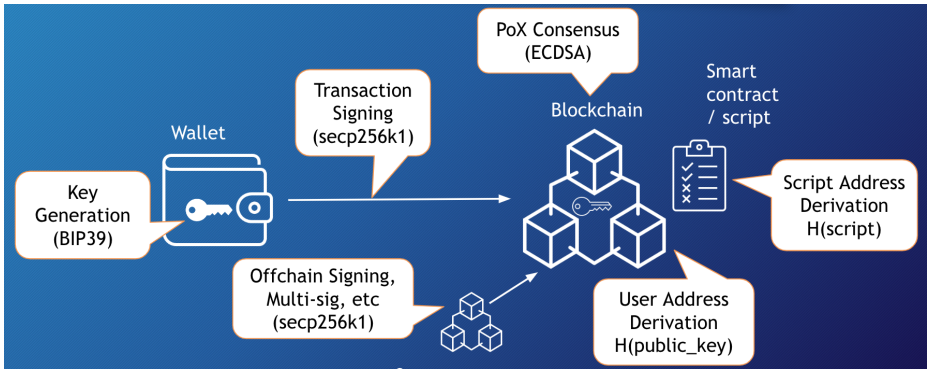

Fig. 1. The Usage of ECDSA in Ethereum

for soliciting, evaluating, and standardizing quantum-resistant public-key cryptographic algorithms. In the Ethereum research community, there was a proposal [11] regarding adoption of NIST standardized quantum-safe signature algorithm Falcon in Ethereum. Recent post [2] by Vitalik Buterin has further highlighted importance of migrating Ethereum towards post-quantum security. In [16], Tan and Zhou proposed a novel generic construction of quantum-safe signature algorithms by augmenting a classical digital signature with a quantum-resistant zero-knowledge proof of knowledge of the pre-image of the private signing key using ZKBoo [7]. For verifying such a quantum-safe signature, a verifier needs to verify validity of both the classical signature and corresponding zero-knowledge proof. In [17], Tan and Zhou further analyzed the impact for migrating blockchain away from ECDSA for post-quantum security.

In this paper, we present two proposals to enable a smooth migration towards post-quantum security for Ethereum. Our proposals are built upon the quantum-safe signature algorithm proposed in [16]. The first proposal covers how Ethereum at Layer-1 can be updated to be post-quantum secure by natively verifying every additional quantum-resistant zero-knowledge proof submitted by the users. The second proposal further improves on scalability of the first one by leveraging zk-Rollups coupled with a tree-based recursive proof aggregation process. In particular, our proposals align well with the current implementations of Ethereum validators and clients and therefore achieve great backward compatibility. The initial performance evaluation results demonstrates a path for deploying our solution in practice.

The rest of the paper is organized as follows: In Section 2, we present some preliminaries, followed by the detailed description and implementation considerations of our two migration proposals in Section 3. Section 4 evaluates the performance of our migration proposals and highlights the key observations with respect to the real-world deployment. Finally, we conclude the paper in Section 5.

2 Preliminaries

In this section, we cover the preliminaries on quantum-safe digital signature algorithms and zero-knowledge proofs, Ethereum transactions, and zk-rollups.

2.1 Quantum-Safe Digital Signature Algorithms

In [16], Tan and Zhou described a novel approach for layering in quantum-resistance into classical digital signature algorithms like ECDSA by applying quantum-safe zero-knowledge proofs on the pre-image of the private signing key. The proposed generic quantum-resistant digital signature scheme consists of the following three algorithms:

- $\mathbf{KeyGen}_q(1^n) \rightarrow \{\rho, K_p\}$ takes in a security parameter 1^n which defines the cryptographic key strength of n, and outputs a secret pre-image ρ and a public key K_p. K_p is the associated public key to the private key $H(\rho)$ where $H(\cdot)$ is a collapsing hash function [19][3]
- $\mathbf{Sign}_q(M, \rho) \rightarrow \{\sigma, \phi\}$ takes in a message M and the secret pre-image ρ, and outputs a signature σ computed using the classical signature algorithm $\mathbf{Sign}(M, H(\rho))$ as well as a quantum-resistant zero-knowledge proof ϕ that i) $H(\rho)$ is computed from ρ and ii) σ is computed from $H(\rho)$.
- $\mathbf{Verify}_q(M, K_p, \sigma, \phi) \rightarrow \{\text{result}\}$ takes in a message M, the public key K_p and signature σ, and outputs accept if and only if the classical signature verification algorithm $\mathbf{Verify}(M, K_p)$ returns accept and ϕ is a valid zero-knowledge proof that σ is computed from ρ.

