POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Quantum-resistant Blockchain

Introduction of Post-Quantum Cryptography in Hyperledger Fabric

Supervisors
Prof. Antonio Lioy
Dr. Ignazio Pedone

Candidate
Cosimo Michelagnoli

Academic Year 2022-2023

Summary

The aim of this thesis is to explore existing post-quantum algorithms in the implementation of blockchain technologies. The choice of which blockchain would be best to study fell on a project founded by the Linux Foundation in 2015, a non-profit association and the largest open source organization in the world. Such project, named Hyperledger Fabric, is open source and is considered the modular blockchain framework for enterprise blockchain platforms. We will then replace the cryptographic primitives vulnerable new post-quantum algorithms, which are resilient and secure to a possible future advent of quantum computing. The first phase concerns the analysis of the Fabric framework by thoroughly understanding its mechanics and identifying the various use cases where public key ciphers are used. This will allow us to understand the design of the architecture in order to be able to apply modifications. The main focus is digital signatures and the secure exchange of keys, operations considered to be particularly vulnerable. In the implementation phase, we replace the vulnerable algorithms with the selection of post-quantum algorithms released by NIST, by making the quantum-resistant signature operations. Finally, in the verification phase tests will be carried out to verify the performance obtained, outlining the pros and cons of the implemented post-quantum algorithms. The work would like to demonstrate the possibility of adapting existing technologies to the new post-quantum protocols released by NIST with imaginable benefits in terms of effectiveness, security and cost of adaptation to the new future scenario.

Acknowledgements

I cordially thank Prof. Antonio Lioy for entrusting me with this work, which has given me the opportunity to enrich my professional knowledge.

I also sincerely thank Dr. Ignazio Pedone for guiding and supporting me with his precious advice in the realization of this work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The advent of quantum technologies seems to be close to causing a real quantum leap in the world of computation, specifically quantum computation. This type of computation involves a real paradigm shift at the hardware level, and consequently a whole new architecture at the software level. This new model of computation is essentially based on two key elements: superposition of states and entanglement. It is expected, from a currently theoretical point of view, to increase significantly in terms of computational capacity compared to classical computers, which would obviously have positive implications in areas such as machine learning, artificial intelligence, big data and so on, areas in which very complex computations require ever-increasing computational capacity. Note that such machines are not just a theoretical model, but are already present experimentally and publicly accessible via the cloud (e.g., IBM Quantum Experience). Moreover, languages already exist for running algorithms on such machines (e.g. Qiskit, Amazon Braket, etc.). This would have a disruptive impact in the area of cybersecurity. The latter in fact uses many asymmetric cryptography algorithms, based on the assumption that classical computers cannot factor an integer number because it is computationally too complex.

At the same time, we are witnessing a paradigm shift in the Internet world, which is identified by the name Web3. The technologies on which Web3 is based are distributed ledgers, better known as Distributed ledger technology. This term refers to electronic 'ledgers' (or registers), geographically distributed over a large network of nodes. The secure storage of encrypted information is based on consensus techniques involving all or part of the participants, i.e. procedures to ensure that all nodes in the network agree on all legitimate transactions. Distributed ledgers base part of their security on public key ciphers, the best known Distributed ledger being the blockchain. It is a consensus-based system in which every transaction is recorded and verified by all network participants. A blockchain is immutable by its very nature. Once added to the blockchain, information cannot be changed or deleted. This immutability is fundamental to data integrity and is guaranteed by protocols vulnerable, as mentioned above, to quantum computing. Fortunately, computer security is already addressing this issue with new solutions such as quantum key distribution, post-quantum algorithms and random number generation. Post-quantum cryptography is an approach to developing cryptographic schemes that are resistant to new computing power and have the advantages of being compatible with the existing cryptographic infrastructure. For these reasons, it seems possible to identify a solution applicable to existing blockchains. This area is still in its earliest stages; the US National Institute of Standards and Technology (NIST) has recently announced a group of quantum-resistant algorithms that will become part of a cryptographic standard in the

coming years.

The aim of this thesis is to explore existing post-quantum algorithms in the implementation on blockchain technologies. The choice of which blockchain to study fell on a 2015 project founded by the Linux Foundation, an association of enterprises in the technology sector and the largest open source organization in the world. Such project, named of Hyperledger Fabric, is open source and is considered the modular blockchain framework for enterprise blockchain platforms. Hyperledger Fabric is a well-known and adaptable approach for constructing permissioned distributed ledger platforms. Access control and identity management are handled by a Membership Service Provider (MSP), whose cryptographic interface only supports basic PKI authentication techniques such as RSA and ECDSA classical signatures. Furthermore, MSP-issued credentials may only utilize one signature method, confining credential-related tasks to traditional single- signature primitives. RSA and ECDSA are vulnerable to quantum attacks, and a post-quantum standardization effort to identify quantum-safe entry counterparts is now happening. In this research, we propose a redesign of Fabric's credential management methods and accompanying specifications to include alternative digital signatures that guard against both classical and quantum attacks with a single quantum-safe signature. We provide a Fabric implementation with quantum signatures that works with the Open Quantum Safe (OQS) library. During the initial phase of analysis of the framework we discovered that Hyperledger Fabric passes public key identities as X.509 certificates. Previous work has included the creation of custom hybrid X.509 certificates to support an additional PQ-signing algorithm. However, our goal was to minimize the number of changes to the framework from the official version that is currently used by companies. We tried to follow a good balance between a quantum-safe solution and the needs of the corporate world that already uses this framework, which is hostile to big changes. In addition, the X.509 standard does not currently support multiple signing algorithms in the same certificate, and our intent was to provide an X.509-compliant solution. For these reasons just mentioned, we decided to adopt a solution using X.509 with a single post-quantum algorithm. Our choice fell on one of the algorithms that entered NIST's fourth round of standardization in July 2022. The purpose of this work was to overcome some critical issues of the official solution and provide an implementation by verifying its feasibility in terms of performance. Therefore, to verify the impact of the changes, we compared Fabric with our solution in terms of performance with the help of custom tests. Showing how the new algorithms for PQC achieved levels comparable to those of the classical algorithms. Another essential feature we tried to include in our solution was the ability to adapt to continual changes in post-quantum cryptography standards, as the NIST is working to complete the list of post-quantum signature algorithms. It must be compatible with all remaining candidates and should not be dependent on the method that is ultimately picked. The choice of one of them must be as straightforward as a setting update.

