Design of a Quantum-Resistant Communication Protocol Using Blockchain With Post-Quantum Security

Vignesh.M - 3122225002154
Vijay.R.S - 3122225002156

Problem Statement

Current blockchain-based communication platforms rely on cryptographic algorithms (e.g., RSA, ECDSA) that are susceptible to quantum attacks. With the rise of quantum computing, these platforms face the risk of message theft, forgeries, and loss of communication privacy. There is a lack of real-world, scalable, user-friendly, public blockchain protocols designed from the ground up to ensure secure, end-to-end encrypted messaging using natively post-quantum cryptographic primitives. This creates an urgent need for solutions that can preserve privacy and integrity of public communications in the quantum era.

Motivation

- Blockchain's promise of permanent security is on a collision course with the reality of quantum computing. The cryptographic signatures (ECDSA) that protect major cryptocurrencies are vulnerable to quantum attacks like Shor's algorithm, placing a significant portion of the digital economy at risk—including an estimated 25% of all Bitcoin in circulation. This threat is immediate, driven by "Harvest Now, Decrypt Later" strategies where adversaries store today's encrypted data for future decryption.
- In response, governments are issuing urgent mandates: the U.S. has set a 2035 deadline for federal agencies to migrate to Post-Quantum Cryptography (PQC), and the E.U. has set an even more aggressive 2030 target for its critical infrastructure. Despite this, enterprise adoption is dangerously low at just 5%.

Literature Survey

- The rise of quantum computing poses a direct threat to the security of modern blockchain systems, which rely on classical cryptography like the Elliptic Curve Digital Signature Algorithm (ECDSA) that will be breakable by quantum machines. This report summarizes the state of research on integrating Post-Quantum Cryptography (PQC) to secure these networks.
- The leading PQC solutions, particularly NIST-standardized lattice-based signatures like CRYSTALS-Dilithium and FALCON, offer a viable path forward. However, this transition involves significant trade-offs: PQC algorithms introduce larger keys and signatures, which drastically reduce transaction throughput, increase network latency, and demand more storage. The central challenge is not just technical but also one of governance, requiring a coordinated, network-wide migration—a process known as achieving crypto-agility—in a complex, decentralized ecosystem. [1]

Literature Survey

1. The Quantum Threat to Blockchain

Blockchain's core value is its permanent, immutable ledger, secured by cryptography that is assumed to be unbreakable for the long term. Quantum computing invalidates this assumption.

Vulnerability of Digital Signatures: Shor's quantum algorithm can efficiently solve the mathematical problems that underpin the security of ECDSA, the signature scheme used by Bitcoin and Ethereum. A quantum attacker could derive a user's private key from their public key, which is revealed during a transaction, and steal their funds.

Weakening of Hashing: Grover's algorithm offers a quadratic speed-up in breaking cryptographic hash functions like SHA-256, which are fundamental to Proof-of-Work (PoW) mining.[1] This would disrupt the security and economic balance of PoW-based networks.

Literature Survey

"Harvest Now, Decrypt Later": The threat is immediate. Adversaries can record today's encrypted blockchain data and decrypt it once a powerful quantum computer is available. Because blockchain data is public and permanent, it is a prime target for this attack strategy.

2. The Post-Quantum Cryptographic Arsenal

In response to the quantum threat, the cryptographic community, guided by the NIST standardization process, has developed several families of PQC algorithms. For blockchain, the most relevant are digital signature schemes.

Lattice-Based Cryptography (CRYSTALS-Dilithium, FALCON): This is the most promising family, offering a strong balance of security, speed, and relatively compact size. Their security is based on the hardness of problems in high-dimensional lattices. FALCON is particularly noteworthy for its exceptionally small signatures and fast verification, making it well-suited for the storage and computation constraints of blockchain.

Literature Survey

3. Performance Analysis and Trade-Offs

Quantitative benchmarking reveals the stark performance costs of migrating to PQC.

Size Overhead: PQC signatures are substantially larger than ECDSA's ~70 bytes. FALCON signatures are around 10 times larger, Dilithium signatures are ~35 times larger, and SPHINCS+ signatures can be over 100 times larger.

