Towards Quantum-Safe Blockchain: Exploration of PQC and Public-key Recovery on Embedded Systems

Dominik Marchsreiter
TUM School of Computation, Information and Technology
Technical University Munich
dominik.marchsreiter@tum.de

Abstract—Blockchain technology ensures accountability, transparency, and redundancy in critical applications, including IoT with embedded systems. However, the reliance on public-key cryptography (PKC) makes blockchain vulnerable to quantum computing threats. This paper addresses the urgent need for quantum-safe blockchain solutions by integrating Post-Quantum Cryptography (PQC) into blockchain frameworks. Utilizing algorithms from the NIST PQC standardization process, we aim to fortify blockchain security and resilience, particularly for IoT and embedded systems. Despite the importance of PQC, its implementation in blockchain systems tailored for embedded environments remains underexplored. We propose a quantum-secure blockchain architecture, evaluating various PQC primitives and optimizing transaction sizes through techniques such as public-key recovery for Falcon, achieving up to 17% reduction in transaction size. Our analysis identifies Falcon-512 as the most suitable algorithm for quantum-secure blockchains in embedded environments, with XMSS as a viable stateful alternative. However, for embedded devices, Dilithium demonstrates a higher transactions-per-second (TPS) rate compared to Falcon, primarily due to Falcon's slower signing performance on ARM CPUs. This highlights the signing time as a critical limiting factor in the integration of PQC within embedded blockchains. Additionally, we integrate smart contract functionality into the quantum-secure blockchain, assessing the impact of PQC on smart contract authentication. Our findings demonstrate the feasibility and practicality of deploying quantum-secure blockchain solutions in embedded systems, paving the way for robust and future-proof IoT applications.

I. Introduction

A Blockchain is a decentralized and distributed network that stores information using distributed ledger technology [1][2]. Information can be exchanged between participating nodes without trust in a centralized organization. The technology provides accountability, resilience and transparency by utilizing consensus algorithms and cryptographic secure operations [3].

Besides some emerging blockchain applications in financial services and cryptocurrencies the Internet-of-Things (IoT) is the dominant blockchain application and most researchers today associate blockchain applications with IoT [4]. Blockchain can secure IoT systems, e.g. in energy cyber-physical systems, vehicular cyber-physical systems for intelligent transportation systems and autonomous driving, aviation systems, supply chain systems and sensors [5]. Blockchains have specific benefits for applications with embedded systems in IoT: With identity management and governance mechanisms, IoT devices can simply be registered and identified using a shared ledger to tag devices to a specific user. Rights and ownership can be easily and securely transferred between the participating parties in the network. Blockchains allow for the simplification of security protocols. Solutions for industrial IoT with blockchains are already available [6]. The IBM report from 2025 points out that IoT would face trouble if there is no migration toward a decentralized way of networking and use a blockchain-like system to track and coordinate devices [7]. Manufacturers of IoT devices must integrate blockchain technology to track and coordinate the high amount of connected devices [8]. However, IoT blockchain applications are based on small and low-cost embedded devices that are restricted in the computing power that is needed to perform consensus and verification operations. Embedded systems are limited in energy consumption, data storage and have hardware limitations [9]. These prerequisites make the implementation of blockchains on embedded systems challenging. Foundational research on blockchain frameworks in IoT is still ongoing.

In addition, quantum computers can break classical cryptography in blockchains used to sign signatures and perform verifications. Classical blockchain's security leverages classical public-key cryptography (PKC) to deploy authenticated transactions. Widely used blockchains are still based on generic classical cryptographic primitives. This is also shown by the comprehensive analysis of the top 30 mainstream cryptocurrencies by [10].

PKC schemes like Rivest–Shamir–Adleman (RSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) are common primitives in blockchain protocols, relying on complex mathematical problems such as factorization large prime numbers (RSA) and finding the discrete logarithm of a random elliptic curve element (ECDSA), which are considered difficult for classical computers to solve. However, the rapid progress in quantum computing poses a significant threat to these systems. Shor's quantum

algorithm [11] can efficiently solve these mathematical problems in the future, rendering classical PKC vulnerable. To ensure the long-term security of trust management infrastructures, it's imperative to develop quantum-resistant blockchains. Post-quantum cryptography (PQC) offers one such solution, relying on mathematical problems resistant to quantum attacks and deployable on existing electronic devices. Recognizing this, the National Institute of Standards and Technology (NIST) has initiated the transition towards PQC to counter the quantum threat and is presently conducting the PQC standardization initiative [12]. Initially, 69 post-quantum algorithms were submitted by researchers worldwide in 2016, with only 7 algorithms advancing to the finalist stage [13]. By 2022, NIST identified four algorithms for standardization, including CRYSTALS-Kyber for key-encapsulation mechanism, along with CRYSTALS-Dilithium, Falcon, and SPHINCS$^+$ for digital signatures [14], [15], [16], [17]. The first draft of the PQC NIST standard was presented for publication in 2023 [18]. Each algorithm offers distinct trade-offs concerning performance, key size, and signature/ciphertext sizes. Additionally, two stateful hash-based signature algorithms, XMSS and LMS, have been standardized by the Internet Engineering Task Force (IETF) and subsequently integrated into a special publication NIST SP 800-208 in October 2020 [19], [20], [21].

Given the lengthy development and long lifespan of many blockchain applications, the integration of PQC solutions is essential. Building and converting to quantum-secure blockchains is necessary. Despite PQC primitives are not yet being standardized, assessing the performance characteristics and suitability of these algorithms for blockchain in different applications is crucial to ensure future usability. Standardized PQC solutions must cater to diverse blockchain application needs and requirements. While previous studies have benchmarked various PQC crypto-primitives [22], comprehensive integration into blockchain networks and thorough exploration and evaluation are yet to be undertaken. Integrating PQC and exploring different design options within existing classical blockchains presents a complex challenge. Especially for resource constraint environments like embedded systems and IoT blockchain applications efficient solutions must be developed.

The well known concept of public-key recovery (PKR) mode with ECDSA has not been thoroughly explored or analyzed within the context of PQC and blockchain. Presently, blockchains utilize ECDSA in PKR mode, enabling the derivation of a public key from an ECDSA signature along with specific function domain parameters. This approach offers the advantage of including only the signature in transactions and saving memory, thereby efficiently utilizing the limited bandwidth and storage capacities of blockchain environments. It is plausible that certain PQC algorithms could support this mode, potentially facilitating the integration of smaller transaction sizes within the blockchain.

In this work, we present for the first time a quantum-secure blockchain platform for embedded and computer systems able to integrate and explore the different blockchain configurations enhanced with the PQC algorithms Falcon, Dilithium, SPHINCS$^+$ and XMSS. We also analyzed for the first time and implemented PQC algorithms in publik-key recovery (PKR) mode and integrated them into our platform. Our quantum-secure blockchain platform allows a fast and scalable exploration of different configurations, allowing us to emulate the design and operational conditions of well-known classical blockchains. The contributions can be summarized as follows.

Our contribution: This work introduces a quantum-secure blockchain architecture built upon Post-Quantum Cryptography (PQC), detailing its design and evaluation on embedded systems. Additionally, it provides the first analysis of public key recovery within the context of PQC. In summary, our contributions include:

- Design and implementation of a quantum-secure blockchain for embedded systems able to support a wide variety of standardized PQC algorithms (by NIST and IETF);
- Performance and security exploration of PQC primitives for the quantum-secure blockchain. We show the different design alternatives, trade-offs and limitations of running a quantum-secure blockchain on embedded systems;
- First exploration of the PQC in Public Key Recovery (PKR) mode;
- We demonstrate a quantum-secure blockchain for authenticating users with smart contracts.

II. Related Work

The market offers over 20,000 cryptocurrencies, each providing decentralized services to users [23]. However, only a limited number of blockchains have taken steps to address the quantum threat or utilize PQC to ensure long-term security. Table I shows an overview of the most relevant 10 coins according to their market capitalization as listed by [24]. Tokens are not included since they are based on other blockchain platforms and adopt their cryptography.

The German Federal Office for Information Security (BSI) already addressed the need for quantum-secure cryptography in blockchains in [25]. Additionally, [26][27] presented how quantum computers put blockchain security at risk. [28] discussed the quantum-secure capabilities for Bitcoin and the results show that Bitcoin's PoW algorithm is resistant enough to a quantum computer attack; however, the elliptic curve signature scheme is at risk. It was pointed out that hash and lattice-based cryptography schemes are the only reasonable options concerning the sum of signature and public key lengths. Until now there is no final quantum-secure solution for Bitcoin.

TABLE I: Cryptocurrencies and blockchain platforms state-of-the-art. Presents the properties of having smart contracts (SC), being post-quantum secure (PQ) and the type of public-key cryptography (PKC)

Abbreviation	Name	Consensus mechanism	Smart contracts	PQ	PKC
BTC	Bitcoin	PoW	no	no	ECDSA
ETH	Ethereum	PoS	yes	no	ECDSA
SOL	Solana	PoS	yes	no	EdDSA
BNB	BNB	PoW	no	no	ECDSA
XRP	XRP	Federated Consensus	yes	no	ECDSA
AVAX	Avalance	PoS	yes	no	ECDSA
ADA	Cardano	PoS	yes	no	EdDSA
DOGE	Dogecoin	PoW	no	no	ECDSA
DOT	Polkatdot	NPoS	yes	no	ECDSA
TON	Toncoin	PoS	yes	no	ECDSA
	Hyperledger Fabric	variety	yes	no	ECDSA
	R3 CORDA	variety	yes	yes/no	ECDSA, SPHINCS
NEX	Nexus	PoW & nPoS	yes	yes	ECDSA, Falcon-512
	BitcoinPQ	PoW	no	yes	XMSS
QRL	Quantum Resistant Ledger	PoW	no	yes	XMSS
IOTA	IOTA	FPC	yes	yes/no	EdDSA (W-OTS+)

Some blockchains have already explored quantum-secure capabilities: In [29], the blockchain IOTA announced its quantum-resistance capability due to the integration of Winternitz One-Time Signature (W-OTS). However, this modification was undone due to implementation issues that forced the use of EdDSA. Therefore, the current IOTA implementation is no longer quantum-secure [29]. In 2018 [30] the cryptocurrency Quantum-Resistant Ledger (QRL) proposed an alternative to Bitcoin. QRL uses XMSS [19], an IETF specified PQC algorithm. While QRL is quantum-secure it is not possible to explore other stateless PQC NIST alternatives. In the same year the experimental branch of Bitcoin presented the BitcoinPQ network, a quantum-secure Bitcoin alternative based also based on XMSS [31]. The mainnet of the blockchain launched in 2018, but currently, there is no activity on the network anymore. Nexus and R3 Corda are non-quantum-secure blockchains but offer PQC as an optional feature to their users (e.g., SPHINCS$^+$-256 and Falcon-512) but claim these PQC algorithms impractical for many blockchain applications because of its relatively slow signing speed and huge signature output [32], [33].

Some works that refer to quantum-secure blockchains explore the PQC primitives theoretically without their full integration into a blockchain setup. The authors of [34] provide a broad review of blockchains and explore the performance of the NIST PQC round 2 standardization candidates for digital signatures and public-key encryption schemes. Results show that there is no large academic initiative on the application of quantum-secure implementations to blockchain and no single PQC algorithm provides all requirements e.g. small key size, short signature sizes, fast execution and low energy consumption to be a suitable successor to classical cryptography. A similar work was presented in [35].