The thesis is divided into the following chapters:

- Chapter 2: analysis of distributed ledger technologies;
- Chapter 3: blockchain introduction;
- Chapter 4: quantum-resistant cryptographic analysis according to the state of the art;
- Chapter 5: an extensive analysis of the Hyperledger Fabric framework;
- Chapter 6: examinations of the need to implement post quantum cryptography in blockchains;

- Chapter 7: implemented solution and adopted approach;
- Chapter 8: analysis of tests conducted since the implementation of the solution;
- Chapter 9: conclusions;
- Appendix A: user's manual;
- Appendix B: developer's manual.

Chapter 2

Distributed ledger technologies

2.1 Introduction

Distributed Ledger Technology (DLT) is a set of replicated, shared and synchronized digital data geographically spread across multiple sites, countries or institutions. A computerized system that records asset transfers is referred to as distributed ledger technology. Transactions and other details are recorded in many locations at the same time. The distributed ledger database does not feature an administration facility or centralized data storage. Instead, the database persists across several individuals or geographical regions.

Users can utilize distributed ledger technology to record, exchange, and synchronize data and transactions over a distributed network with many participants. It may also be viewed as a set of technologies with similar architecture that can be performed in a variety of ways with distinct rules. Depending on whether the ledgers are accessible to anybody or by the devices, distributed ledger technology may be characterized as public or private (also called nodes). It may also be classified as permissioned or permissionless, depending on whether participants need permission from a certain entity to make changes to the ledgers. [1]

DLT, in essence, serves as a shared, digital infrastructure for DLT-based applications (e.g., financial transactions) by allowing the operation of a highly available, append-only distributed database (referred to as distributed ledger) in an untrustworthy environment, the ledger is replicated locally by storage and computing devices (referred to as nodes). Individuals or groups manage and control nodes (referred to as node controllers1). An untrustworthy environment is distinguished by the sporadic occurrence of Byzantine failures, such as crashed or (temporarily) inaccessible nodes, network latency, and hostile node activity.[1]

2.2 Properties

DLT transfers and appends data to the ledger in the form of transactions, which are then recorded in a chronologically ordered sequence. Each transaction includes information (such as the transaction receiver or date) as well as a digital representation of specific

[1] Written by CFI Team, Distributed Ledger Technology, January 19, 2023, https://corporatefinanceinstitute.com/resources/cryptocurrency/distributed-ledger-technology

assets or program code of a smart contract. When a new transaction is received by a node, it is validated by a proof of ownership for the digital representation of the asset based on digital signatures and public key cryptography.[1]

DLT properties are collections of DLT traits that are shared by all DLT designs. For example, throughput and scalability are both related to DLT property performance. A DLT system must be capable of ensuring the following properties, either in its current state or with minimal system changes [2].

- *Shared recordkeeping*: enable multiple parties to collectively create, maintain, and update a shared set of authoritative records (the 'ledger').
- *Independent validation*: enable each participant to independently verify the state of their transactions and integrity of the system.
- *Tamper evidence*: allow each participant to detect non-consensual changes applied to records trivially.
- *Tamper resistance*: make it hard for a single party to unilaterally change past records (i.e. transaction history).
- *Multi-party consensus*: enable all parties to come to agreement on a shared set of records
  - If permissionless, without relying on a single party or side-agreements, and in the absence of ex ante trusted relationships between parties; and
  - If permissioned, through multiple record producers who have been approved and bound by some form of contract or other agreement.

2.3 Permissionless vs permissioned networks

Regarding the requirements for a node to be approved to validate transactions and record them on the ledger, distributed ledgers can be classified as permissioned or permissionless. While permission-less ledgers have received the most attention from the general public (with Bitcoin as the paradigmatic example), permissioned distributed ledgers are best qualified to address the majority of the use cases of interest to industry and governmental institutions. The main reasons for this are technical and legal in nature. Among the first are goals like the cost and delay for recording a transaction, the cost of the consensus algorithm, and the fairness properties among participants. The specific application should drive the decision between permissionless and permissioned blockchains (or use case). Before parties agree to do business with each other, most enterprise use cases require extensive vetting. Supply chain management is an example of how multiple businesses exchange information. Supply chain management is an excellent application for permissioned blockchains. Only trusted parties should be allowed to participate in the network. To execute transactions on the blockchain, each participant in the supply chain would need permission. These transactions would enable other companies to determine where a specific item is in the supply chain [2].

2.4 Consensus in Distributed Systems

Consensus is a fundamental idea in distributed systems that is not limited to blockchain. It is used in circumstances where numerous processes or nodes must maintain a shared

state of a data item.