The above construction is able to achieve quantum resistance while maintaining backward compatibility with existing classical digital signature implementations.

2.2 Ethereum Transactions

An Ethereum transaction refers to an action initiated by an externally-owned account (EOA). As of Ethereum's London upgrade, there are three transaction types: 0x0 (EIP-2718 [21]), 0x1 (EIP-2930 [5]), and 0x2 (EIP-1559 [3]). Transactions with type 0x0 are legacy transactions that contain the following parameters:

- **nonce**: A sequentially incrementing counter which indicates the transaction number from the account;
- **gasPrice**: The number of Wei to be paid per unit of gas for conducting a transaction or executing a contract;
- **gasLimit**: The maximum amount of gas units that can be consumed by the transaction;
- **to**: The 160-bit receiving address;
- **value**: The number of Wei to transfer from sender to recipient;

[3] Collapsing hash functions are defined to be collision-resistant in the face of quantum attacks.

- **data**: The optional field to include arbitrary data;
- **v, r, s**: The signature of the transaction used for determining the identifier of the sender.

Transactions with type 0x1 were introduced in EIP-2930 and contain, along with the legacy parameters, an **accessList** parameter. Each access list is a tuple of an account address and a list of storage keys that the transaction plans to access. Finally, transactions with type 0x2 were introduced in EIP-1559 for addressing the network congestion and overpricing of transaction fees caused by the historical fee market. To this end, EIP-1559 replaces **gasPrice** in legacy transactions with an in-protocol, dynamically changing base fee per gas at each block.

2.3 Quantum-Safe Zero-knowledge Proofs

In cryptography, zero-knowledge proof (ZKP) [8] is a method by which one party (i.e., a prover) can prove to another party (i.e., a verifier) that a given statement is true, without disclosing additional information beyond the fact that the statement is true. ZKPs need to satisfy the formal requirements of *completeness*, *soundness*, and *zero-knowledge*, thereby enabling one to build trustless applications. Earlier design of ZKPs are not quantum-resistant and those that only use collision-resistant hash functions are plausibly post-quantum secure. **Z**ero-**K**nowledge **S**calable **T**ransparent **AR**gument of **K**nowledge (zk-STARK) [1] and MPC-in-the-Head (MPCitH) [10] are well-known quantum-safe ZKP examples widely used in practice.

2.4 zk-Rollups

A zero-knowledge rollup (a.k.a. zk-Rollup) [18] is a Layer-2 scaling technique for Ethereum. zk-Rollups bundle transactions into batches that are executed off-chain and verified on-chain using non-interactive ZKPs, thereby greatly increasing transaction throughput and reducing transaction costs. In practice, zk-Rollups inherit the security of a Layer-1 blockchain and rely on it for data availability and settlement. zk-Rollups are typically realized using two smart contracts deployed on a Layer-1 blockchain, namely a *main contract* and a *verifier contract*. While the main contract stores rollup blocks, track transactions, and monitor state updates, the verifier contract verifies ZKPs submitted by the Layer-2 rollup nodes. When combining zk-Rollups with post-quantum signatures, an off-chain node can verify multiple signatures and generate a (succinct) zero-knowledge proof that could be verified by a smart contract. Such a combination is able to improve the system scalability and user experience significantly.

3 Migration Approaches

To protect Ethereum transactions from potential risks of the rapid growth of quantum computing, we describe two incremental proposals for migrating Ethereum towards post-quantum security in this section. Without loss of generality, legacy transactions will be used as examples throughout this section.

3.1 Layer-1 Hard Fork

Our first proposal is to introduce new transaction types that augments each existing transaction type with a new parameter **proofUri**. This parameter contains the URI of a quantum-safe zero-knowledge proof (e.g., zkSTARK or MPCitH). With the introduction of **proofUri**, an augmented legacy transaction will be processed as illustrated in Fig. 2.