Previous works also explore tailor-made PQC constructions for Bitcoin. In [36] a post-quantum secure algorithm was implemented in a Bitcoin like instance. The construction is based on a quantum-secure signature scheme that combines a hash-based one-time signature with Naor-Young chaining [37]. A comparison with XMSS and SPHINCS$^+$ was performed. Results show a huge performance overhead, which turns the use of the new construction for blockchain applications impractical. This results in smaller signatures and better performance of the algorithm compared to hash-based signature schemes.

In [38], the authors present a post-quantum blockchain (PQB) construction for smart city applications. They proposed a post-quantum PoW consensus protocol and used a lightweight public key-reduced identity-based signature scheme. The transaction protocol for the post-quantum blockchain is designed with the identity-based signature scheme Rainbow, which is already broken and no longer considered secure [39]. This work does not create a real blockchain to perform analysis and performance evaluation. Instead, they used theoretical analysis and simulation experiments with Magma [40], a system language for computational algebra. Certificateless IBE with lattices was also explored in [41]. The paper presents a new post-quantum-resistant blockchain framework with a consensus algorithm based on aggregated signatures.

In [42] the NIST PQC algorithms are compared with ECDSA in a Bitcoin exchange scheme. However, this work only focused on the NIST Level 5 implementations. The results showed that Falcon-1024 is a suitable choice for blockchain cryptography. It is unsure if this also applies for levels 1 and 3, and for blockchain implementations with small embedded devices.

Some works proposed completely new PQC algorithms that are not part of the standardization process by NIST [43][44]. Since they are not NIST-approved the trust in the security is unsure.

Besides using PQC to secure blockchain cryptography, researcher proposed solutions using quantum-key distri-

bution and utilizing the quantum advantage [45][46][47]. However, this requires urban fiber networks and is besides an experimental realization not suitable for broadly embedded blockchains.

A theoretical exploration of the capabilities of PQC in Blockchain for IoT is shown in [48]. The work is based on the Helium blockchain but provides no measurements.

There is a lack of research in the development of post-quantum-resistant blockchain frameworks with PQC and embedded systems. Until now no known research and implementation has been performed on quantum-secure blockchain with embedded systems. There are no results for a comparison of all NIST PQC algorithms in an embedded systems environment for blockchains. Moreover, the behavior and feasibility of PQC and smart contracts are evaluated in this work.

III. Quantum threat in Blockchain

A blockchain is known as a chain of data that links blocks together in a queue. Every node in the network has a copy of this chain in a file called a ledger. It implements a peer-to-peer (P2P) communication network between multiple nodes. Figure 1 illustrates a schematic overview of a blockchain.

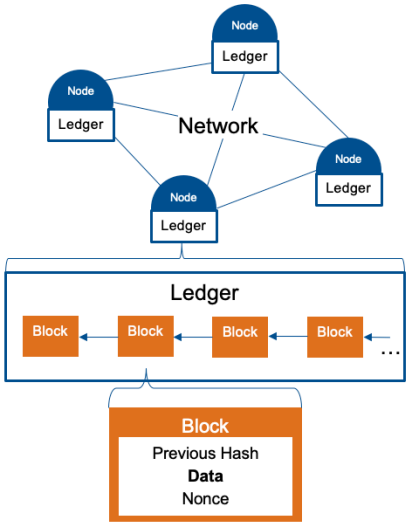


Fig. 1: Blockchain technology schematic.

Different participants can agree on a course of action via an insecure channel. To archive a distributed consensus in an unsafe environment blockchains use a consensus algorithm. One of the most used algorithms is the Proof-of-Work (PoW) algorithm which creates trust using computation power. Miner try to find a nonce H_n such that the relation 1 is satisfied [49]:

$$m = H_m \wedge n \leq \frac{2^{256}}{H_d} \; with \; (m, n) = PoW(H_{\not d}, H_n, d) \quad (1)$$

The PoW function calculates two values: m the mixHash and n a pseudorandom number. $H_{\not d}$ is the new block header, d is a large data, H_d is the new block's difficulty. To improve blockchain sustainability, alternative consensus algorithms are integrated in Blockchains (e.g., Proof-of-Stake/PoS, Proof-of-History/PoH, Proof-of-Burn /PoB or Proof-of-Transfer/PoT). Ethereum uses PoS since 2022 [50]. It has been shown in [50] that the adoption of this algorithm saves up to 95% of energy when compared to PoW. Nevertheless, the integration of PoS might increase the complexity in the blockchain implementation.

The blockchain is characterized by a sequence of different events. A transaction is initiated by defining the value of the transaction fields which include the recipient address and transaction value. As in [51], a transaction is performed through the following steps:

1) Sender generates a new transaction.
2) New transaction is broadcast to all nodes and miners.
3) Miner nodes collect new transactions into a block.
4) Each miner node works on finding a nonce with proof-of-work for its block.
5) When a node finds the correct nonce, it broadcasts the block to all other nodes.
6) Nodes accept the block only if all transactions are valid and not already spent.
7) Nodes express their acceptance and attach the block to the ledger.
8) Nodes working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

The transaction for a three-node setup is shown in Figure 2. Only when a block is mined and accepted it will be attached to the ledger.

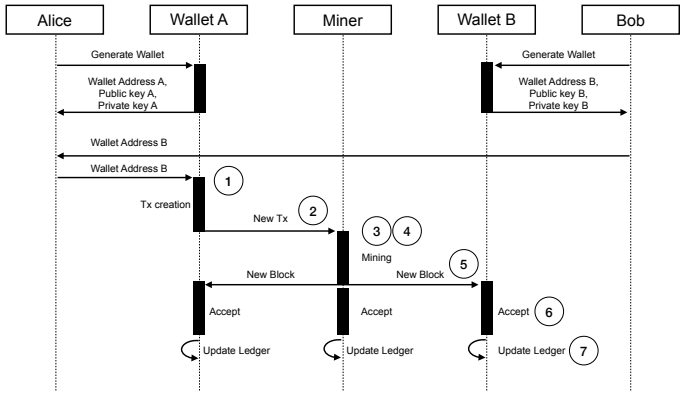


Fig. 2: Detailed flow describing a transaction between two users (Alice and Bob) and a miner node.

The classical cryptographic primitives used in blockchain technology are based on hashing and digital signature algorithms. Hash functions are used in two phases. The first is at the consensus algorithm (PoW) where hashes generate the block digest value. The second one is in the wallet generation and transaction

process. Bitcoin uses Keccak-256 [52], the predecessor of SHA-3 hash algorithm [53]. Keccak-256 and SHA-3 are commonly used in most of the blockchains.

A. The Quantum Threat

When cryptographically-relevant quantum computers are available, the quantum algorithms of Grover [54] and Shor [11] can be used to half the security of symmetric cryptography and break PKC, respectively. While classical computing will find the answer in $O(N)$ steps, where N is the size of the function's domain, the quantum computer executing the Grover's algorithm will need only $O(\sqrt{N})$ steps. Grover affects the algorithm AES (Advanced Encryption Standard) and hashing algorithms (e.g., SHA-2, SHA-3). However, the classical symmetric algorithms are considered secure as long as the key sizes are long enough. In practice, this means we need to double the key size to reach the same security strength. AES-256 and SHA3-256 are considered quantum-secure.

In contrast, Shor's algorithm can solve the hard problem of traditional PKC in polynomial time [55]. This means algorithms such as RSA and ECDSA will be broken [56]. These algorithms are used for signing information and key exchange by many applications. It is estimated that a cryptographically-relevant quantum computer will impact thirty-six percent of all Bitcoins that are on wallet addresses with exposed public keys [57].

Quantum computing can not get a significant exponential advantage in brute-force searching and collision-finding problems. Therefore, the hash functions are considered to be quantum-secure to exponential speedup. Bitcoin's hash function SHA-256 can be partially reduced in security by the quantum attack, but it is enough to double up the digest size to be safe in the future. Quantum computing is unlikely to impact the Bitcoin's PoW system. To strengthen this point: Optimistic estimates for 2028 for the speed of a quantum computer with 4,400,000 Qubits and a low error rate results in 13.8 GHashes/s [28]. Nowadays, ASICs used for blockchain mining are already able to perform 14 THashes/s which is a lot faster than what a single quantum computer could be able to calculate in the future [28]. A powerful quantum computer has the potential to undermine the hash function (used in the PoW and wallet generation/transaction processes) as well to break and forge the signature schemes. While quantum computers are not able to provide an exponential advantage in collision-finding problems [35] and therefore by doubling the hash size (e.g., from SHA-128 to SHA-256) the hashes can be patched and continue being secure, the signatures must be modified. Post-quantum signatures are required [58][59].

Digital signatures are the public-key cryptographic primitives used for authenticating communication parties, to provide data integrity and to avoid communication repudiation. The signature value is calculated based on the data and a secret key known only by the signer. Usually, key pairs sk and pk are generated (key generation). The keys used for encryption/decryption and signing/verifying are commonly different. The private key used for signing is known as the signature key, while the public key is the verification key. The signer uses the signing key and the sign algorithm to generate a signature (sign). The authenticity of the message is checked through the verification process (verify). Since a digital signature is created by the private key sk of the signer, a signed transaction cannot be repudiated in the future. Different classical digital signature algorithms are used in Blockchain. It includes the Elliptic Curve Digital Signature Algorithm (ECDSA) and Edwards-Curve Digital Signature Algorithm (EdDSA) with curves secp256k1, curve25519 or ed25519 and RSA. Table II lists the key sizes of classical signing algorithms used in blockchains that are threatened by quantum computers.

TABLE II: Classical algorithms: Key sizes for 128-bit security level that threatened by quantum computers. The values in brackets are the uncompressed version. All sizes in bytes.

SIG Parameter Set	Secret key	Public key	Signature
ECDSA Secp256k1	32	33 (65)	64
ECDSA curve25519	32	32	64
EdDSA ed25519	32	32	64
RSA 3072	384	384	384

ECDSA, with a security level of 256 bit, has a 32 bit long private key. The public key consists of two points x,y on the elliptic curve, each 32 byte long and an additional prefix of 1 byte ($pu = prefix$ [1 $byte$] + x [32 $bytes$] + y [32 $bytes$]). This prefix is used for serialization as defined by the Standards for Efficient Cryptography Group (SECG) [60].

IV. Post-Quantum Cryptography

New mathematical hardness assumptions are required to construct quantum-secure PKC schemes. PQC refers to a set of algorithms that rely on mathematical problems that are believed to be hard to break using a large-scale quantum computer. Current PQC algorithms can be executed on traditional computers and offer security against both classical and quantum attacks. There are five post-quantum families:

- Codes: Based on the hardness of decoding a general linear code (e.g., Classic McEliece, BIKE, HQC);
- Hash-trees: Based on the hardness of the security of hash functions (e.g., SPHINCS$^+$);
- Multivariate: Based on the hardness of solving systems of multivariate polynomials over finite fields (e.g., Rainbow, GeMSS, Picnic);
- Isogeny: Based on the hardness of finding the isogeny mapping between two super singular elliptic curves with the same number of points (e.g., SIKE); and

- Lattices: Based on hard problems described in lattices (e.g., short-vector-problem (SVP), closest-vector-problem (CVP) and Learning-with-Errors (LWE) (e.g., Kyber, NTRU, SABER, FrodoKEM, NTRU Prime, Dilithium, Falcon).

A. NIST Standardization Process

In 2017, the NIST announced a standardization process for PQC [12]. The goal of this process is to standardize at least one KEM (Key exchange mechanism) and at least one signature scheme. While for the first round of the standardization competition 69 submissions were selected, in the second round 26 submissions remained. In July 2020, the third round of this competition was announced. The remaining PQC candidates were classified into two categories: i) finalist, which includes the set of PKE/KEM and signatures that are candidates for the NIST standardization of 2022; and ii) alternate, which are considered by the NIST as promising solutions but require further development for possible future standardization. For the category of PKE/KEM, four finalists (Classic McEliece, Kyber, NTRU, SABER) and five alternate candidates (BIKE, FrodoKEM, HQC, NTRU Prime, SIKE), and for the category of signatures, three finalists (Dilithium, Falcon, Rainbow) and three alternate candidates (GeMSS, Picnic, SPHINCS$^+$) were announced.