Consensus algorithms are based on the well-known Byzantine generals issue, which was initially defined in 1982 by Lamport in his work "The Byzantine Generals Problem." The Byzantine generals' dilemma is described in detail below. The old eastern Roman Empire's capital was Byzantine. In Byzantine, there are various fiefs, each with its own general and army to repel foreign invasions. When confronted with opponents, each general has two options: assault or retreat (Figure 2.1). Only when all honest generals agree on an order to assault or withdraw can a war be won with minimal fatalities. However, because Byzantine is so huge, these generals are unable to discuss the order collectively because they must protect their fiefs independently. As a result, instructions from generals are relayed by signalmen. Finally, the generals make their final judgments (attack or retreat) by issuing commands to the other generals and receiving orders from the other generals. In this case, we presume the signalmen are truthful. Some of these generals, however, are traitors who may give incorrect instructions or different orders to various generals, undermining the overall decision of the honest generals. In summation, the Byzantine general's dilemma might be defined as the difficulty of getting honest generals to agree in the presence of multiple traitors.

In a distributed system, the consensus method is used to tackle the challenge of guaranteeing data consistency in the presence of several failed nodes. Crash fault nodes and Byzantine fault nodes are the two forms of failure nodes. Crash fault nodes fail just by ceasing to function; that is, they can only cease operating and exhibit no other harmful behavior. Messages can only be delayed or lost in this instance. Byzantine fault nodes, on the other hand, act randomly. They might transmit incorrect messages to other nodes or send various messages to different nodes in order to sabotage the consensus-building process. Various nodes in a typical distributed system frequently represent different persons or consortiums, which may behave arbitrarily when there is no central function. As a result, the DLT consensus method should be able to withstand Byzantine fault nodes. [3]

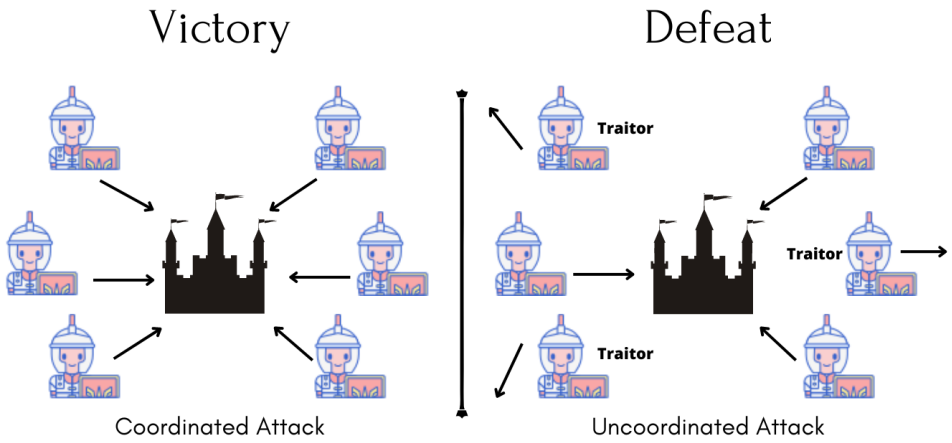

Figure 2.1. A visual representation of the Byzantine General's Problem.

2.5 Types of Distributed Ledger Technology

One of the key objectives of DLTs is to enable user interaction without the requirement for a third party that they can trust. In fact, a scenario where there is some mistrust

between the participating parties, such as business partners or unknown entities, is crucial for DLT. DLTs are added a level of security, traceability, and transparency to this kind of environment by design. DLTs are essentially a set of functions and data structures for recording transactions (Figure 2.2. However, generally speaking, all DLTs are based on three well-known technologies: (i) public key cryptography, (ii) distributed peer-to-peer networks, and (iii) consensus mechanisms, they have been mixed in a fresh and novel way. Each DLT differentiates itself using a different data model and technologies. A secure digital identity is created for each participant using public key cryptography as the idea is to function in an unsafe environment. To be able to record transactions in the Distributed Ledger, each participant is provided with a set of keys (one public, one private) (DL). This digital identity is utilized to enforce ownership control over the assets that the DL manages. In order to grow the network, avoid a single point of failure, and prevent a single player or small number of players from controlling the network, a peer-to-peer network is used. Blockchain and distributed ledgers based on the structure directed acyclic graph, DAG, are notable instances of DLTs. Some characteristics that are shared by different DLT architectures include that data is exchanged across numerous nodes in a peer-to-peer network where the integrity of the data is assured by node consensus. The subsections that follow discuss some of the primary technologies, as well as any advantages, drawbacks, and distinctive qualities they may have.