Throughput Reduction: This size increase directly slashes transaction throughput. Studies modeling the impact on a Bitcoin-like chain estimate that switching to Dilithium could reduce the number of transactions per block by over 90%.

The "Sign-Once, Verify-Many" Paradigm: In blockchain, a transaction is signed once but verified by thousands of nodes. This makes verification speed and signature size the most critical metrics. This is why FALCON, despite slower signing, is a leading candidate due to its extremely fast verification and compact signatures, which reduce the overall burden on the network.

Methodology

1) **System Design and Implementation**

(A) Core Architecture and Cryptography

- The protocol's core function treats every message as an end-to-end encrypted and authenticated transaction. To achieve quantum resistance, the architecture exclusively utilizes NIST-standardized cryptographic primitives from the CRYSTALS (Cryptographic Suite for Algebraic Lattices) suite.
- Key Encapsulation: For secure message encryption and delivery, CRYSTALS-Kyber is employed.
- Digital Signatures: For transactional authenticity, integrity, and non-repudiation, CRYSTALS-Dilithium is used for all digital signatures. These lattice-based algorithms were selected due to their strong security guarantees and relatively compact public key sizes compared to alternative PQC schemes like SPHINCS+, making them more suitable for a blockchain environment where data efficiency is crucial.

Methodology

(B) Consensus Mechanism:

Delegated Proof of Luck (DPoL)

- The network achieves consensus using a novel Delegated Proof of Luck (DPoL) mechanism. This hybrid model was chosen to overcome the limitations of traditional algorithms; it avoids the high energy consumption of Proof of Work (PoW) and mitigates the "rich-get-richer" centralization risk of Proof of Stake (PoS). The DPoL model works in two stages:
- 1. Delegation: Token holders vote to elect a fixed, smaller set of trusted delegates who are responsible for running high-performance nodes and producing blocks. This governance layer is inspired by Delegated Proof of Stake (DPoS).

Methodology

- **Luck-Based Selection:** For each block, a block producer is chosen from this delegate pool through a cryptographically secure lottery. The selection is based on a Verifiable Random Function (VRF), which ensures the process is both unpredictable and universally verifiable. This "Proof of Luck" component ensures fairness and prevents resource-based centralization. This approach is designed to provide high speed, scalability, and energy efficiency while maintaining network security and decentralization

(C) Implementation

- The prototype for this blockchain protocol is being developed in Python, chosen for its extensive cryptographic libraries and rapid prototyping capabilities.

Methodology

2) Evaluation Plan

The performance and security of the proposed protocol will be rigorously evaluated. The methodology is designed to quantify the performance trade-offs inherent in using Post-Quantum Cryptography and to theoretically validate the protocol's security claims.

(A) Performance Evaluation: A prototype of the protocol will be deployed in a simulated network environment to measure its performance characteristics. The testing environment will consist of multiple nodes running on virtual machines to simulate a decentralized network. The following key metrics will be measured:

- **Transaction Throughput:** This metric will quantify the system's processing capacity. It will be measured in Transactions Per Second (TPS) by bombarding the network with a sustained load of transactions and calculating the rate at which they are successfully committed to the blockchain