NIST has defined five different security levels (from 1 to 5). Each security level is defined by a comparatively easy-to-analyze classical reference primitive (e.g., AES, SHA) as follows [12]:

- Level 1: Equivalent to AES-128
- Level 2: Equivalent to SHA-256
- Level 3: Equivalent to AES-192
- Level 4: Equivalent to SHA-512
- Level 5: Equivalent to AES-256

Each post-quantum cryptosystem is instantiated through different parameter sets.

In 2022, NIST announced that the PQC standard will include Kyber [14] as a KEM and Dilithium [61] Falcon [16] and SPHINCS$^+$ [62] as signatures. In the selection are two lattice-based algorithms (Dilithium and Falcon) and a hash-based algorithm (SPHINCS$^+$). The complete set of algorithms is composed by Dilithium2, Dilithium3, Dilithium5, Falcon-512, Falcon1024, SPHINCS$^+$-SHA256-128, SPHINCS$^+$-SHA256-192 and SPHINCS$^+$-SHA256-256. The two stateful hash-based signature algorithms XMSS and LMS have been standardized by IETF and further integrated in the NIST SP 800-208.

SPHINCS$^+$ has a variety of configuration options. The SPHINCS$^+$-simple is faster as no pseudorandom function (PRF) is used to generate bitmasks. In addition, a user can select between SHA-256, Shake-256 and Haraka as hash functions (note that Haraka is not a NIST-approved hash function). In addition, the SPHINCS$^+$ can be set to small (s) or fast (f) options, which corresponds to the size-optimized or speed-optimized code, respectively. The signature sizes differ between the two options (for the fast (level 1,3,5): 17088 Byte, 35664 Byte, 49856 Byte and for small(level 1,3,5): 7856 Byte, 16224 Byte, 29792 Byte). In table III we focus only on the small versions.

XMSS is a stateful quantum-secure signing algorithm. XMSS uses the Winternitz One-Time-Signature scheme (W-OTS) and hash trees to generate public keys. In contrast to traditional signature schemes, XMSS is based on a tree structure characterized by the parameter height h. This value determines the number of messages that can be signed with one key pair. A higher value of h will increase the number of signatures n_{sig} allowed for a single key pair $n_{sig} = 2^h$, e.g., a three hight $h = 20$ can sign 2^{20} with one key pair. However, the signature size is increased and the key generation is slowed down. XMSS in QRL [30] (with a 128-bit of security level) uses the Winternitz parameter $w = 16$ and a tree of height $h = 10$. They report in [63] a verification time of 15 ms when only a single thread is used. A XMSS signature has a length $4 + n + (len + h) * n$ [bit] for $n = 32$, $h = 20$ and $len = 67$. XMSS-MT-SHA2_20/2_256 is used in the QRL blockchain [30]

The private key, public key and signatures sizes for all algorithms are displayed in table III.

TABLE III: PQC algorithms: Configuration parameters. All sizes in bytes.

SIG Parameter Set	Public key	Secret key	Signature
Falcon-512	897	1281	690
Falcon-1024	1793	2305	1330
Dilithium2	1312	2528	2420
Dilithium3	1952	4000	3293
Dilithium5	2592	4864	4595
SPHINCS$^+$-SHA256-128s	32	64	7856
SPHINCS$^+$-SHA256-192s	48	96	16224
SPHINCS$^+$-SHA256-256s	64	128	29792
XMSS-MT-SHA2_20/2_256	213	35	352

PQC key sizes are bigger than classical ECDSA key sizes. SPHINCS$^+$ secret key size doubles the public size since the secret key also stores an instance of the public key. In this construction the signing function uses the public seed and the root for fault attack mitigation.

V. Quantum-Secure Blockchain

The quantum threat will impact digital signatures used in classical blockchain frameworks. While hashes can be patched by increasing the key sizes, digital signature algorithms must be changed. Our solution to a quantum-secure blockchain integrates different post-quantum signature primitives from the NIST standardization process. While NIST and IETF already selected the PQC signature algorithms (Dilithium, Falcon, SPHINCS$^+$, XMSS) the integration on the blockchain and its impact on the overall system has not yet been widely performed. To present a useful exploration of the impact of the PQC integration in blockchain, our quantum-secure blockchain implements

all the core concepts and implementation criteria of Bitcoin and Ethereum. Bitcoin (40%) and Ethereum (19%) together represent about 60% of the current blockchain market [24]. The implementation criteria of our quantum-secure blockchain are summarized as follows:

- Decentralization: Property of the system to resist the failures of the Byzantine Generals' problem. The system should be designed to reach the correct consensus if more than 50 % of the mining nodes are honest.
- Immutability: The ability for a blockchain ledger to remain an indelible, permanent and unchanging history of transactions.
- Shared Ledger: A Database, called Ledger, stores the blockchain in a file. Each full node should have a copy of the ledger.
- Peer-to-Peer Network: A network between all nodes needs to be established that can handle data traffic between wallets and miner clients.
- Smart Contract Functionality: The blockchain should be enabled for smart contracts to execute and store them on the blockchain. These smart contracts can later be used for other blockchain applications, e.g. for IoT applications.
- Exchange of Value: Coins represent a digital currency that can be used to pay transaction costs or can be exchanged between wallet accounts.
- Post-Quantum Cryptography: PQC should be used to make the blockchain quantum-secure. It should be possible to use the NIST-selected PQC algorithms in the blockchain to test and compare them to each other.
- Owner-Controlled Asset: Only the owner of an asset can transfer that asset. The owners are the holders of a particular set of keys. Not even a node operator can transfer an asset because only the user is in charge of the private key.

Some production-level requirements are left out of the scope of the paper. The security against public attackers is left as future work.

A. Transactions and Blocks

Transactions (T) hold sets of information that are needed to perform a transaction of coins between two parties. It includes information about the sender (T_t), the sending amount (T_v), Data (T_d), the gas limit (T_g), the gas price (T_p), a signature (T_r, T_S) and a Nonce (T_n). The gas price refers to the amount of value that must be paid to the miner for processing transactions and the gas limit is the amount of work estimated a miner does. A transaction is $T = \{T_t, T_v, T_d, T_g, T_p, T_r, T_s, T_n\}$. Figure IV presents an overview of all components of a EIP-2718[1] transaction type. An Ethereum Improvement Proposal (EIP) specifies a standard and potential new features for Ethereum.

[1]https://github.com/ethereum/EIPs/blob/master/EIPS/eip-2718.md

TABLE IV: Contents of an EIP-2718 transaction.

Short	Name	Description
T_t	Address	20 Byte of hash value of the public key
T_{pu}	Public Key	Public key of the sender
T_v	Value	Amount of Value sent from sender [Wei]
T_d	Data	Data field for e.g. smart contract code
T_g	gasLimit	maximum amount of gas that should be used for execution (usually 21000 gas for one transaction)
T_p	gasPrice	Price per unit of gas for execution [Wei]
T_r,T_s	Signature	Signature to determine the sender
T_n	Nonce	Number of transactions of this account

A wallet address is encoded with 20 bytes since it corresponds to the hashed value of the public key. The receiver is identified by its 20 bytes-long wallet address.

Figure V shows the chosen structure of transactions and blocks for our quantum-secure blockchain. A block stores the previous hash, the transactions and a nonce. The nonce for the block is different from the transaction nonce and is a value that is needed for mining. The block format differs between different blockchain platforms. Bitcoin is composed by a block header (existing in the previous hash), the Merkle Root of the transactions in the block, a timestamp and a nonce. The block header and the Merkle tree hash enable the participation of lightweight nodes in the blockchain, avoiding the need to store all transactions on the devices. Ethereum's block structure is more complex. The Block includes three Merkle trees and each one is composed of 15 distinct fields. It is made of a header that contains the metadata, followed by the transaction list that makes up the bulk of the block size. Since the Merkle tree scheme remains secure from quantum computer attacks, we use the structure from Bitcoin.

The block size is typically limited by the blockchain. This is necessary because if the size is not restricted, less-performing nodes in the network cannot keep up with the block computation due to speed and storage requirements. This would lead to more centralized computing performance, which is against the decentralization characteristic of the blockchain. Bitcoin has a block limit of 2 MB in size[64]. Ethereum restricts the block size not in terms of the maximum data size per block but rather by using a block gas limit. It is usually set to 30 million gas, which is two times the target block size of 15 million gas. We expect large blocks because of the huge signature sizes and the addition of the public keys in the transactions. Therefore a high value of 10 MB was chosen for the quantum-secure blockchain.

B. Integration of PQC

PKC is used in blockchains to generate wallets, sign messages and verify signatures. These operations are shown in figure 3. Digital signatures algorithms must be replaced by PQC signatures algorithms. The quantum-secure blockchain integrates PQC at all cryptographic

TABLE V: Transaction format (left) and block format (right). All sizes are in byte

Name	Size
Recipient address	20
Value	8
Data	0
gasLimit	2
gasPrice	5
Signature	sig
Sender	pu_key
Nonce	1

Name	Size
Previous Hash	32
Transactions	tx
Nonce	4

calculation steps and modifies the signing and verification operations by using a PQC primitive instead of the PKC signing operation and integrates the PQC signing algorithms selected by the NIST competition and IETF. Figure 3 shows the detailed flow describing the wallet generation, signing and verification operations. It includes the different functionalities of the applicant, wallet, random number generator, SHA-3 and PQC algorithms.

In the first operation, the wallet generation, the applicant requests a new wallet address. The random number generator produces a random string of bits which is used as a private key. The private key is used as input for the PQC signature algorithm. The PQC algorithm prodces the public key from the provided secret key. This public key gets hashed by a SHA-3 function into a 20-byte long wallet address. The second operation, signing, is triggered by a request of the applicant to the wallet. The transaction is serialized and hashed. The PQC algorithm uses this value to produce a signature of this corresponding message. The transaction signature together with the hash is sent back to the wallet. This signature is used during the verification of a transaction. Finally, in the verification operation, the hash of the transaction is calculated. The second hash is calculated using the provided signature and the public key. When the PQC in public-key recovery mode is used, the public key must be calculated from the signature. This operational step needs to be added and requires the signature (marked in grey in Figure 3). Finally, both hashes are checked and compared to check the validity.

Section III-A identifies that the security of hash functions is partially reduced by quantum attacks through the execution of the Grover's algorithm. To develop a quantum-secure blockchain a long enough hash digest can be used. This will ensure a quantum-secure PoW algorithm. SHA-256 is considered long enough to be quantum-secure and will be used in this blockchain as a secure hash function.