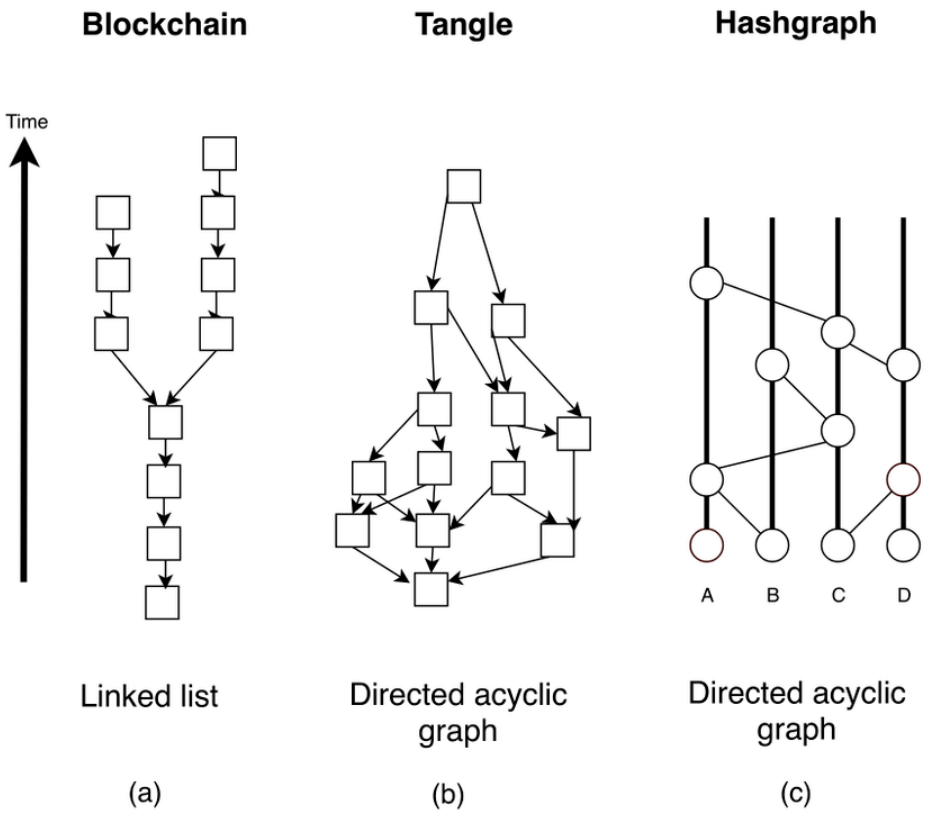

Figure 2.2. Summary of existing DLTs. Source: [4]

2.5.1 Blockchain

A distributed, decentralized, and unchangeable ledger used to maintain transaction history is called a blockchain. The data is accessible to all network participants and is regularly added to. A blockchain is fundamentally a linked list of blocks that are linked to one another using hash codes, with each block referencing the block before it in the chain. This has the advantage of making it difficult for anyone to alter the content of blocks because any change to one block invalidates all succeeding blocks. The fundamental components of a blockchain are [5]:

- Transactions, which are signed pieces of information created by the participating nodes in the network then broadcast to the rest of the network ;
- Blocks, are collections of transactions that are appended to the blockchain after being validated;
- A blockchain is a ledger of all the created blocks that make up the network;
- The blockchain relies on Public keys to connect the different blocks together (similar to a linked list);
- A consensus mechanism is used to decide which blocks are added to the blockchain.

2.5.2 Tangle

New models are being created and tested as interest in DLT grows. The Tangle is one of the more recent technologies that positions itself as a blockchain substitute, particularly in terms of Internet of Things (IoT). Instead of the global blockchain, we have a Directed Acyclic Graph (DAG) known as the *tangle*. Tangle is a consensus system and decentralized data storage architecture based on the DAG data structure. Tangle naturally succeeds the blockchain as the next evolutionary step, and it provides the elements necessary to construct a machine-to-machine micropayment system. This strategy is now being implemented as a cryptocurrency known as IOTA. IOTA was founded with the intention of reducing or doing away with transaction costs. In an IoT world where M2M micro-transactions are typical, this is very important. The elimination of the distinction between transaction makers and validators is another significant notion that IOTA introduced. A short word on terminology: sites are tangle graph transactions. The network is made up of nodes, which are entities that issue and evaluate transactions. The connections (direct edges) between each node (referred to as a site) in the DAG stand in for a transaction, and the edges between transactions stand in for their validations. The tangle graph's site set, which serves as the ledger for storing transactions, is made up of transactions issued by nodes. The tangle's edge set is obtained as follows: the user must select and validate two current transactions before adding a new transaction. Because there are no miners in a Tangle, the same users who initiate transactions also validate other users' transactions. For example, if Alice wants to do business with Bob, she must first validate two previous transactions using a Proof-of-work (POW) computation. These approvals are reflected in the curved borders. We say A indirectly approves B if there is no directed edge between transactions A and B but there is a directed path of at least two lengths from A to B. It is believed that the nodes verify to ensure that the permitted transactions do not contradict. If a node discovers that a transaction contradicts the tangle history, the node will not authorize the conflicting transaction, either directly or indirectly. As a transaction obtains further approvals, the system accepts it with greater trust. In other

words, convincing the system to accept a double-spending transaction will be tough. It is crucial to note that the tangle imposes no criteria for determining which transactions a node would allow. In the white paper [6] it is argued that if a significant number of nodes follow a "reference" rule, then every fixed node should follow a similar rule.

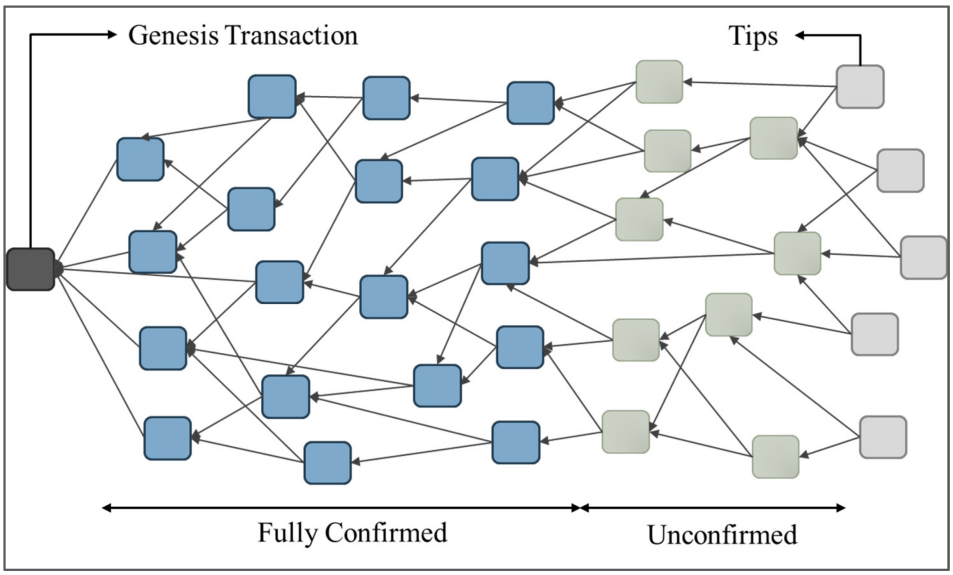

Figure 2.3. Tangle structure. Source: [7]