Methodology

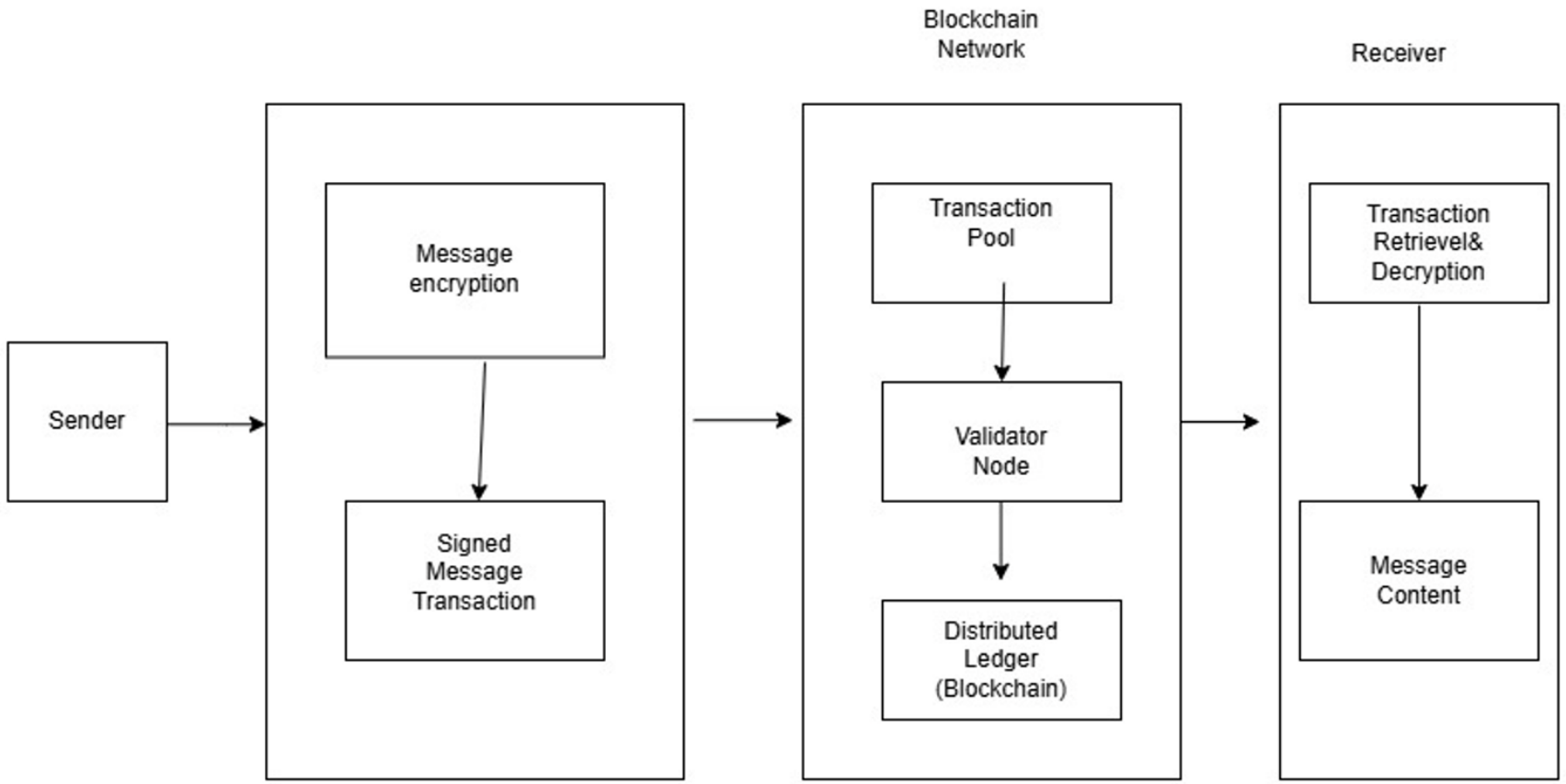
- **Transactional Latency:** This metric will measure the time required for a transaction to be finalized. It will be calculated as the time difference between the initial submission of a transaction by a client and its final confirmation in an immutable block, measured in milliseconds.
- **Space Overhead and Scalability:** This analysis will focus on the on-chain data footprint. We will measure and compare the average size of transactions and blocks against a theoretical baseline using traditional ECDSA signatures. This will quantify the "PQC tax"—the additional storage space required—and its impact on the long-term scalability of the ledger.

Methodology

(B) Security Analysis: Given that a physical large-scale quantum computer is not yet available for empirical testing, the security of the protocol will be validated through a theoretical analysis. This analysis will involve:

- **Cryptographic Primitive Review:** Verifying that the chosen PQC algorithms (CRYSTALS-Kyber and CRYSTALS-Dilithium) are implemented according to NIST's specifications and are secure against all known classical and quantum attack vectors.
- **Protocol-Level Threat Modeling:** Analyzing the entire protocol, including the DPoL consensus mechanism, to identify and address potential vulnerabilities. The protocol's resilience will be argued against established threats, such as those described by Shor's and Grover's algorithms, and against network-level attacks like Sybil attacks