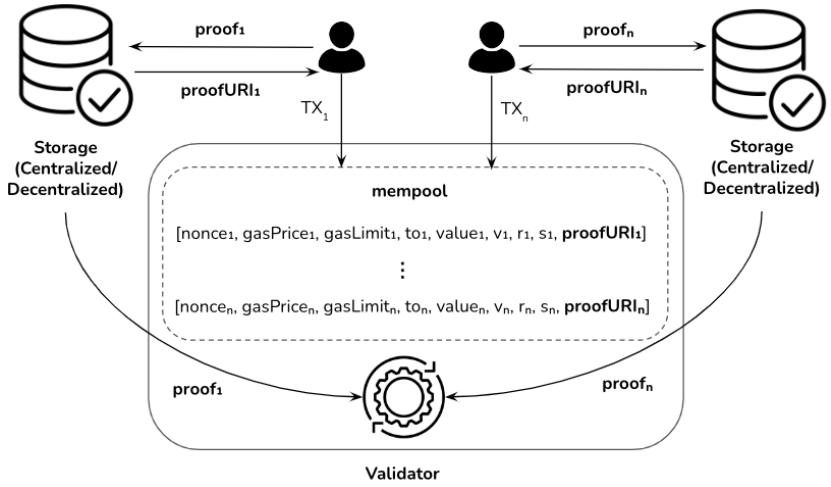

Fig. 2. The Process of Augmented Legacy Transactions

- A user generates a post-quantum secure zero-knowledge proof for the statement that he/she knows a secret (e.g., a mnemonic phrase) which can be used to generate the user's Ethereum wallet address by following a specified address derivation process (e.g. BIP-39 [14]).
- The user stores the generated **proof** to a selected storage provider and obtains a publicly accessible URI **proofUri**.
- The user creates a transaction by augmenting a legacy transaction with **proofUri** and sends it to the mempool of a validator.
- Upon receiving a new transaction in the mempool, a validator first verifies the ECDSA signature in the transaction, followed by retrieving the **proof** from the storage provider with **proofUri** and verifying its validity. A transaction is considered as valid if both signature and proof verification succeed, besides other sanity checks.

The validation of a newly proposed block works as before, except that a validator needs to verify the **proof** retrieved from storage provider with the **proofUri** for each transaction in the block. Note that each transaction is publicly

verifiable provided that the corresponding **proof** is publicly available and a user might choose to delete the **proof** after the transaction passes the Ethereum's PoS consensus process. The introduction of **proofUri** does not incur significant storage overhead for existing Ethereum validators and the length of **proofUri** can be further restricted (e.g., less than 256 characters). The computational cost of verifying a **proof** depends on the usage of a specific post-quantum secure zero-knowledge proof scheme.

3.2 Layer-2 zk-Rollups

To further improve the system scalability, our second proposal is to employ Layer-2 zk-Rollup architecture and process augmented legacy transactions in batch, as illustrated in Fig. 3.

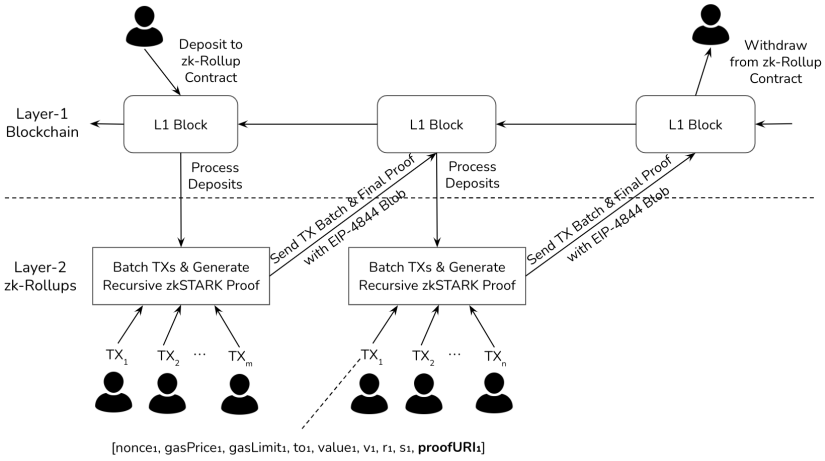

Fig. 3. Scalable Process of Augmented Legacy Transactions with zk-Rollups

The above process follows a typical zk-Rollup workflow with the following modifications:

- Given a transaction batch processed by a rollup node, it needs to first retrieve all the zkSTARK proofs using the corresponding **proofUri**'s and then aggregate those proofs into a final proof in a recursive manner.
- The transaction batch together with the single final proof are submitted to the Layer-1 blockchain via a blob transaction as specified by EIP-4844 [4].