Since there is no concept of blocks, the network does not impose any architectural limitations on the transaction rate. Each transaction is handled uniquely (the concept of blocks is considered to be a bottleneck for the blockchain technology). Hashcash is used by IOTA as the POW method because it has a lower level of difficulty, making it possible for devices with minimal processing capacity to complete the computation. The primary goal of implementing the POW concept is to get rid of spam transactions. The primary benefits of tangling are [5]:

- *Scalability*: As the Tangle relies on all the network users to validate the submitted transactions, as the number of users increase it results in faster validation time. Additionally, as the network requires lower computation power to compute the POW, this gives users more incentive to join the network (e.g., smartphones and IoT devices).
- *Post-quantum signature*: IOTA relies on the Winternitz One-Time signature scheme to generate the public addresses. This method is believed to be quantum resistant. As quantum computing is becoming reality, using stronger encryption algorithms increases trust in the platform. In IOTA documentation, it is advised to use each generated address only one time, as every time it is used fragments of the private key could be leaked.
- *Transactions fees or Micro-transactions*: As IOTA merges the two roles (transaction makes and miners) into one role, there is no need for transaction fees. The incentive for supporting the network is to be able to submit transactions. This means that the participants of the network use their own computing power to trade assets instead of paying a miner to do that for them (when we say that the miner does it on their behalf, we refer to the fact that a miner adds the transaction to the chain).

Assuring authentication in the Tangle

To transfer tokens from one address to another in IOTA, you must first make a transaction. A transaction contains details on the payer, the receiver, and the number of tokens to be exchanged. Of course, the fact that only the token's owner can initiate such a transaction is crucial. This is performed in the legal world by adding a signature to a contract. In the digital world, digital signatures exist particularly for this reason. They are, however, more powerful than their pen and paper counterparts since they guarantee that not a single letter in your contract may be changed once it has been signed. Cryptography researchers around the world have developed many sophisticated mathematical theories that can be used for digital signatures over the last 50 years. They all guarantee unforgeable messages, but hash-based signatures, i.e. signatures based on hash functions, are probably the easiest to understand. A hash function is a useful tool in cryptography that takes an input and produces an output that appears to be unrelated. If the hash function is good, it should be extremely difficult to reverse the process. This means that if I only know the output, I will never be able to guess the original input in a reasonable amount of time. Because of this, hash functions are known as one-way functions.

That is all we need to create the secure digital signatures used in IOTA, which employ the Winternitz one-time signature scheme. WOTS, unlike other authentication methods like as RSA and DSA, does not rely on the computational difficulties of factorization or discrete logarithm calculation. Instead, WOTS generates digital signatures using a one-way cryptographic hash function. These digital signatures are produced at random and are single-use, which means they can only be used to authenticate a message once. This implies that even if an attacker obtains a digital signature, he will be unable to use it to authenticate subsequent communications.

Signature scheme: WOTS

A sequence of trytes represents any IOTA transaction. In the human world, numbers are represented by the digits 0-9; in the trinary world, they are represented by trytes with values ranging from 0 to 26. For the sake of brevity, we will refer to the hash function as f and assume that the message to be signed contains precisely two trytes: m1 and m2. If Alice created a transaction sending 1i to her friend Bob and now wishes to demonstrate that this was truly issued by her. Alice first produces two random numbers, priv1 and priv2, which she keeps private as her private key. She then hashes both numbers exactly 27 times with the hash function f to get $H^{27}(priv1) = pub1$ and $H^{27}(priv2) = pub2$. [2]

The two resulting numbers pub1 and pub2 form Alice's public key, which she then shares with everyone as her IOTA address. Since f is a one-way function, it is not possible to determine the original priv1 and priv2 based on the public key.

To sign a message with that key, i.e. sign a transaction from that IOTA address, Alice hashes her private key exactly that many times to get sign1. Hence, zero times if m1 has a value of 0, once if m1 has a value of 1, and so on. She repeats the same with m2 and adds sign1 and sign2 as signatures to her transaction. After that, anyone who knows pub1 and pub2 can validate this signature. All they have to do is use the hash function f on sign1 and sign2 the remaining times to get to 27 (27 - m1 and 27 - m2 times). If the result matches pub1 and pub2, then the signature is correct. [2]

[2] Research&Development, Assuring Authenticity in the Tangle with Signatures, Feb 27, 2019, https://blog.iota.org/assuring-authenticity-in-the-tangle-with-signatures-791897d7b998

2.5.3 Hashgraph

By many, hashgraph is regarded as a continuation of where the idea of blockchain originates. In a tangle network, the user must select and validate two before adding a new transaction. To swiftly obtain consensus among nodes, Hashgraph uses a voting algorithm in conjunction with the gossip protocol and a DAG as its data structure for recording transactions. The consensus protocol of the hashgraph represents a breakthrough. It is theoretically demonstrated to assist in data replication considerably more quickly than blockchain. A hashgraph is made up of vertices and columns on an abstract level. In the hashgraph, users can essentially only take two actions.

1. Submit a transaction: users can submit an event that contains a new transaction;
2. Gossip about a transaction: users can randomly choose other users and tell them what they know.

One of the main concepts that hashgraph has introduced is the concept of ordering and fairness, because it allows events to be ordered and validated based on their order. This means that if two users compete for access to the same resources, the one who submitted the transaction first will be given priority (in blockchain the reverse order can happen depending on the transactions selected by the miners). The hashgraph distributed ledger system tries to solve some of the limitations of standard blockchains. Here are some advantages and disadvantages of using it. [5] Pros:

- *High speed*: Hashgraph employs a "state gossip" consensus process, which permits extremely fast transaction rates as compared to typical blockchains.
- *High scalability*: Hashgraph employs a distributed ledger technology, which enables massive volumes of data and transactions to be processed concurrently.
- *Security*: Hashgraph employs a consensus technique of the "byzantine fault tolerance" kind to assure network security even in the presence of compromised nodes.