C. PQC in Blockchain Virtual Machines.

A VM can also perform computations in a blockchain platform, thus making it possible to use cryptographic operations in smart contracts. Note that a smart contract does not have any privacy requirement since everyone is able to have access to the transactions and the deployed

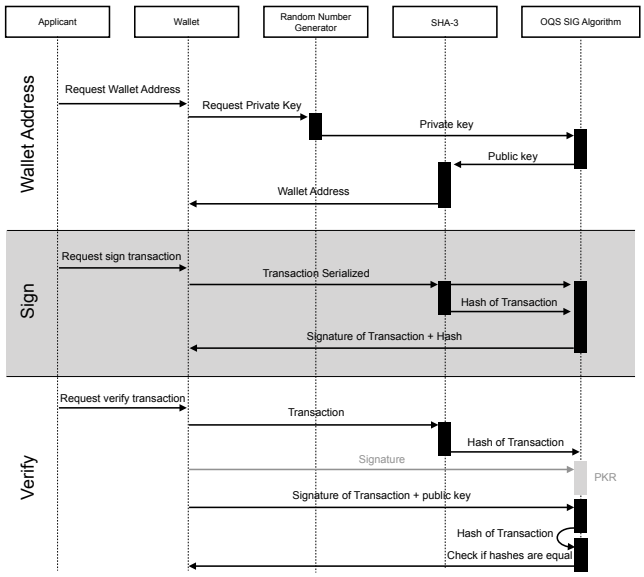


Fig. 3: Detailed flows describing wallet generation, transaction signing and verification with PQC algorithms.

code on the blockchain. If a private key is generated in a smart contract, the key can be seen by everyone who has access to the blockchain. All data sent to the VM in a transaction can be de-compiled by everyone. Signing in a smart contract scenario would necessarily expose the private key. Therefore the cryptographic functionality is executed outside the VM and only the hash, signatures and public keys are sent to the VM. Public key operations like checking signatures, proofs and hashes are useful in some smart contract applications. To perform cryptographic operations with this data in the VM, algorithms need to be deployed in the VM. PQC algorithms are very data-intensive and are considered expensive (in terms of the gas price). Moreover, a blockchain has a block size limit which limits the size of the algorithm code lines. Pre-deployed smart contracts with cryptography algorithms can be developed before their release. These smart contracts are available to everyone and can be implemented with the consensus of all participants in the blockchain network. Cryptographic algorithms (e.g., Keccak256, SHA-256, public-key-recovery and EC-recovery) are pre-deployed in the VM. In order to transition to PQC, the verification functions must be correctly implemented in the VM. In our quantum-secure blockchain, we pre-deployed the smart contracts SHA-256 and public-key-recovery for Falcon.

D. Quantum-secure Blockchain Implementation

The system has two main software containers: A wallet client and a miner client. These can be executed on different devices and run apart from each other. Figure 4 and Figure 5 show the dependencies between the different software modules for the miner and wallet container.

The wallet container handles the user accounts and requests to the blockchain. A user interacts with his wallet,

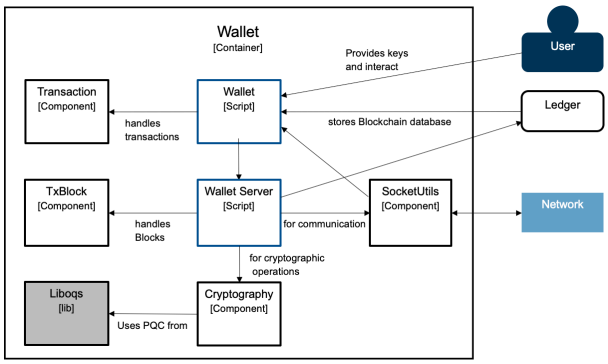


Fig. 4: Architecture design of the wallet client.

e.g., to initiate transactions. The wallet container executes a wallet client application which does all operations. The block validity checks, ledger updates and simultaneous block attachments are done by a wallet server module. This module waits for new incoming mined blocks and checks the block's correctness before attaching blocks to the blockchain. There are two objects for transactions and block objects. The transaction object has the structure of EIP-2718 shown in table IV. The block object stores the hash value H, the transactions T_n and the nonce N referring to table V ($B = \{H, T_0, T_1, ..., T_n, N\}$). A network connects all participating wallet and miner clients. The network functionalities are handled by the component SocketUtils. The cryptographic functionalities are implemented in a software component called cryptography. For a clean and secure implementation of the PQC algorithms, the library liboqs from the Open-Quantum-Safe project (OQS) [22] is implemented in the cryptography software component. The algorithms are written in C and based on the pqclean [65] implementation. The Falcon public key recovery (PKR) functionality is written in C based on the NIST submission documents [16]. The blockchain is stored in a JSON database file, which is called Ledger.

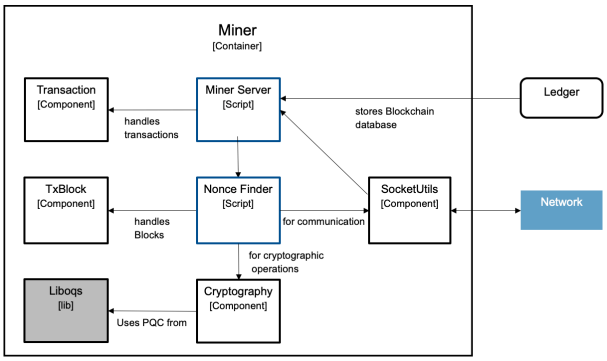


Fig. 5: Architecture design of the miner client.

The miner container uses similar software components to the wallet client. There are two main scripts: the miner server and the nonce finder. The nonce finder task operates the consensus algorithm. The quantum-secure blockchain uses the Proof-of-work consensus algorithm. This PoW algorithm is integrated in the TxBlock component. The miner server handles new incoming transactions and creates blocks with new transactions. Depending on the predefined block size limit B_{max} transactions are wrapped in a block before they are processed by the nonce finder task.

VI. Public-Key Recovery

To minimize the transaction payload, different blockchains (e.g., Bitcoin and Ethereum) use a special mode of ECDSA called public-key-recovery (PKR). PQC in PKR mode has not been discussed nor exploited to build efficient quantum-secure blockchains. The PKR reduces the total key size in transactions which is relevant for the communication efficiency in blockchain environments. In order to verify the signature, the signature and the signer public key are required by the verifier. In a typical public-key infrastructure (PKI), the public key is stored in a certificate that is signed by a certificate authority and can be requested by the receiver. However, due to the high amount of public keys, this is no longer possible in a blockchain environment. The public key and the signature are transmitted together in a transaction, resulting in big transmitted packets. A classical transaction would have a signature $sig = \{r, s\}$ and a public key pk. ECDSA in PKR mode eliminates the need of adding the public key because the public key can be recovered from an extended signature. This is very beneficial in bandwidth-constrained environments like blockchains where the transmission of public keys can not be afforded. The public key can be recovered by calculating [60]:

$$pk = r^{-1} * (sR - eG) \tag{2}$$

To perform PKR, the elliptic curve domain parameters $T = \{p, a, b, G, n, h\}$, a message M and the ECDSA signature $\{r, s\}$ are required. G is part of the description of the elliptic curve. From the message M the e can be computed in a similar way to the ECDSA signature verification. R is a elliptic curve point that has r as the x-coordinate. However, the elliptic curve can have 0, 1 or 2 points (R and R') with the x-coordinate r. These points can be valid public keys candidates. To find the correct point, a byte v with the parity and finiteness of the coordinates of the curve point is added to the signature. The exchanged information corresponds to the extended signature $sig = \{r, s, v\}$ and the signed message. It is required that the sender's public key and the signature are integrated on a single transaction.

$$\boldsymbol{c} = \mathrm{NTT}^{-1}(\mathrm{NTT}(\boldsymbol{a}) \odot \mathrm{NTT}(\boldsymbol{s})) \tag{3}$$

Where $\odot$ denotes a coefficient-wise multiplication and NTT^{-1} is the inverse NTT.

PKR mode for PQC algorithms has not yet been explored. If a PQC-PKR mode is used, the overall size overhead of the transaction can be reduced. This will

enable the implementation of efficient quantum-secure systems. We are going to discuss the PKR possibilities for different NIST PQC candidates.

Dilithium. Dilithium is based on the Fiat-Shamir with Aborts approach [61]. Dilithium mathematical operations are performed in the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\ (x^n + 1)$, where n and q are both integers. For the Dilithium's key generation, the signer samples a matrix with $k \times l$ polynomials in the ring R_q where $q = 2^{23} - 2^{13} + 1$ and $n = 256$. Two vectors $\mathbf{s_1}$ and $\mathbf{s_2}$ are sampled with coefficients in the range of R_q and maximum size η. The final step computes $\mathbf{t} = \mathbf{As_1} + \mathbf{s_2}$. Such that the public key pk is $(\mathbf{A}, \mathbf{t})$ and the vectors $\mathbf{s_1}$ and $\mathbf{s_2}$ form the secret key sk. The signing key is used to sign a message of 32 bytes. Two values are calculated, the signature z and a value c. The resultant signature z can be rejected if a dependency to the secret key is detected. Then a new signature must be calculated. The verifier validates the signature if the coefficients of z satisfy some boundary conditions and if c correspond to the hash of an intermediate calculated valued $\mathbf{w_1'}$ as follows.

$$w_1 = HighBits(Az - ct, 2\gamma_2) \tag{4}$$

$$[||z||_\infty < \gamma_1 - \beta]\ and\ [c = H(M||w_1)] \tag{5}$$

Note that the $\mathbf{w_1'}$ can be used to retrieve the public key (A, t). However, retrieving w_1' from c is impossible since it is used in a hash function that can not be reversed. Dilithium can not be used to perform public key recovery from the signature.

Falcon. Falcon did not propose PKR as part of the specification for NIST but describes the possibility to perform PKR mode in their own specification [16]. The mathematical operations in Falcon [16] are performed in the NTRU ring. The key generation is based on solving the so-called NTRU equation, which includes the gaussian sampling of elements of the polynomials. Four polynomials are created and integrated in the matrix B. Then an LDL tree T is computed and normalized. To sign a message m, the hash to a point $c \leftarrow H(r||m, q, n)$ is performed and the vector t is computed as $\mathbf{t} = (\mathrm{FFT}(c), \mathrm{FFT}(0)) * \mathrm{FFT}(\mathbf{B})^{-1}$. With z generated by a Fast-Fourier sampler, the signature s is set to $\mathbf{s} = (\mathbf{t} - \mathbf{z})\mathbf{B}$. The signature's elements are Gaussian distributed and s will rejected, if does not meet a boundary condition. To verify the Falcon signature, two checks are performed. First, it is verified that the signature is not perpendicular to some condition. Secondly, a boundary condition is also verified. If both conditions are met, the signature is accepted. Otherwise, it is rejected.

The signature of the Falcon algorithm is $sig = \{r, s\}$ with $s = s_2$. The second part $s = s_1$ is not added to the signature because it is unnecessary in the normal mode. In order to use the public key recovery mode, the signature $sig = (s_1, s_2, r)$ must be extended with both s_1 and s_2. This doubles the total signature size when compared to the normal Falcon mode. However, it makes the public key very compact because only requires to send the hash of the public key $pk = H(h)$ instead of the whole key. This makes the public key extremely compact $|pk| = 2\lambda$. The combination of both $|pk| + |sig|$ is in theory about 15% shorter. The reduction results from the trade of h with s_1. While h cannot be reduced in size, s_1 can be compressed using the same function as for s_2 ($s_i = Compress(s_i)$) which results in a 35% reduction in size.

The public key h can be recovered by the following equation from the signature (r, s_1, s_2):

$$h = s_2^{-1}(HashToPoint(r||m, q, n) - s_1) \tag{6}$$

To perform this calculation the hash of the public key $H(h)$ needs to be sent in addition to the signature. Moreover, $r = \{0, 1\}^{320}$ is a random salt, m is the signed message, q is a modulus $q \in N$ (in Falcon $q = 12289$) and n is the ring degree (512 or 1024). The verifier accepts the retrieved public key if both assumptions are fulfilled:

- (s_1, s_2) is short
- $pk = H(s_2^{-1}(HashToPoint(r||m, q, n) - s_1)$

To verify that the pk is correct, the hash of the new public key can be calculated and compared $H(h) = H(pk)$.

TABLE VI: Falcon normal compared to public-key-recovery mode. All sizes in bytes.