To generate a final proof in a recursive manner, the rollup node can leverage a tree-based recursive proof aggregation process as shown in Fig. 4.

During the aggregation process, all the zkSTARK proofs (i.e., $\mathbf{proof}_1, \ldots, \mathbf{proof}_m$) generated by users become the leaf nodes of an aggregation tree. For

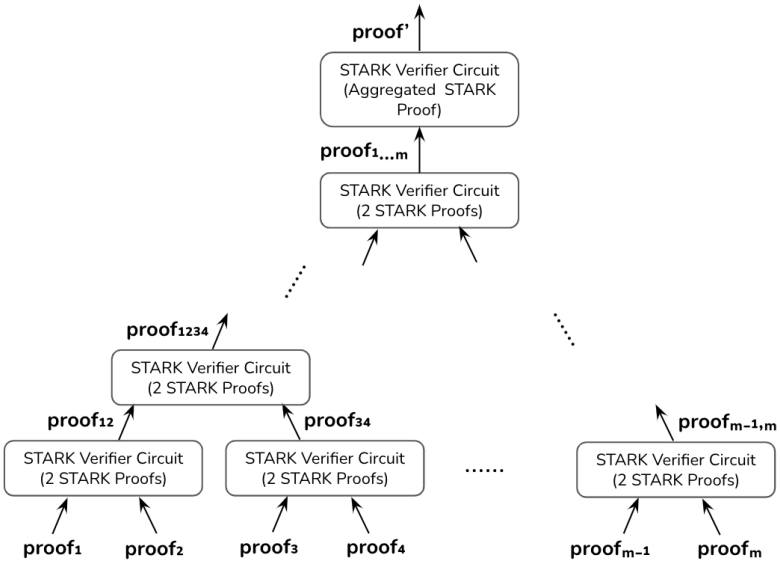

Fig. 4. A Tree-Based Recursive Aggregation of zkSTARK Proofs

each intermediate node, a STARK verification circuit is instantiated to aggregate two proofs generated by its child nodes, respectively. Once the aggregated proof (i.e., $\mathbf{proof}_{1,\ldots,m}$) is generated, another STARK verification circuit is employed to compress the aggregated proof into a final proof that can be verified on-chain[4].

3.3 Implementation Considerations

In this subsection, we discuss implementation considerations when deploying the proposed solution in practice.

User Proof Generation A majority of Ethereum wallets implement the advanced form of deterministic wallet (i.e., Mnemonic code based hierarchical deterministic (HD) wallet) as specified in BIP-32 [20], BIP-39 [14] and BIP-44 [13]. The key derivation in HD wallet follows a tree-like structure in which a parent key can derive a sequence of child keys and each child key can derive a sequence of grandchild keys, as illustrated in Fig. 5.

Hundreds of hash function operations need to be performed for deriving an Ethereum address from a 12/18/24-word's mnemonic code. As a result, proving the statement that a user knows the mnemonic code corresponding to an Ethereum address leads to a complex ZK circuit and long proof generation time.

[4] See https://github.com/starkware-libs/starkex-contracts/blob/master/evm-verifier/solidity/contracts/StarkVerifier.sol for an example of a STARK proof verifier written in `Solidity`.