Cons:

- *High cost*: Hashgraph has a consensus mechanism that necessitates a large amount of processing power, raising the expenses for network users.
- *Patented*: The underlying technology of Hashgraph has been patented, which may limit its diffusion and usage by third parties.
- *Less decentralized*: Compared to other distributed systems, the network is more under the control of the business creating Hashgraph.

Although Hashgraph is generally a promising technology with significant potential, particularly for commercial and business usage, it is not yet as widely deployed and used as blockchains, therefore there may need to be more research and validation to demonstrate its true long-term significance.

Chapter 3

Blockchain

The terms blockchain and distributed ledger technologies are commonly used interchangeably. However, they are not the same. Blockchain is a sort of distributed ledger technology that employs encryption to make difficult to manipulate. It is a distributed, immutable ledger used for recording transactions, transferring ownership, and tracking assets. Blockchain ensures security, transparency, and trust in various sorts of digital asset transactions. [8]

3.1 Blockchain description

Blockchain is a type of distributed ledger technology that creates a chronological chain of blocks, thus the term "block-chain." A block is a collection of transactions that are grouped together and added to the chain all at once. Another important aspect of blockchain technology is timestamping. Each block is timestamped, and each subsequent block refers to the prior block. This timestamped chain of blocks, when combined with cryptographic hashes, offers an immutable record of all transactions in the network, dating back to the first (or genesis) block. A list of transactions must be agreed upon by all participants. Divergence will occur in either instance, resulting in forks. Each miner has a unique blockchain state that varies from miner to miner. When two or more miners mine two separate blocks at the same time, the local states will be different. Two blocks will then point to the same prior block. The global state of a blockchain is built by combining all local states. The point in a global state where various blocks have the same antecedent block is referred to as a fork. To resolve forks, different blockchain implementations, such as Bitcoin and Ethereum, utilize different approaches to identify the main branch of blockchain. Bitcoin resolves forks by using the deepest branch as the primary branch of the blockchain. For this reason, it employs Nakamoto's consensus protocol. The main branch is chosen as the branch with the most nodes. Greedy Heaviest Observed Subtree (GHOST), an Ethereum consensus method, chooses the heaviest subtree as the primary branch [9].

3.2 Architecture of a Block

Blocks are files that permanently store data connected to the blockchain network. A blockchain network is a decentralized network in which a group of nodes or participants work together to administer a common ledger of transactions or occurrences. Blocks

function similarly to the pages of a record book. A chain of these blocks, like the one shown in the figure 3.2, eventually becomes a blockchain [10]. A block contains just one parent block with a preceding block hash included within the block header. It's worth mentioning that uncle block hashes (children of the block's ancestors) would even be retained in the Ethereum network. A blockchain's original block is known as the genesis block, and it has no parent block [11].

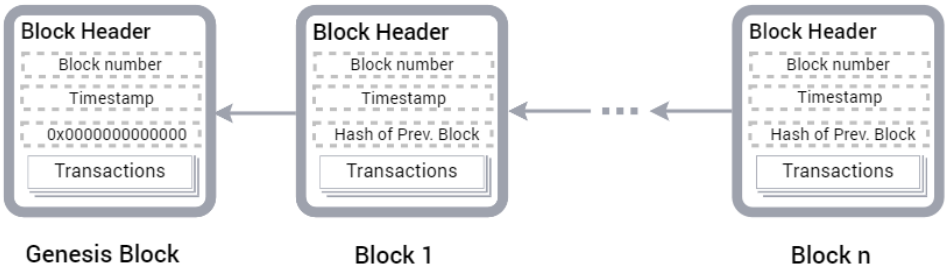

Figure 3.1. An illustration of how blocks are linked together to build a blockchain. Each block has a header and a number of transactions. A block's transactions are hashed to provide a fixed-length hash result, which is appended to the block header. When the first valid block is created, every subsequent valid block must include the hash output of the preceding block header. Every valid block is linked to the ones before it by the hash of the preceding block header, which is contained in every block. As a result of connecting each block to the previous blocks, a chain of blocks (blockchain) is formed.

3.2.1 Structure

A block consists of the block header and the block body as shown in figure 3.2.1 In particular, the block header includes [11]:

- *Block version*: indicates that set of block validation rules to follow.
- *Merkle tree root hash*: the hash worth of all the transactions within the block.
- *Timestamp*: current time as seconds in Greenwich Mean Time.
- *nBits*: target threshold of a legitimate block hash.
- *Nonce*: associate degree 4-byte field, that sometimes starts with zero and will increase for each hash calculation.
- *Parent block hash*: A 256-bit hash worth that points to the previous block.

3.2.2 Merkle Tree

Andreas Antonopoulos in the book Mastering Bitcoin defines the Merkel Tree as: *"Merkle trees are used to summarize all the transactions in a block, producing an overall digital fingerprint of the entire set of transactions, providing a very efficient process to verify whether a transaction is included in a block".* [12]

The Merkle tree, also known as a binary hash tree, is a data structure used to store hashes of individual data in huge datasets in order to speed up dataset verification. It is