Version	Pub key	Signature			Add	Sum
	h	r	s_1	s_2		$\sum$
Falcon-512	897	40		625	1	1563
Falcon-512 PKR		40	625	625	1+32	1323
Falcon-1024	1793	40		1239	1	3073
Falcon-1024 PKR		40	1239	1239	1+32	2551

Table VI shows the different results. It can be observed that the reduction for Falcon-512 is 15,36% and for Falcon-1024 16,99%. The additional 1 byte corresponds to the header.

SPHINCS$^+$ and XMSS. In hash-based algorithms, it is not natively possible to shrink the size of the public key. Both algorithms, XMSS and SPHINCS$^+$, use public key recovery functions already in their basic code foundation. It is not possible to reduce the public key for these two algorithms similar to PKR.

VII. Experimental Results

In the experimental results section, we perform measurements and calculations to understand the behavior, constraints and benefits of a quantum-secure blockchain with PQC. First, we describe our experimental setup. Second, we measure the wallet, transaction and block size for all algorithms. Third, we measure the speed of the cryptographic algorithms. Fourth, we evaluate the performance of the blockchain for the different configurations on

a computer-based setup and an embedded device-based setup. Last, we measure the impact of PQC in smart contracts.

Table VII gives an overview of the parameters we were interested in.

TABLE VII: Test parameters with description for experiments.

Symbol	Parameter	Description
t_{keygen}	Keygen/sec	Test the key generation time of all PQC algorithms
t_{sig}	Signs/sec	Perform example signing operations with all PQC algorithms and measure the time
t_{verify}	Verify/sec	Perform example verification operations with all PQC algorithms and measure the time
len_W	Wallet size	Measure the size of a Wallet account for a user
len_T	Tx size	Measure the transaction size for a 1 coin transaction between two accounts
len_B	Block size	Measure the block size for different numbers of transaction
tps	TPS	Measure the speed of the blockchain by calculating the Transactions-per-second for each algorithm

A. Experimental Setup

The experimental results for the quantum-secure blockchain were tested on a computer with a Skylake processor Intel Core i7 6.Generation i7-9750H, configured to use its 6-cores running at 2,6 GHz, and 16 GB DDR4 RAM memory. A second setup was created to run a blockchain on embedded devices. We used the Raspberry Pi 3b with a ARM Cortex-A53 CPU running at 1,2 GHz. For the software implementation of the post-quantum algorithms, we used the open-source C library "liboqs" from the Open-Quantum-Safe project [66]. Python 3 bindings for liboqs are provided on GitHub for this library[2] The classical algorithm ECDSA was used from the standard Python cryptography library which is a package that provides cryptographic recipes and primitives to Python developers. XMSS was used from the xmss-reference implementation[3], companying RFC 8391, XMSS: eXtended Merkle Signature Scheme. For Falcon in public key recovery mode, a Python implementation of the signature scheme Falcon was used[67]. The Remix IDE[4] for Ethereum was the software tool to create and deploy smart contracts. These smart contracts were written in the programming language Solidity and the correctness was verified by a Remix virtual machine.

B. Wallet, Transaction and Block Sizes

Transaction size. We measured the size of the blockchain transactions for the set of PQC algorithms. Figure 6 compares the required memory space for a blockchain transaction. The transaction size was calculated for a 1-coin transaction between two accounts. The value field was filled with a number equivalent to 1 Ether (1000000000000000000 Wei = 0xDE0B6B3A7640000). The data field is 0 bytes since no smart contract was added to the transaction. The gas limit is set to 21000 (i.e., 0x5208) and the average gas price of 15 Gwei (i.e., 0x37E11D600) is assumed. As this is the first transaction a wallet account produces, the transaction counter (nonce) is set to 1 (0x01).

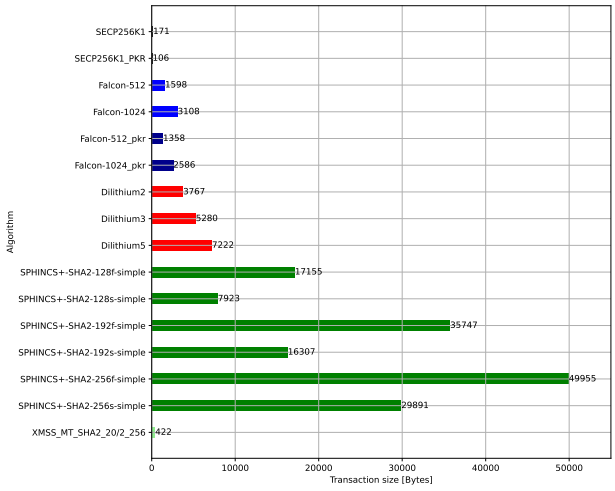


Fig. 6: Size of a transaction with a value equivalent to 1 Ethereum

The results show that transactions on a quantum secure blockchain are bigger than transactions with ECDSA. The size increases to more than 10 times the classical size. The smallest transactions are achieved when using the XMSS algorithm in the blockchain. The best stateless PQC algorithm is Falcon. Moreover, Falcon in PKR mode can reduce the size even further. Huge transactions are created by the SPHINCS$^+$ algorithm.

Block size. The structure of a block is shown in table IV (right). Figure 7 shows the block size that depends on parameters and the number of transactions. The transaction structure and size from the previous figure 6 was used to calculate the block size holding different numbers of transactions.

As mentioned in the previous paragraph SPHINCS$^+$ creates huge transactions which leads also to huge blocks. The results in 7 illustrate again that XMSS has the smallest size followed by Falcon and Dilithium. Table VIII compares the sizes of the public key, private key and signature. Moreover, it displays the sizes of a wallet, transaction and block. When compared to the classical cryptography algorithm ECDSA, the private and public key sizes from SPHINCS$^+$ and XMSS achieve the smallest sizes. However, small signatures are achieved with Falcon and XMSS. Small wallets are possible with SPHINCS$^+$.

[2]https://github.com/open-quantum-safe/liboqs-python
[3]https://github.com/XMSS/xmss-reference
[4]https://remix.ethereum.org/

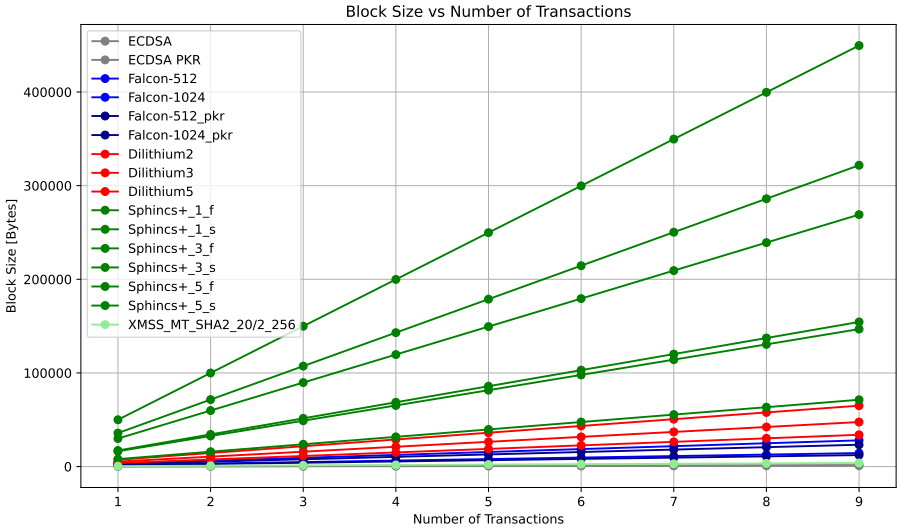


Fig. 7: Block sizes for different amounts of transactions with a value equivalent to 1 Ethereum

TABLE VIII: Size evaluation with different PQC Algorithms. All sizes are in bytes.

Algorithm	Private-key	Public-key	Signature	Wallet	Transaction	Block
ECDSA SECP256K1	32	65	71	97	171	238
ECDSA SECP256K1 PKR	32	65	71	97	106	238
Falcon-512	1281	897	666	2178	1598	3234
Falcon-1024	2305	1793	1280	4098	3108	6254
Falcon-512 PKR	1281	897	666	2178	1358	2754
Falcon-1024 PKR	2305	1793	1280	4098	2586	5210
Dilithium2	2528	1312	2420	3840	3767	7572
Dilithium3	4000	1952	3293	5952	5280	10598
Dilithium5	4864	2592	4595	7456	7222	14482
SPHINCS$^+$-SHA-256-128f simple	64	32	17088	96	17155	15884
SPHINCS$^+$-SHA-256-128s simple	64	32	7856	96	7923	15884
SPHINCS$^+$-SHA-256-192f simple	96	48	35664	144	35747	32652
SPHINCS$^+$-SHA-256-192s simple	96	48	16224	144	16307	32652
SPHINCS$^+$-SHA-256-256f simple	128	64	49856	192	49955	59820
SPHINCS$^+$-SHA-256-256s simple	128	64	29792	192	29891	59820
XMSS-MT-SHA2_20/2_256	213	35	352	248	422	123

The smallest transactions and blocks are achieved with Falcon and XMSS. Falcon with public-key recovery mode can reduce the transaction and block size.

C. Speed of Cryptographic Algorithms

The speed test includes 15 different configurations of algorithms: the baseline classical signature algorithm ECDSA, three Dilithium configurations (Dilithium2/3 and 5), four Falcon implementations (Falcon-521/1024 and Falcon-512/1024 PKR), six SPHINCS$^+$ implementations (SPHINCS$^+$-SHA256-128/193/256 in modes s and f and the XMSS algorithm (XMS_MT_SHA2_20/2_256). ECDSA was chosen with elliptic curve SECP256K1. A modified version of the Falcon reference code was the basis for Falcon in public-key recovery mode. We measured the time required for key generation, signing and verification of the twelve cryptographic configurations and reported them in table IX.

The performance was evaluated for different PQC NIST algorithms and security levels, from NIST Level 1 (lowest) to NIST Level 5 (highest). The results are obtained after executing each algorithm 1000 times. This number was used in previous works to evaluate the performance of the PQC in electronic devices [42]. SPHINCS$^+$-SHA-256 seems to be implemented incorrectly for key generation in level 5 because the speed increases between security levels 3 and 5. This seems to be a bug in the liboqs library that will be reported. In table IX the incorrect values are marked in red.

XMSS-reference library can not be built on ARM machines since some registers are defined differently. Using PKR on the embedded device reduces the computational speed noticeably. Results show that PQC algorithms have

TABLE IX: Speed evaluation of PQC algorithms on i7-9750H CPU and Raspberry Pi 3b.