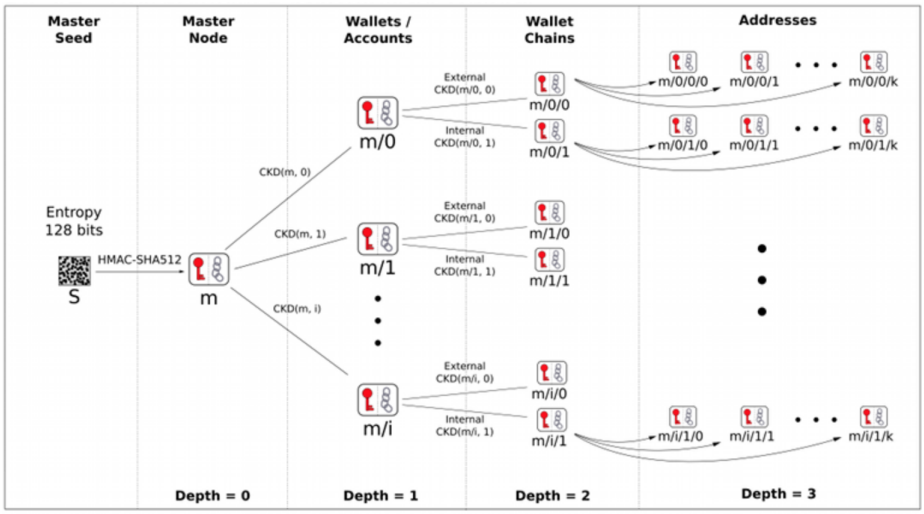

Fig. 5. Key Derivation in HD Wallets [20]

In fact, one can make a flexible trade-off between security and complexity of a proving circuit. For instance, a user may prove that he/she knows the pre-image[5] of the last hash function during the key derivation process in lieu of the knowledge of the mnemonic code itself, which is going to reduce the complexity of a proving circuit significantly.

Mnemonic Code Access Our proposals require access of a user's mnemonic code in order to generate zero-knowledge proofs. One approach is to ask a user to type the mnemonic code each time he/she would like to send a transaction. However, this approach results in poor user experience. Another approach relies on the secure hardware backed Ethereum wallets (e.g., hardware wallets, mobile wallets, etc.). In this case, a user can choose to store the mnemonic code in the secure hardware and enforce a stringent access policy for accessing it.

Data Availability The new transaction type introduces the parameter **proofUri** that is used to locate a user generated **proof** off-chain. Both **proofUri**s and **proof**s should be held by validators until the finality is reached on Ethereum. After that **proofUri** and **proof** can be safely removed by the validators and users, respectively. This approach does not incur additional overhead with respect to the on-chain storage. Furthermore, it allows users to choose any off-chain (centralized/decentralized) storage provider for storing generated **proof**s.

[5] The user can use the mnemonic code and follow the key derivation process as specified in BIP-32 to obtain the pre-image of the last hash function.

Backward-Compatibility and Fallback Our proposals can achieve backward-compatibility by only introducing minimal changes to the existing implementations of Ethereum validators and clients. Moreover, if an Ethereum wallet does not upgrade to support the new transaction type, it can still submit transactions that only contain ECDSA signatures. However, it is up to an Ethereum validator to determine whether those transactions are still being supported. Such a fallback feature allows Ethereum validators and clients to complete software upgrades in an asynchronous manner.

4 Performance Evaluation

In this section, the preliminary performance evaluation results are presented, followed by a discussion of potential improvements.

4.1 Evaluation Setup

We proceeded to implement the proof generation algorithms as described in Section 3.1 based on the following setup:

- **Platform**. We use a standard F32s v2 (Azure) x64 architecture with 32 vCPUs and 64GB RAM running Ubuntu 22.04 LTS.
- **zkVMs**. The Zero-knowledge proof circuits will be generated using zero-knowledge Virtual machines (zkVMs). We use both SP1 [6] version 1.0.8-testnet, as well as RISC0 [7] version 1.0.1 for proof circuit generation and execution.
- **Proof Algorithm**. The algorithm chosen is described in Equation 1.

$$\begin{aligned} Sign_q(M, \rho) &= secp256k1(M, Hash(\rho)) \\ Hash(\rho) &= SHA256(\rho) \\ M &= 256 \text{ bit value} \\ \rho &= 256 \text{ bit value} \end{aligned} \tag{1}$$

4.2 Evaluation Benchmarks

Table 1 shows the results of execution of proof generation for a target security of 100 bits.

The proof size generated per signature is at least 5 MBytes of data which is not gas-efficient as expected. Proof generation also takes several minutes on a CPU which is not feasible for most end-user transactional activities. Allocating more memory to the virtual machines does not improve the performance.

We next explored the possibility of reducing the size of the proof based on:

1. *Adjusting the security bits.* This adjustment is supported on the SP1 zkVM and the results are shown in Table 2.