Block Diagram



Implementation

```
(venv) ubuntu@ubuntu:~/Downloads/pq-block(1)/pq-block$ python node.py config_node1.json
/home/ubuntu/Downloads/pq-block(1)/pq-block/node.py:136: DeprecationWarning: There is no current event loop
  loop = asyncio.get_event_loop()
 * Serving Flask app 'node'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5001
INFO:werkzeug:Press CTRL+C to quit
WARNING:root:Failed to connect to peer ws://127.0.0.1:6002: [Errno 111] Connect call failed ('127.0.0.1', 6002)
INFO:websockets.server:server listening on 127.0.0.1:6001
INFO:root:P2P server running on ws://127.0.0.1:6001
WARNING:root:Failed to connect to peer ws://127.0.0.1:6003: [Errno 111] Connect call failed ('127.0.0.1', 6003)
INFO:websockets.server:connection open
INFO:websockets.server:connection open
INFO:werkzeug:127.0.0.1 - - [29/Aug/2025 12:58:52] "POST /tx HTTP/1.1" 202 -
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
```

Implementation

```
/home/ubuntu/Downloads/pq-block(1)/pq-block/node.py:136: DeprecationWarning: There is no current event loop
  loop = asyncio.get_event_loop()
 * Serving Flask app 'node'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5002
INFO:werkzeug:Press CTRL+C to quit
INFO:websockets.server:server listening on 127.0.0.1:6002
INFO:root:P2P server running on ws://127.0.0.1:6002
INFO:root:Connected to peer: ws://127.0.0.1:6001
WARNING:root:Failed to connect to peer ws://127.0.0.1:6003: [Errno 111] Connect call failed ('127.0.0.1', 6003)
INFO:websockets.server:connection open
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
INFO:root:Added new transaction to mempool. Nonce: 1
```

Implementation

```
ubuntu@ubuntu: ~/Downloads/pq-block(1)... × ubuntu@ubuntu: ~/Downloads/pq-block(1)... × ubuntu@ubuntu: ~/Downloads/pq-block(1)... × ubuntu@ubuntu: ~/Downloads/pq-block(1)... × ubuntu@ubuntu: ~/Downloads/pq-block(1)... ×
ubuntu@ubuntu:~/Downloads/pq-block(1)/pq-block$ source venv/bin/activate
(venv) ubuntu@ubuntu:~/Downloads/pq-block(1)/pq-block$ pip install requests
Collecting requests
  Downloading requests-2.32.5-py3-none-any.whl.metadata (4.9 kB)
Collecting charset_normalizer<4,>=2 (from requests)
  Downloading charset_normalizer-3.4.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl.metadata (36 kB)
Collecting idna<4,>=2.5 (from requests)
  Using cached idna-3.10-py3-none-any.whl.metadata (10 kB)
Collecting urllib3<3,>=1.21.1 (from requests)
  Downloading urllib3-2.5.0-py3-none-any.whl.metadata (6.5 kB)
Collecting certifi>=2017.4.17 (from requests)
  Downloading certifi-2025.8.3-py3-none-any.whl.metadata (2.4 kB)
Downloading requests-2.32.5-py3-none-any.whl (64 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 64.7/64.7 kB 1.1 MB/s eta 0:00:00
Downloading certifi-2025.8.3-py3-none-any.whl (161 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 161.2/161.2 kB 2.0 MB/s eta 0:00:00
Downloading charset_normalizer-3.4.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (151 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 151.8/151.8 kB 2.2 MB/s eta 0:00:00
Using cached idna-3.10-py3-none-any.whl (70 kB)
Downloading urllib3-2.5.0-py3-none-any.whl (129 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.8/129.8 kB 1.7 MB/s eta 0:00:00
Installing collected packages: urllib3, idna, charset_normalizer, certifi, requests
Successfully installed certifi-2025.8.3 charset_normalizer-3.4.3 idna-3.10 requests-2.32.5 urllib3-2.5.0
(venv) ubuntu@ubuntu:~/Downloads/pq-block(1)/pq-block$ python send_tx.py Alice Bob "This is much easier than using curl!"
✅ Transaction submitted successfully!
{'message': 'Transaction submitted', 'tx_hash': '6235747aeaf391c0d7fc3990dd5592b3be01c07cef5b000f12ea54ad62fb81c9'}
(venv) ubuntu@ubuntu:~/Downloads/pq-block(1)/pq-block$ python send_tx.py Alice Bob "This is much easier than using curl!"
✅ Transaction submitted successfully!
{'message': 'Transaction submitted', 'tx_hash': '474c3513b478db0feb10f7fa6fe8da2788460c72334d8d9dee51925233945199'}
(venv) ubuntu@ubuntu:~/Downloads/pq-block(1)/pq-block$ ^C
(venv) ubuntu@ubuntu:~/Downloads/pq-block(1)/pq-block$
```