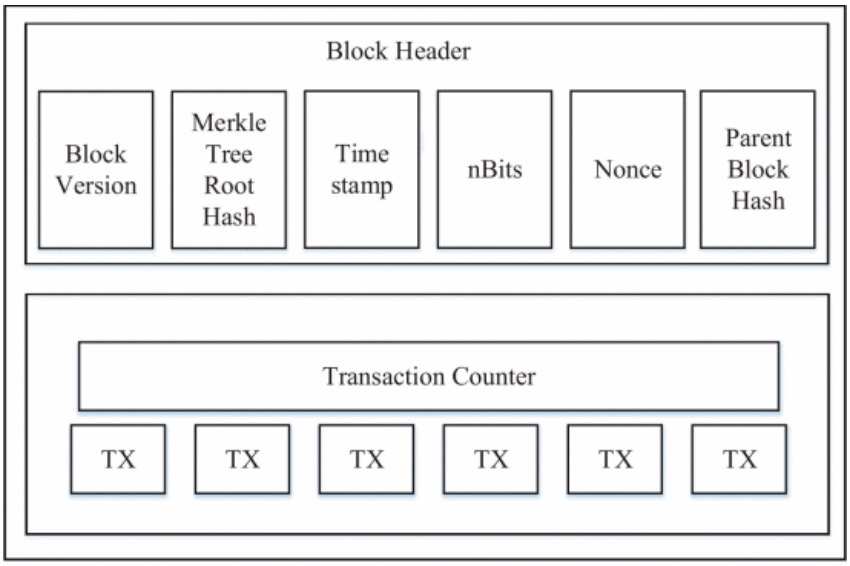

Figure 3.2. Block structure © 2017 IEEE International Congress on Big Data (BigData Congress) [11]

an anti-tampering mechanism that ensures the vast dataset has not been tampered with. In computer science, the term "tree" refers to a branching data structure, as shown in the graphic below.

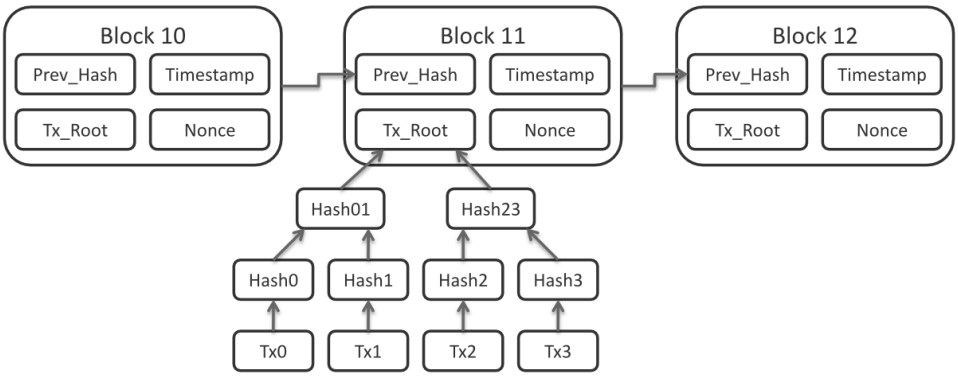

Figure 3.3. Merkle tree. Source: From Wikimedia Commons, the free media repository. [13]

A cryptographic hash of the transactions in the block. As shown in the figure 3.3 the Merkle tree is produced by hashing pairs of nodes in a tree until only one hash remains, known as the Merkle root. As a hashing algorithm, cryptographic algorithms such as SHA-256 are utilized. Other hashing algorithms can be used as well. [10]

3.3 Consensus protocol

Consensus protocols are used in distributed environments to ensure that all state replications follow pre-defined state transitions and rules. Consensus is difficult to achieve in a distributed system. Consensus protocols must be robust in the face of node failure, network partitioning, message delays, ordering, and corruption. Numerous protocols

have been proposed, each with its own set of assumptions regarding synchrony, message broadcasts, failures, malicious nodes, performance, and message security. Each consensus protocol strives for network stability in the presence of f faulty nodes. In general, a network requires $n >= 2f + 1$ entities to withstand f failures. [14]

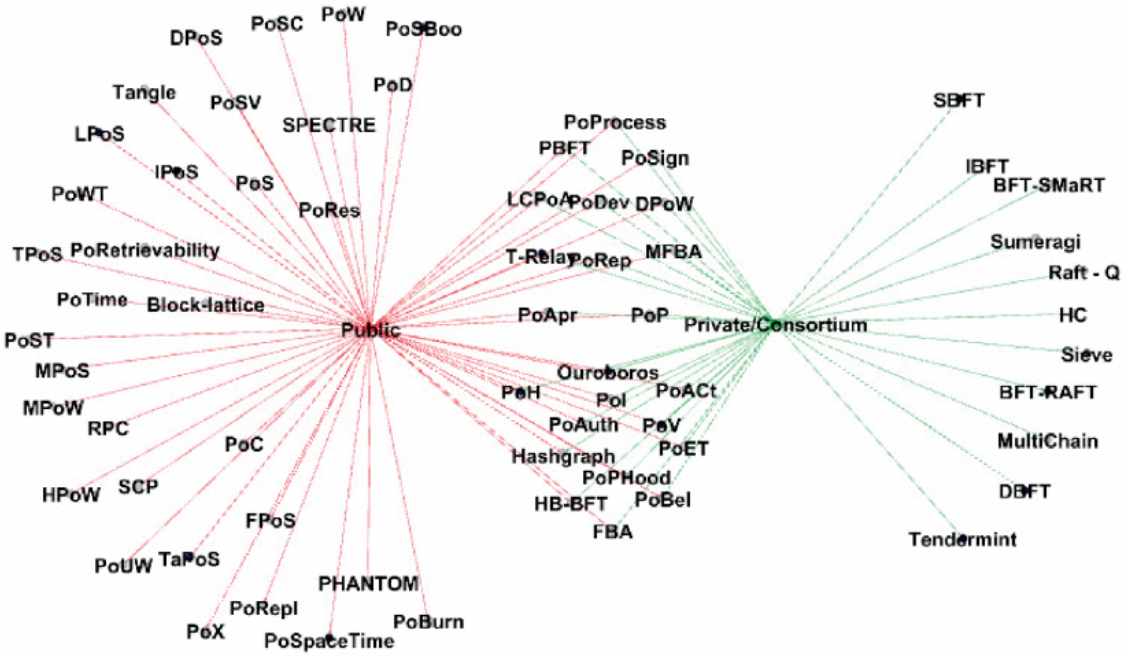

Figure 3.4. Relationship of different consensus protocols with different philosophy-based categories of DLTs.[14]

3.3.1 Proof of Work

In order to create new blocks in the Bitcoin blockchain, the PoW consensus algorithm requires solving a computationally difficult puzzle. The process is colloquially known as "mining", and the nodes in the network that participate in mining are known as "miners." Mining transactions are motivated by economic payoffs, in which competing miners are rewarded with 12.5 bitcoins and a small transaction fee.