[6] available at https://github.com/succinctlabs/sp1
[7] available at https://github.com/risc0/risc0

Table 1. Comparing SP1 and RISC0 proof generation

zkVM	SP1	RISC0
Proof Size (Bytes)	5,333,944	28,675,808
Memory Consumed (MBytes)	42,350	8,470
Program CPU Cycles	10,552,692	13,631,488
Execution Time (seconds)	309	442

Table 2. Comparing Proof Sizes based on Security Bits

Security Bits	60	80	100	128
Proof Size (Bytes)	3,253,624	4,293,784	5,333,944	6,790,168
Memory Consumed (MBytes)	42,350	42,230	42,350	42,150
Execution Time (seconds)	304	304	309	304

The size of proofs generated are linearly reduced based on the security bits, although the memory consumed or execution time remains constant. The reduction of security bits below 100 is not recommended since it will weaken the proof, and allow attackers to potentially use Grover's [9] algorithm, a brute-force search quantum algorithm with quadratic performance speedup, to compromise the system.

2. *Using Layer-2 zk-Rollups as described in Section 3.2.* A feature of the zk-STARK proof system is the ability to batch multiple proofs into a constant-size recursive proof. We tested this setup using both SP1 (compressed) and RISC0 (succinct) options to apply to batch 3 proofs and 10 proofs for comparison purposes. We had to increase the platform's memory from 64 GB to 256 GB to accommodate the execution, and retained the original security bit setting of 100. The results of execution are found in Table 3.

Table 3. Comparing Performance of Recusive proofs.

zkVM	SP1		RISC0	
# of proofs combined	3	10	3	10
Recursive Proof Size (Bytes)	1,771,102	1,771,917	1,799,632	1,831,696
Memory Consumed (MBytes)	55,660	96,685	8,428	8,430
Execution Time (seconds)	442	797	982	1974

Based on our results, we can see that by batching the proofs, we can reduce the unit size per proof in exchange for a higher execution time. When using SP0 to batch 10 proofs, the average proof size per proof is reduced to 177 KBytes and execution time per proof is 80 seconds which starts to become practical.

4.3 Evaluation Discussion

From our preliminary benchmarks, we have obtained some concrete evaluation results which demonstrate a path to practical implementation of our design. The key observations are summarized below.

– The proof generation with SP1 and RISC0 is slow on CPU, thereby limiting their usage in mobile and desktop wallet applications directly. A dedicated ZKP proving service with hardware acceleration (e.g., GPU, FPGA, etc.) should be deployed to improve user experience in practice, as illustrated in Fig. 6. Note that the ZK proving service should run inside a trusted execution environment (TEE) to ensure security of processing users' mnemonic codes. A hardware-based remote attestation process [6] is conducted each time a mobile or desktop wallet needs to connect to the proving service for generating a ZK proof. Although such an implementation increases the attack surface area, it will improve usability and user adoption.

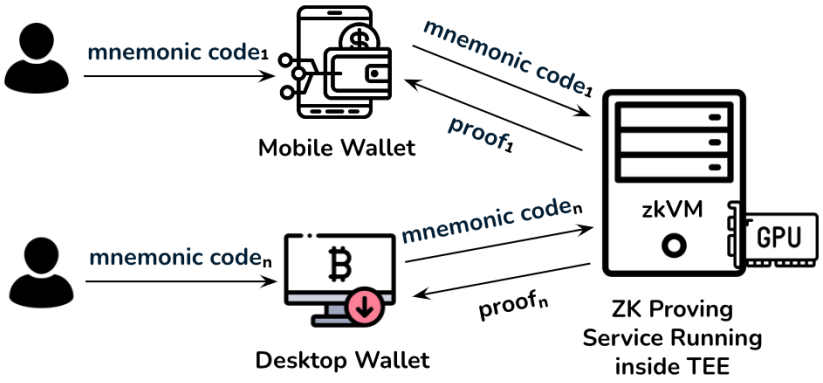

Fig. 6. A ZK Proving Service Running inside a Trusted Execution Environment (TEE)

– The proof size of zkSTARK is large and using a data availability solution like EIP-4844 [4] can effectively reduce the cost of storing transaction batches and the corresponding aggregated proofs on chain.
– The proof aggregation can effectively amortize the proof size, at the cost of higher proof generation time and hardware platform requirements.