Algorithm	i7-9750H CPU			ARM Cortex-A72		
	Keygen/sec	Signs/sec	Verify/sec	Keygen/sec	Signs/sec	Verify/sec
ECDSA SECP256K1	1123,6	1204,8	1388,89	158,5	181,6	224,6
Dilithium2	33898,3	13698,6	32258,1	418,4	138,6	413,6
Dilithium3	21276,6	9090,9	23310,0	245,9	113,0	246,1
Dilithium5	13698,6	7692,3	14705,9	147,7	76,8	144,7
Falcon-512	169,6	4524,9	21929,8	6,7	32,8	1398,7
Falcon-1024	59,5	2336,5	11764,7	2,9	14,8	746,0
Falcon-512 PKR	171,2	4522,1	8576,3	29,5	3,0	5,3
Falcon-1024 PKR	60,4	2332,3	6410,3	12,2	1.4	2.4
SPHINCS$^+$-SHA-256-128f simple	3008,7	156,9	1701,6	67,8	3,0	49,5
SPHINCS$^+$-SHA-256-128s simple	38,0	5,0	1083,4	1,1	0,1	152,2
SPHINCS$^+$-SHA-256-192f simple	1975,3	92,6	1177,9	46,3	1,8	34,2
SPHINCS$^+$-SHA-256-192s simple	26,9	2,7	704,2	0,7	<0,1	101,1
SPHINCS$^+$-SHA-256-256f simple	924,3	43,0	1006,7	18,1	0,8	33,8
SPHINCS$^+$-SHA-256-256s simple	41,9	3,4	495,1	1,2	<0,1	68,5
XMSS-MT-SHA2_20/2_256	0.4	193.7	697.8	-	-	-

different benefits and drawbacks compared to classical ECDSA. Overall PQC algorithms are fast and especially Dilithium is faster than any other parameter set. There is a clear difference between the performance on the i7 CPU and the ARM Cortex-A53 CPU. The limited computing power on the ARM processor lead to very slow cryptographic operations. Using Falcon in public-key-recovery (PKR) mode does not have any influence on the speed of key generation and signature creation. The signature process is only changed to extract the value s_1 from a function already integrated in Falcon normal mode. However, the verification time increases since the public key must be calculated before the start of the verification process.

D. Blockchain performance

High TPS values are the design goal of many blockchains [49] [68] [69].

$$TPS = \frac{\tau}{t} \tag{7}$$

Where τ stands for a typical transaction on the blockchain and t is the average time in seconds that passes until a new transaction can be placed.

To measure the TPS value and compare the results of the different algorithms we created two blockchain setups. One setup utilizes normal computational performance and one that uses embedded system devices (Rapsberry Pi 3b). Both setups act in a private blockchain network. The following graphs show the results of the measurements we perform on both setups.

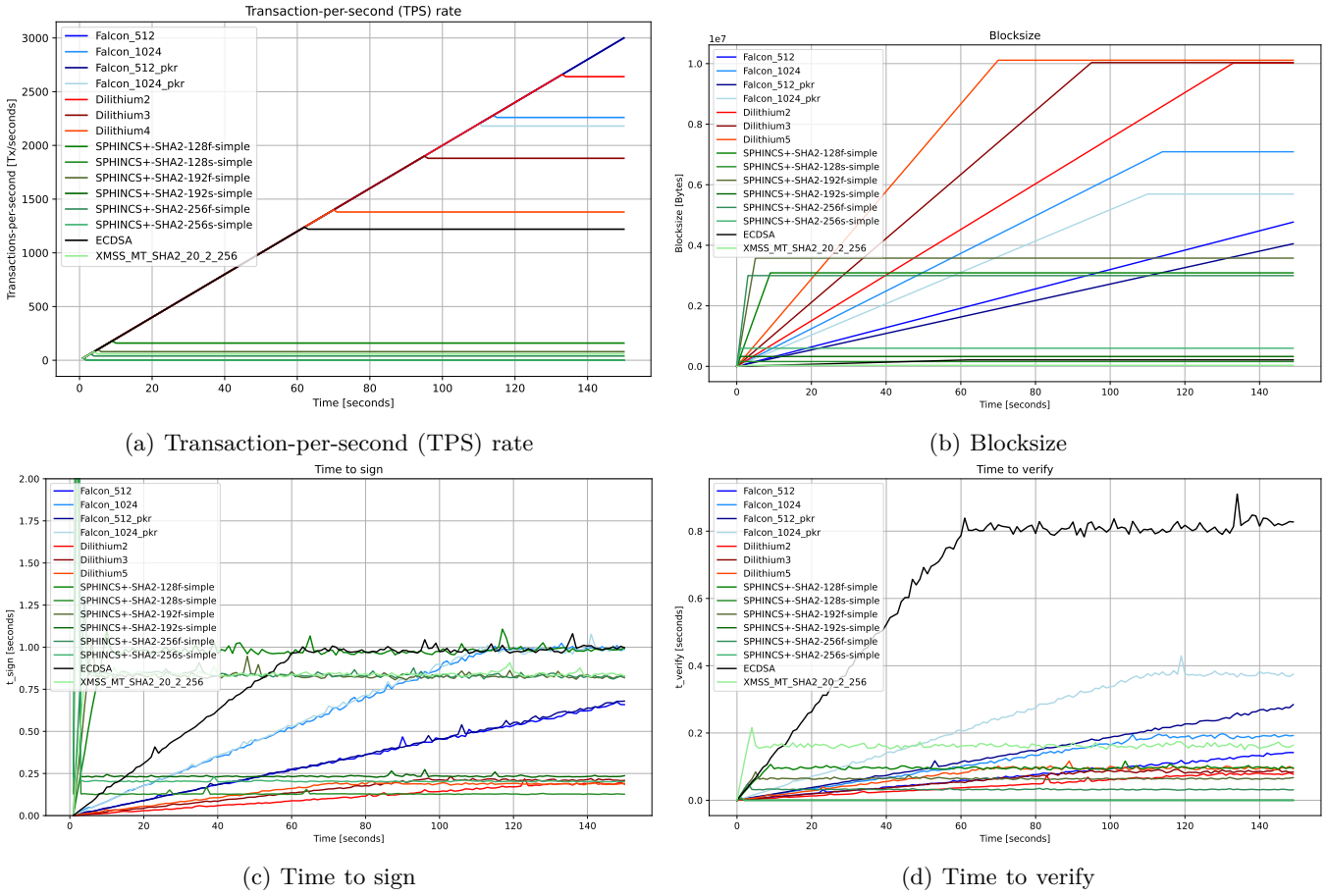
We measure the maximum TPS value, the block size during each mining epoch, the time to sign all transactions and the time to verify all transactions. To reach the maximum TPS value we run the blockchain at block creation time of 1 second and measure transactions-per-second. Adjusting the difficulty value of the mining process automatically keeps the block creation time stable so that every second a new block is mined and attached to the blockchain.

On the first setup (Figure 8), we increased the number of transactions that need to be processed by the blockchain by 10 each second. This leads to a linear increase of transactions in the blocks: 10,20,30,40,...,n transactions per block.

Figure 8 (a) presents the TPS rate for each selected signature algorithm. Without limitations, the TPS rate would increase linearly over time. However, the algorithms are bound at specific TPS rates and follow a horizontal trend. The limitation factor can be seen in the diagrams (b),(c) and (d): In 8 (b) the measured blocksize is displayed for the current block at a specific time. Since the maximum blocksize is limited to 10MB and some algorithms reach this boundary, the number of TPS can only be as high as the number of transactions at the blocksize limit. Figure 8 (c) and (d) present the duration for the signing and verifying process. If the signing or verification process exceeds one second the cryptographic operation takes too long to hold the requirement to attach every second a new block.

We perform the same measurements on the embedded system setup (Figure 9). However, due to the lack of computing performance on the devices we increase the number of transactions only by 1 (1,2,3,..., n transactions per block) each second.

From these two measurements (corresponding to figure 8 and figure 9), we can obtain the maximum TPS rate for each set of algorithms. In table X we list for both setups the maximum TPS value which can be achieved with the selected algorithm. In addition, for this TPS value we present the size of the block, the time for signing and the time for the verification process. Since TPS are not infinite high we can see the limitations of each parameter set(values marked in red). We excluded SPHINCS$^+$ as a useful candidate on the embedded system



(a) Transaction-per-second (TPS) rate

(b) Blocksize

(c) Time to sign

(d) Time to verify

Fig. 8: Performance results for the blockchain setup with computer-based network. Each second the transactions added to the queue are increased by 10 transactions.

experiments since the measurements on the computer setup were not sufficiently good enough. We only focused on the promising algorithms Falcon and Dilithium in the experiments with the embedded systems setup.

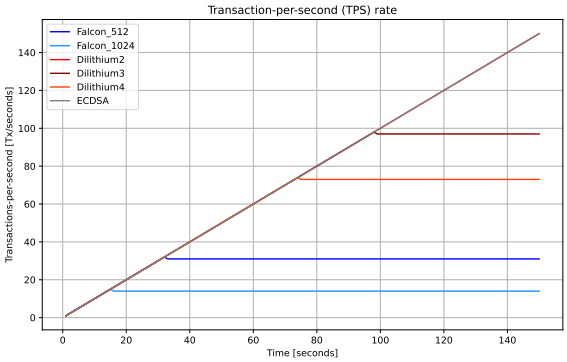
If a quantum-secure blockchain runs on a computer-based system the best choice is the Falcon algorithm. However, when using embedded devices Dilithium archives a higher TPS rate. With the "s" version of SPHINCS$^+$ we are not able to run the blockchain because signing operations take longer than the block creation time (time for signing $>$ 1 second). In this case is mining faster than transactions can be signed and prepared for mining.

Dilithium is limited by the huge signature sizes. The blocks are too big and reach the block size limit of 10 MB. On the embedded setup for Dilithium the time to sign is the bottleneck. If more than 150 (Level 1), 97 (Level 3) and 73 (Level 5) blocks are added to a block the signing process takes too long. However, the 1-second block creation limit is also the limitation for Falcon parameter sets. With Falcon, we can only attach up to 31 on the computer setup and 14 on the embedded setup transactions in a block. Falcon in PKR mode has no drawbacks in performance compared to normal Falcon, but the block sizes are smaller. On the Raspberry Pi, the time for the additional recovery operation during the verification process limits the algorithm's capabilities. The transactions-per-second with XMSS are also limited by its long signing times.

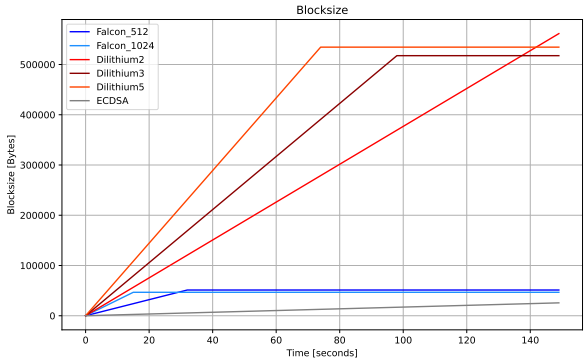
E. PQC in Smart Contracts

The quantum-secure blockchain can be used with smart contracts. These smart contracts can be used to create decentralized applications. Authentication of users is one of the applications for smart contract platforms. It is typically used to verify someone's identity before giving access to resources that only a specific person is allowed to use. While this can be implemented by sharing secret keys between the user and the authenticating party, or by relying on users bio-metrics, a blockchain can be used. Figure 10 shows the use of the blockchain for a sample setup of user authentication. It relies on the Distributed Single Sign-On (DSSO) concept [70]. It comprises two steps: a registration procedure and ii) a login procedure.

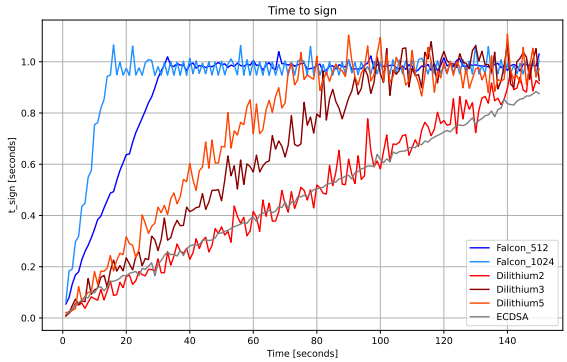
In the registration procedure, the user's wallet account is used to deploy a smart contract that stores the public key. If deployed, everyone can call the smart contract and



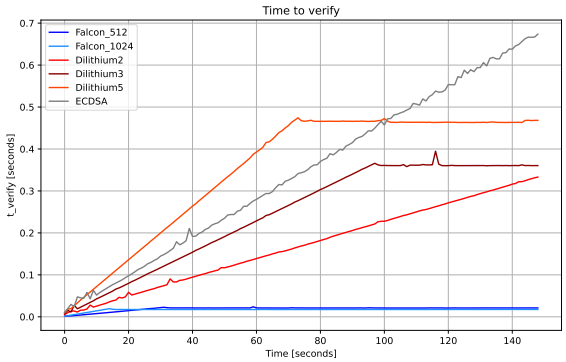
(a) Transaction-per-second (TPS) rate



(b) Blocksize



(c) Time to sign



(d) Time to verify

Fig. 9: Performance results for the blockchain setup with embedded systems based (Raspberry Pi 3b) network. Each second the transactions added to the queue are increased by 1 transaction.