Implementation

```
WARNING:root:Invalid nonce for 914d710ff828e99343d05f1f76abed686fb24ab95aa6a48fbae9a65dcd7d941325141b09bca4b053f55c33150b3001332bb16b2b889cb41183f60b7cef6cf4dad508d1cc9bacea525fae9921a
ade7a6c7ae8221bd6ec0fd8c8bfa52b99f6759af1e3d4b40b900340b96fa33282cfc91146c8962ede6b6fce9ff744bc0573c21fc8c848f3ef40e4295e1068b317cc7f742328ec4b762bae4bfc146b25f03f1338097cd6bb4e4a56243
554bd139926a0b79b1a6f4364223441ffee02c6ebc6b0a01996aba22db131bb468cf162098eac90bcfaa8f2047ef7a58199a29d92f0f041d8a441bb705ff99544d7a09431824e6731b0622e703e3ee2fedfb747664fafd547c971264
5f65d841ce9f015a44128c7586443fe2551d5bdbb4b3b2db81d5d6f514287bc2be6a5a92280fa3bd6cea613ea17b7ac45bb1003d43ac46897b7e16ce493bd1079dd291c0c566d7eaf117d3891f8d740f50fffcd660949f03702f033d
5d5b481c0a99d01a14c778dfbfaba30325430f04e89e6540f74b180339e35ba7f26aa9c731a5a06d72373cc5c98c632d535f6364aa75aadd6a768f49a4b30f27fa2fc4ebaf542b7e3f5d1e88001ec23915b5112d1b03c178a036f6d3
8e1967813140bbef41b27173932ea76916ea87e248ed5d39d1a5ad2e4bec7e6f2b9e703ee626c7fcded818d0cc51f1d3535439aed63ef0cac5c1d054393c2ec5f048d07463e13dbc490897a4e9c9238bc0a34d43cf3f9a6b70467fbb
ac79b8b8fceda8a16f4ae319ed8f74e77fc62ad9ab529df7f5516ca418f61922c074cd95d6f6c329603ae5927caac5e7f645ee3a93df64fb70f46cfec26dfea2a3b170ef4485fb9ccb4f1c8f5f2485fe33e1d1bdeecde1c6864c88d2
bd7f9f468f37e09304aa667be61610f5b41b981df914d59f85ffd529f0e74665efbfefc93ffb988c7b0a98386877dc670662edc6cb1a9298e2f639e3db1890c8e8fbbe6b68b9fe1542b62f97bc5bb8eb06b3a20fd65e8bd272fa48b7
f765ecffd43fe68a60c13881cc8bbcfaa1a54c085a01128dffdc4748765c3477dfdea61db9a276ec80151d77ee92a1b881b0ffe011daf08d4a0cd362fcf41f57752688ab7bb49255cdb96df37751b2ac6a79242538e7c12e6ef0c33e
6f02084ba573ec53cc924b1446b5d43e3b4614f4f9502082802829e7603df66e3bacb9a8837e8d4c64c089b5338f9770aa485633c328eaf79687f312a4f7371c28d4e65b0ca6ccc1c2e63964c66287be2dd617dd4b7ac181b7affc86
506f5abdbff4fa5528d56de2548e1e5788e0014e15fcc0419ec34e5fd50b09c0c62de7195bbed197a8831692b77dbb4cc9c5d92c3e0b309551b40b69364edcf4e7e3c8840b5f392268a9db6e7c226fff7a6500d5f5d7bf1e25e76e2b
f0eb3c2e2ebc25e8e30eb469fc5292ca741a472652293a3b934252364a8f2da3d4529dacd6a810f41dfc4112d4c8e389a68d2be73d263ae0af507dda32472ccb1eb0578c23263cb138c85a791c6c308bfddf49f771f7ec865e8dce71
be0ec6c15c2b4b91971b13097990d396c48a1d366b14ecf91256cd249a9251766b7a144b6cde1eb62de6824142e85078298c8256a7863c45983b6bd2990a49ee055cf15e4d2499985f5e470f074571b926c504838ce74136c7659f7d