5 Conclusion

Although the threat of quantum computers to endanger the security of Ethereum may be some years away, the early articulation of clear requirements and direction of how a quantum-safe Ethereum can be achieved will remove any wild speculation to the fate of Ethereum and its community. We have proposed concrete

approaches that focused on backward-compatibility for a smooth post-quantum migration experience for both users and developers, while not compromising on quantum-safety. The initial performance evaluation demonstrates the technical viability of our solution, and highlights the need for proof generation time and proof size reduction in order to make the solution usable. As our future work, we will focus on continuing improving performance and user experience of our solution by exploring memory-efficient ZK proof systems as well as customized ZK circuits.

References

1. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046 (2018), https://eprint.iacr.org/2018/046, https://eprint.iacr.org/2018/046
2. Buterin, V.: How to hard-fork to save most users' funds in a quantum emergency. Tech. rep. (2024), https://ethresear.ch/t/how-to-hard-fork-to-save-most-users-funds-in-a-quantum-emergency/18901
3. Buterin, V., Conner, E., Dudley, R., Slipper, M., Norden, I., Bakhta, A.: Eip-1559: Fee market change for eth 1.0 chain. Ethereum improvement proposals (2019), https://eips.ethereum.org/EIPS/eip-1559
4. Buterin, V., Feist, D., Loerakker, D., Kadianakis, G., Garnett, M., Taiwo, M., Dietrich, A.: Eip-4844: Shard blob transactions. Ethereum improvement proposals (2022), https://eips.ethereum.org/EIPS/eip-4844
5. Buterin, V., Swende, M.: Eip-2930: Optional access lists. Ethereum improvement proposals (2020), https://eips.ethereum.org/EIPS/eip-2930
6. CCC: A technical analysis of confidential computing v1.3 (2022), https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC-A-Technical-Analysis-of-Confidential-Computing-v1.3_unlocked.pdf
7. Giacomelli, I., Madsen, J., Orlandi, C.: {ZKBoo}: Faster {Zero-Knowledge} for boolean circuits. In: 25th usenix security symposium (usenix security 16). pp. 1069–1083 (2016)
8. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing. p. 291–304. STOC '85, Association for Computing Machinery, New York, NY, USA (1985). https://doi.org/10.1145/22145.22178, https://doi.org/10.1145/22145.22178
9. Grover, L.K.: Quantum mechanics helps in searching for a needle in a haystack. Physical review letters **79**(2), 325 (1997)
10. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing. p. 21–30. STOC '07, Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1250790.1250794, https://doi.org/10.1145/1250790.1250794
11. Kuo, P.C., Cheng, C.M., Tam, C.: Eip-7592: Precompile for falcon signature verification. Ethereum improvement proposals (2023), https://github.com/ethereum/EIPs/pull/8103/files
12. NIST: Post-quantum cryptography (2017), https://csrc.nist.gov/projects/post-quantum-cryptography

13. Palatinus, M., Rusnak, P.: Multi-account hierarchy for deterministic wallets. Bitcoin improvement proposals (2014), https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki
14. Palatinus, M., Rusnak, P., Voisine, A., Bowe, S.: Mnemonic code for generating deterministic keys. Bitcoin improvement proposals (2013), https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki
15. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172, https://doi.org/10.1137/S0097539795293172
16. Tan, T.G., Zhou, J.: Layering quantum-resistance into classical digital signature algorithms. In: Liu, J.K., Katsikas, S., Meng, W., Susilo, W., Intan, R. (eds.) Information Security. pp. 26–41. Springer International Publishing, Cham (2021)
17. Tan, T.G., Zhou, J.: Migrating blockchains away from ecdsa for post-quantum security: A study of impact on users and applications. In: Garcia-Alfaro, J., Navarro-Arribas, G., Dragoni, N. (eds.) Data Privacy Management, Cryptocurrencies and Blockchain Technology. pp. 308–316. Springer International Publishing, Cham (2023)
18. Thibault, L.T., Sarry, T., Hafid, A.S.: Blockchain scaling using rollups: A comprehensive survey. IEEE Access **10**, 93039–93054 (2022)
19. Unruh, D.: Computationally binding quantum commitments. In: Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35. pp. 497–527. Springer (2016)
20. Wuille, P.: Hierarchical deterministic wallets. Bitcoin improvement proposals (2012), https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki
21. Zoltu, M.: Eip-2718: Typed transaction envelope. Ethereum improvement proposals (2020), https://eips.ethereum.org/EIPS/eip-2718