TABLE X: Performance results of PQC algorithms on i7-9750H CPU and Raspberry Pi 3b.

Parameter Set	i7-9750H CPU				ARM Cortex-A72			
	TPS	Block [Byte]	t_{sign} [sec]	t_{verify} [sec]	TPS	Block [Byte]	t_{sign} [sec]	t_{verify} [sec]
ECDSA SECP256K1	1220	213316	0.997	0.827	150	25664	0.875	0.673
Falcon-512	>3000	4765056	0.658	0.141	31	51204	0.982	0.021
Falcon-1024	2260	7088556	0.989	0.192	14	46671	0.952	0.017
Dilithium2	2640	10022916	0.186	0.077	150	561468	0.942	0.336
Dilithium3	1880	10033936	0.209	0.083	97	517574	0.926	0.360
Dilithium5	1380	10112236	0.190	0.093	73	534538	0.993	0.468
SPHINCS$^+$-SHA-256-128f	160	3088116	0.995	0.096	-	-	-	-
SPHINCS$^+$-SHA-256-128s	0	158516	0.127	<0.001	-	-	-	-
SPHINCS$^+$-SHA-256-192f	80	3574836	0.818	0.067	-	-	-	-
SPHINCS$^+$-SHA-256-192s	0	326196	0.236	<0.001	-	-	-	-
SPHINCS$^+$-SHA-256-256f	40	2997396	0.827	0.031	-	-	-	-
SPHINCS$^+$-SHA-256-256s	0	597876	0.202	<0.001	-	-	-	-
XMSS-MT-SHA2_20/2_256	60	33876	0.827	0.163	-	-	-	-

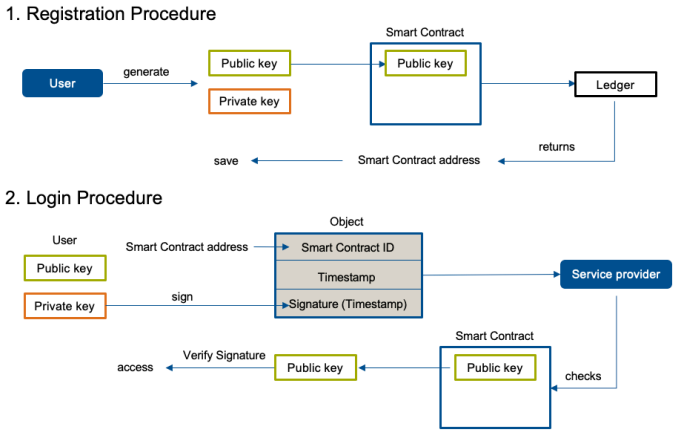


Fig. 10: The concept for authentication with smart contracts

get the user's public key. If the user wants to log in, he uses the contract ID and timestamp to generate a signature by using his private key. The service provider can verify if the signature is correct and compare calling the smart contract to get the public key. If both match, he can give access to the user. For the quantum-secure blockchain, the sizes of the public keys and signatures increase. This leads to more storage in block transactions and the smart contract size and can influence the amount of gas needed to deploy and call the smart contract.

Table XI shows the size of the code snipes. The public key in the smart contract is PEM encoded, adding information at the key's beginning and end. Besides classical cryptography (ECDSA) the shortest size is achieved using SPHINCS$^+$ and XMSS. This relates to the short public key sizes of SPHINCS$^+$. However, even with large key sizes of Dilithium or Falcon, it is possible to run the code on the quantum-secure blockchain, but the disadvantage is that the deployment of the smart contract is more expensive due to the sizes which are up to 6 times larger.

TABLE XI: Size comparison for a Smart Contract for authentication. All sizes in bytes.

Algorithm	Size of Smart Contract [bytes]
ECDSA SECP256K1	150
Dilithium2	1429
Dilithium3	2069
Dilithium5	2709
Falcon-512	1014
Falcon-1024	1910
Falcon-512 PKR	1014
Falcon-1024 PKR	1910
SPHINCS$^+$-128s	149
SPHINCS$^+$-192s	165
SPHINCS$^+$-256s	181
XMSS-MT-SHA2_20/2_256	330

VIII. Discussion

In this study, our research aimed to uncover the primary challenges associated with developing a quantum-secure blockchain using embedded systems and Post-Quantum Cryptography (PQC). Specifically, we determined which PQC algorithms are most suitable for this application and identify the limitations and constraints of each algorithm. Our investigation also involved assessing the necessary adaptations when transitioning from classical cryptography to PQC. To achieve these objectives, we conducted an in-depth analysis and devised a blockchain design to evaluate the behavior, performance, and sizes of a quantum-secure blockchain implemented on embedded systems. Additionally, we examined the impact of PQC on smart contracts within a post-quantum blockchain framework. Notably, we integrated the public-key recovery feature with Falcon to reduce transaction sizes, further enhancing the efficiency of the blockchain system. Overall, our findings shed light on the complexities and considerations involved in implementing a quantum-secure blockchain on embedded systems, offering valuable insights for future research and development in this field.

Our results show that XMSS has the smallest transaction and block sizes but being a stateful algorithm. Falcon is the best stateless algorithm and has even smaller transaction and block sizes with PKR mode. For a blockchain, the verification time is more important than the time for signing Results show that user only signs a transaction once, while the signature is verified multiple times by other network nodes. All measurements of the PQC algorithms are aligned with the results reported in the Open Quantum Safe project [22]. Comparing speed results in table IX to the values of the OQS project, we have a slightly better performance for all algorithms with our computer setup. PQC algorithms are fast and especially Dilithium is faster than any other parameter set. The XMSS key generation is very slow. SPHINCS$^+$ is not useful on embedded systems because the algorithm performs very slow for signing operations. The PKR mode of Falcon reduces speed noticeably, especially on embedded systems. Dilithium is fast and on the Raspberry Pi 3b it achieves comparable results to classical ECDSA.

With high performance blockchain setups best results are achieved by using the PQC algorithm Falcon. However, when using embedded devices, Dilithium archives a higher TPS rate. This is due to the slow signing performance of the Falcon algorithm on the ARM CPU. During our experiments we found out that Dilithium is limited by the huge signature sizes.

Currently, blockchains employ ECDSA in public key recovery mode, allowing for the derivation of the public key from the signature alone, thus optimizing memory usage. Falcon offers a similar public key recovery feature, presenting a potential alternative. However, further research is warranted to address security concerns, as this

mode is not part of the NIST standardization process.

While XMSS presents an appealing signature option due to its configurable nature, its implementation complexity poses challenges. Managing private key utilization is crucial to prevent scheme insecurity, as XMSS implementations typically allow a maximum of 1024 signatures [30]. Integrating PQC into blockchain will significantly increase block size and demand more resources, such as ledger capacity and larger smart contracts. This results in fewer transactions per block and a slowdown in transaction processing rates. Despite XMSS's smaller key sizes, its signature sizes and implementation complexity hinder widespread adoption in blockchain.

We demonstrate the authentication of users through smart contracts on a quantum-secure blockchain, successfully executing verification operations within them. Due to the large code size, pre-deployment of program code (including cryptographic operations) in Virtual Machines (VMs) is necessary. Expanding the block size limit may alleviate issues caused by large transaction sizes. For instance, Bitcoin Cash can increase its block size from 1 MB to 8 MB or 32 MB. However, arbitrary expansion is impractical as it escalates bandwidth and storage requirements, conflicting with decentralized computing goals and slowing down transactions per second (TPS).

The transition of classical blockchains to future quantum-secure variants must address data legacy. Existing coins or tokens stored on old addresses, protected by outdated key pairs, require updating to new quantum-resistant key pairs. Implementing PQC capabilities and introducing new features to existing blockchains is complex and requires hard forks, necessitating adoption of new software versions by the network. While exploring PQC with current blockchain solutions is challenging, it's crucial to develop strategies and mechanisms for a smoother transition to quantum-secure mechanisms. Core developer teams maintaining open-source networks should commit to this task.

IX. Summary and Conclusions

A quantum-secure blockchain is imperative for ensuring the enduring security of applications reliant on blockchain technology. In this work, we introduce a quantum-secure blockchain framework tailored for exploring various Post-Quantum Cryptography (PQC) algorithms within embedded systems, including the utilization of PQC algorithms in public key recovery mode.

Our findings reveal that integrating PQC algorithms into blockchain poses challenges due to the significantly larger key sizes: PQC public key sizes incur overhead ranging from 50% to 3988% compared to classical ECDSA key sizes (65 bytes), while signature sizes experience overhead ranging from 972% to 41961% of the ECDSA signature size. Nonetheless, we demonstrate that the public key recovery mode may facilitate the integration of PQC into blockchain. Specifically, Falcon in public key recovery mode reduces public key and signature sizes by 15% for low security levels (level I) and by 17% for high security levels (level V). Additionally, we evaluate the performance of PQC algorithms for key generation, signing, and verification operations on both computers and embedded devices. Our results indicate that Dilithium surpasses other PQC algorithms (Falcon and SPHINCS$^+$) on embedded devices, even outperforming classical ECDSA for signing and verification operations.

In summary, our investigation into blockchain performance on embedded systems identifies Falcon-512 as the most suitable algorithm, offering robust security and computational efficiency, with XMSS serving as a viable stateful alternative. Dilithium exhibits superior transactions-per-second (TPS) rates on embedded devices compared to Falcon due to Falcon's slower signing performance on ARM CPUs. While Dilithium's computational efficiency is notable, its large signature sizes present challenges. Moreover, we integrate smart contract functionality into the quantum-secure blockchain, demonstrating the impact of Post-Quantum Cryptography (PQC) on authentication within smart contracts and highlighting the feasibility of employing quantum-secure blockchain in embedded system environments.

References

[1] P. Tasatanattakool and C. Techapanupreeda, "Blockchain: Challenges and applications," in 2018 International Conference on Information Networking (ICOIN), IEEE. IEEE, Jan. 2018, pp. 473–475.

[2] A. A. Monrat, O. Schelén, and K. Andersson, "A survey of blockchain from the perspectives of applications, challenges, and opportunities," IEEE Access, vol. 7, pp. 117 134–117 151, 2019.

[3] F. Rizal Batubara, J. Ubacht, and M. Janssen, "Unraveling transparency and accountability in blockchain," in Proceedings of the 20th annual international conference on digital government research, ser. dg.o 2019. ACM, Jun. 2019, pp. 204–213.

[4] J. Abou Jaoude and R. George Saade, "Blockchain applications – usage in different domains," IEEE Access, vol. 7, pp. 45 360–45 381, 2019.

[5] D. B. Rawat, V. Chaudhary, and R. Doku, "Blockchain: Emerging applications and use cases," arXiv preprint arXiv:1904.12247, 2019.

[6] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang, "Consortium blockchain for secure energy trading in industrial internet of things," IEEE Transactions on Industrial Informatics, pp. 1–1, 2017.