6016a728706703bd83a190c160fdb81ab8fbd46a05cabde87011fc9cf4789e28c4c0784156bdb3c045ba042d0512737df30a42083f0a6fa56b773ce973e9e16cecdc8dc4b340a2e806a0e84d2eeb9ac6d631419847e766b7c099b293
65393946ef3e1661e6e52707a80d9c135b7f349d6ca5274aa70ea87ba808b2aa72dbcefb3f09ebec8774989f2275c8ef46a62655580a257e24847bea5097fd102ab05dd982a1a5bc359447c0061f3b52c73210120c8478da619d9cc7
ff11e37d8d3108f9eac56c1948fe6c55da26fc0d52612d8af185b42f63006b5d752b925af35a7f532e684ed68fd79ee094e2748174637a72eb8eef06ec8eeb6b23ea3cf8acc2952262b83f7c5ca7eef2a00639a94ebe5d94eb36f477
0053247bed4d24e6f3dad43131ceed2088861054a71c709a13f99dec11cea0aa9bbcb3d3fa16a7361f70419ab341795f1236dcc08cfeacaf7cb3af06530e495aa274792857e5769d8df13cd0e97de85eb6f5f88b592efec35042204b
13d069c144091e423db350d17874d33300ef066aea1eca4a3bde99fa4c1d672abb21e437556db60aded80648954ff5d1666a2a356818e9858a712b9f71c52c81c14b0844ed178f2402cb2b1f0835c942f6bf2757bc61687f09616971
d0590571d3ff5332e060e2f611e43b906105fe7c89427fc7e6f9a1600275c9453f6a47e03873e3851a4d77b8a02460052946a7f7821236569bf06d8f84ba15852df97849b8b7dbcbed028ac21835925fdd10047ef9c1a760c91d2d50
86a1fbbce2324e481525cbce861d84e9c8fde6923757ad659cdad1e256e644b14ae1ae7dac8f14a4d77f686feab9fb07c242cdecc911bc3c5c39d4f1111295042c85e432750f9d4b7a116525a5b6188c86ceacd0b9d9e419b2f9347b
3d132fcdf4ad67fe262f71c82015c9077a4b7558095182da7d42de29a05b2c89f2a5670efcbc852dd65054c238439d9760e62fa8f221d7a9d7fc85d16cee813d525f88dc6c8849e81f352a2a77f91101744a3f90103a9411c352fb20
```

References

[1]Reviewing Crypto-Agility and Quantum Resistance in the Light of Agile Pratices
https://www.researchgate.net/publication/376888322_Reviewing_Crypto-Agility_and_Quantum_Resistance_in_the_Light_of_Agile_Practices
[2] Post-Quantum Delegated Proof of Luck for Blockchain Consensus Algorithm
https://www.researchgate.net/publication/384123807_Post-Quantum_Delegated_Proof_of_Luck_for_Blockchain_Consensus_Algorithm
[3] Beyond ECDSA and RSA: Lattice-based digital signatures on constrained devices
[4] Study on Implementation of Shor's Factorization Algorithm on Quantum Computer
https://www.researchgate.net/publication/379667366_Study_on_Implementation_of_Shor's_Factorization_Algorithm_on_Quantum_Computer

References

- [5] Post-quantum blockchains: A review on platforms, challenges and opportunities by R. M. Fernandez et al. (2023).
- [6] Post-Quantum Delegated Proof of Luck for Blockchain Consensus Algorithm by Hyunjun Kim,Wonwoong Kim,Yeajun Kang,Hyunji Kim and Hwajeong Seo
- [7] CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme" by L. Ducas et al.