[7] P. B. Veena Pureswaran, "Device democracy: Saving the future of the internet of things," Feb. 2016, [Accessed 20-07-2024]. [Online]. Available: https://www.ibm.com/thought-leadership/institute-business-value/en-us/report/device-democracy

[8] R. Avila, "Blockchain in embedded systems and devices: Security, reliability and efficiency," Embedded Development Talk, Dec. 2021, [Accessed 20-07-2024]. [Online]. Available: https://www.qt.io/embedded-development-talk/how-blockchain-improves-iot-embedded-systems-qt

[9] S. Falcone, J. Zhang, A. Cameron, and A. Abdel-Rahman, "Blockchain design for an embedded system," Ledger, Apr. 2019.

[10] L. Wang, X. Shen, J. Li, J. Shao, and Y. Yang, "Cryptographic primitives in blockchains," Journal of Network and Computer Applications, vol. 127, pp. 43–58, Feb. 2019.

[11] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on. IEEE, 1994, pp. 124–134.

[12] National Institute of Standards and Technology (NIST), "Announcing request for nominations for public-key post-quantum cryptographic algorithms," 2016, Available at: https://csrc.nist.gov/news/2016/public-key-post-quantum-cryptographic-algorithms.

[13] National Institute of Standards and Technology, "Status report on the first round of the NIST post-quantum cryptography standardization process," 2019, https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8240.pdf.

[14] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber algorithm specifications and supporting documentation(version 3.02)," Aug. 2021, [Accessed 07-20-2024]. [Online]. Available: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf

[15] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium algorithm specifications and supporting documentation," 2019, [Accessed 20-07-2024]. [Online]. Available: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf

[16] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-fourier lattice-based compact signatures over ntru," 2020, [Accessed 20-07-2024]. [Online]. Available: https://falcon-sign.info/falcon.pdf

[17] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The sphincs+ signature framework," in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2129–2146. [Online]. Available: https://doi.org/10.1145/3319535.3363229

[18] National Institute of Standards and Technology, "Status report on the third round of the nist post-quantum cryptography standardization process," 2022, Available: https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413.pdf.

[19] Internet Research Task Force (IRTF), "Xmss: extended merkle signature scheme," 2018, Available at: https://datatracker.ietf.org/doc/html/rfc8391.

[20] ——, "Leighton-micali hash-based signatures," 2018, Available at: https://datatracker.ietf.org/doc/html/rfc8554.

[21] National Institute of Standards and Technology (NIST), "Recommendation for stateful hash-based signature schemes," 2020, Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-208.pdf.

[22] M. M. Douglas Stebila, "Post-quantum key exchange for the internet and the open quantum safe project," Selected Areas in Cryptography (SAC) 2016, LNCS, vol. 10532, pp. 1–24, Springer, Oct. 2017. [Online]. Available: https://openquantumsafe.org

[23] C. Hicks, "Different types of cryptocurrencies," Frobes, 2023. [Online]. Available: https://www.forbes.com/advisor/investing/cryptocurrency/different-types-of-cryptocurrencies/

[24] B. Chez. [Online]. Available: https://coinmarketcap.com/

[25] D. C. Berghoff, D. U. Gebhardt, D. M. Lochter, and D. S. Maßberg, "Blockchain sicher gestalten," Mar. 2019, accessed 25-02-2024]. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Krypto/Blockchain_Analyse.html

[26] A. K. Fedorov, E. O. Kiktenko, and A. I. Lvovsky, "Quantum computers put blockchain security at risk," Nature, vol. 563, no. 7732, pp. 465–467, Nov. 2018.

[27] W. Yin, Q. Wen, W. Li, H. Zhang, and Z. Jin, "An anti-quantum transaction authentication approach in blockchain," IEEE Access, vol. 6, pp. 5393–5401, 2018.

[28] D. Aggarwal, G. K. Brennen, T. Lee, M. Santha, and M. Tomamichel, "Quantum attacks on bitcoin, and how to protect against them," Ledger, [S.l.], v. 3, oct. 2018, Oct. 2017.

[29] N. Verma, S. Kumari, and P. Jain, "Post quantum digital signature change in IOTA to reduce latency in internet of vehicles (IoV) environments," in 2022 International Conference on IoT and Blockchain Technology (ICIBT). IEEE, 2022.

[30] The QRL, "Quantum resistant ledger (QRL)," 2016, [Accessed 20-07-2024]. [Online]. Available: https://github.com/theQRL/Whitepaper/blob/master/QRL_whitepaper.pdf

[31] N. Anhao, "Bitcoin post-quantum," BitcoinPQ, Tech. Rep., 2018, [Accessed 20-07-2024]. [Online]. Available: https://bitcoinpq.org/download/bitcoinpq-whitepaper-english.pdf

[32] R3, "R3 publishes a new post-quantum signature algorithm tailored to blockchains," Aug. 2018. [Online]. Available: https://corda.net/blog/r3-publishes-a-new-post-quantum-signature-algorithm-tailored-to-blockchains/

[33] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: An introduction," Aug. 2016, [Accessed 20-07-2024]. [Online]. Available: https://docs.r3.com/en/pdf/corda-introductory-whitepaper.pdf

[34] T. M. Fernandez-Carames and P. Fraga-Lamas, "Towards post-quantum blockchain: A review on blockchain cryptography resistant to quantum computing attacks," IEEE Access, vol. 8, pp. 21 091–21 116, 2020.

[35] A.-T. Ciulei, M.-C. Cret, and E. Simion, "Preparation for post-quantum era: a survey about blockchain schemes from a post-quantum perspective," 2022.

[36] W. van der Linde, "Post-quantum blockchain using one-time signature chains," Master's thesis, Radboud University, 2018.

[37] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," in Proceedings of the twenty-first annual ACM symposium on Theory of computing - STOC '89. ACM Press, 1989.

[38] J. Chen, W. Gan, M. Hu, and C.-M. Chen, "On the construction of a post-quantum blockchain for smart city," Journal of Information Security and Applications, vol. 58, p. 102780, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212621000284

[39] W. Beullens, "Breaking rainbow takes a weekend on a laptop," Cryptology ePrint Archive, Paper 2022/214, 2022, [Accessed 20-07-2024]. [Online]. Available: https://eprint.iacr.org/2022/214

[40] W. BOSMA, J. CANNON, and C. PLAYOUST, "The magma algebra system i: The user language," Journal of Symbolic Computation, vol. 24, no. 3-4, pp. 235–265, sep 1997.

[41] R. Saha, G. Kumar, T. Devgun, W. Buchanan, R. Thomas, M. Alazab, T.-H. Kim, and J. Rodrigues, "A blockchain framework in post-quantum decentralization," IEEE Transactions on Services Computing, pp. 1–1, 2021.

[42] P. Thanalakshmi, A. Rishikhesh, J. Marion Marceline, G. P. Joshi, and W. Cho, "A quantum-resistant blockchain system: A comparative analysis," Mathematics, vol. 11, no. 18, p. 3947, Sep. 2023.

[43] C.-Y. Li, X.-B. Chen, Y.-L. Chen, Y.-Y. Hou, and J. Li, "A new lattice-based signature scheme in post-quantum blockchain network," IEEE Access, vol. 7, pp. 2026–2033, 2019.

[44] Y.-L. Gao, X.-B. Chen, Y.-L. Chen, Y. Sun, X.-X. Niu, and Y.-X. Yang, "A secure cryptocurrency scheme based on post-quantum blockchain," IEEE Access, vol. 6, pp. 27 205–27 213, 2018.

[45] D. Rajan and M. Visser, "Quantum blockchain using entanglement in time," Quantum Reports, vol. 1, no. 1, pp. 3–11, Apr. 2019.

[46] E. O. Kiktenko, N. O. Pozhar, M. N. Anufriev, A. S. Trushechkin, R. R. Yunusov, Y. V. Kurochkin, A. I. Lvovsky, and A. K. Fedorov, "Quantum-secured blockchain," Quantum Science and Technology, vol. 3, no. 3, p. 035004, May 2018.

[47] X. Sun, M. Sopek, Q. Wang, and P. Kulicki, "Towards quantum-secured permissioned blockchain: Signature, consensus, and logic," Entropy, vol. 21, no. 9, p. 887, Sep. 2019.

[48] H. Gharavi, J. Granjal, and E. Monteiro, "Post-quantum blockchain security for the internet of things: Survey and research directions," IEEE Communications Surveys & Tutorials, pp. 1–1, 2024.

[49] D. Gavin Wood, "Ethereum: A secure decentralised generalised transaction ledger berlin version," 2013, [Accessed 20-07-2024]. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[50] Ethereum.org, "The merge," 2022, [Accessed 20-07-2024]. [Online]. Available: https://ethereum.org/de/upgrades/merge/

[51] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[52] G. Bertoni, J. Daemen, M. Peeters, and G. van Assche, "The keccak sha-3 submission," Jan. 2011, [Accessed 20-07-2024]. [Online]. Available: https://keccak.team/files/Keccak-submission-3.pdf
[53] M. J. Dworkin, "SHA-3 standard: Permutation-based hash and extendable-output functions," jul 2015.
[54] L. K. Grover, "A fast quantum mechanical algorithm for database search," in Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866
[55] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124–134.
[56] M. Roetteler, M. Naehrig, K. M. Svore, and K. Lauter, "Quantum resource estimates for computing elliptic curve discrete logarithms," 2017.
[57] A. Gnip, "How many bitcoins are vulnerable to a hypothetical quantum attack?" medium.com, 2018, [Accessed 20-07-2024]. [Online]. Available: https://medium.com/@sashagnip/how-many-bitcoins-are-vulnerable-to-a-hypothetical-quantum-attack-3e59e4172e8
[58] A. H. Lone and R. Naaz, "Demystifying cryptography behind blockchains and a vision for post-quantum blockchains," in 2020 IEEE International Conference for Innovation in Technology (INOCON). IEEE, Nov. 2020.
[59] S. Chauhan, V. P. Ojha, S. Yarahmadian, and D. Carvalho, "Towards building quantum resistant blockchain," in 2023 International Conference on Electrical, Computer and Energy Technologies (ICECET). IEEE, Nov. 2023.
[60] D. R. L. Brown, "Sec 1: Elliptic curve cryptography," May 2009.
[61] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium-Round 3," 2020, [Accessed 20-07-2024]. [Online]. Available: https://pq-crystals.org/dilithium/data/dilithium-specification-round3.pdf
[62] J.-P. Aumasson, D. Bernstein, W. Beullens, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, and B. Westerbaan, "Sphincs+," Oct. 2020.
[63] J. Buchmann, E. Dahmen, and A. Hülsing, "XMSS - a practical forward secure signature scheme based on minimal security assumptions," in Post-Quantum Cryptography. Springer Berlin Heidelberg, 2011, pp. 117–129.
[64] "Bitcoin - Open source P2P money — bitcoin.org," https://bitcoin.org, [Accessed 25-02-2024].
[65] M. J. Kannwischer, P. Schwabe, D. Stebila, and T. Wiggers, "Improving software quality in cryptography standardization projects," in 2022 IEEE European Symposium on Security and Privacy Workshops. IEEE, jun 2022.
[66] P. Kampanakis and S. R. Fluhrer, "Lms vs xmss : Comparion of two hash-based signature standards," 2017. [Online]. Available: https://eprint.iacr.org/2017/349.pdf
[67] T. Prest, "falcon.py," https://github.com/tprest/falcon.py, 2020.
[68] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0.8.13."
[69] B. Chase and E. MacBrough, "Analysis of the XRP ledger consensus protocol," CoRR, vol. abs/1802.07242, 2018. [Online]. Available: http://arxiv.org/abs/1802.07242
[70] O. J. Eric Borgsten, "Authentication using smartcontracts in a blockchain," Master's thesis, University of Gothenburg, 2018.