Changing anything from a block necessitates redoing the work. Changing history is even more difficult because a user must re-compute n that satisfies l for all blocks mined after the block under attack. This necessitates a significant amount of computational power, referred to as hash rate [14].

3.3.2 Proof of Stake

The PoS algorithm extends the PoW algorithm. In PoS, nodes are known as "validators," and instead of mining the blockchain, they validate transactions in order to earn a transaction fee. There is no need for mining because all coins exist from the start. Simply put, nodes are chosen at random to validate blocks, and the likelihood of this random selection is determined by the amount of stake held. So, if node X has two coins and node Y has one coin, node X is twice as likely to be asked to validate a block of transactions. The specific implementation of PoS can differ depending on the use case

or software design. Instances include Proof of Deposit and Proof of Burn. Under a PoW consensus environment, the PoS algorithm saves expensive computational resources that would otherwise be spent on mining [14].

3.3.3 Proof of Elapsed Time

Intel's PoET consensus mechanism is designed to operate in a Trusted Execution Environment (TEE), such as Intel's Software Guard Extensions (SGX). Hyperledger's Sawtooth implementation demonstrates PoET in action. The PoET consensus mechanism is a hybrid of a random lottery and a first-come, first-served basis, rather than competing to solve the cryptographic challenge and mine the next block, as in the Bitcoin blockchain.

All nodes work on the puzzle and announce the block after the waiting time, along with the waiting proof created by TEE and easily verifiable by all participant nodes. This negates the benefit of having more computational power because the miner with the shortest waiting time will be able to announce the block quickly. PoET necessitates specialized hardware, which limits participation and decentralization [14].

3.3.4 Simplified Byzantine Fault Tolerance

The SBFT consensus algorithm is a modified version of the Practical Byzantine Fault Tolerant (PBFT) algorithm that aims to provide significant improvements over Bitcoin's Proof of Work consensus protocol. The fundamental idea is that a single validator bundles proposed transactions and forms a new block. Given the permissioned nature of the ledger, the validator is a known party, unlike the Bitcoin blockchain. Consensus is reached when a certain number of other nodes in the network ratify the new block. In order to be tolerant of a Byzantine fault, the number of nodes that must reach consensus is $2f + 1$ in a system containing $3f + 1$ nodes, where f is the number of faults in the system. For example, if we have 7 nodes in the system, then 5 of those nodes must agree if 2 of the nodes are acting in a faulty manner [14].

3.3.5 Proof of Authority

PoA is a consensus algorithm for permissioned ledgers. It employs a set of "authorities," or designated nodes with the ability to create new blocks and secure the ledger. Authorities' identities are verified both online and in the public sector. Time is divided into steps, and each step has a mining leader capable of generating blocks. Authorities take turns proposing blocks on a round-robin basis for each step, and a block is accepted onto the blockchain once it has been signed off by the majority of authorized nodes. By identifying the authorities, PoA becomes intrinsically centralized. As a result, it works best for private blockchains and consortiums [14].

3.4 Blockchain Security Primitives

The security characteristics provided by blockchain are primarily supported by public-key/asymmetric cryptography and hash functions, which are discussed in depth in the following subsections.

3.4.1 Hash Functions

Blockchains typically utilize hash functions like SHA-256. A hash function is a mathematical function that transforms a series of input data to a fixed-size numeric output. This numerical number is known as a hash and may be used to identify and validate data. Furthermore, a hash function is built in such a manner that it is computationally expensive for an attacker to build an input that yields the same hash value (known as a collision), resulting in the hash serving as a form of unique fingerprint of the input data. On a blockchain, each block of transactions is linked with a unique hash that is computed using a hash function. Its hash serves as the block's digital signature and renders it immutable: any effort to change even a single transaction within the block results in a change to the block's hash, making the change instantly identifiable. Moreover, the hash of the next block is computed by incorporating the hash of the preceding block, resulting in a chain of interconnected blocks (hence the term blockchain), each of which is uniquely recognized by its hash and location in the chain.

Finally, hash functions are employed to generate addresses (wallet addresses). The hash of a public key, which is used to receive transactions, is used to generate cryptocurrency addresses. [15].

3.4.2 Public-Key Cryptography

Public key cryptography is an asymmetric encryption technique that encrypts and decrypts communications using a pair of keys, one public and one private. The public key is made available to everyone who wishes to send an encrypted communication, but the receiver is kept in the dark about the private key. The fundamental advantage of public-key cryptography is that it does not need the sender and receiver to share the same secret key, as symmetric-key cryptography does. Moreover, public key cryptography may be used to generate digital signatures, allowing the authenticity and integrity of messages to be confirmed. Public key cryptography is a critical component of blockchains since it ensures the security and integrity of the data stored on the blockchain. Each participant in a blockchain has two keys, one public and one private, and uses their private key to digitally sign the transactions they want to record on the blockchain. The digital signature confirms that the transaction was indeed approved by the owner of the matching private key. As a result, when a signing method is safe, it is assured that only the individual who possesses a private key could have created a certain signature. Public-key Cryptography is also required for wallets, which are private key containers that hold files and basic data. In a blockchain system, each user has a wallet with at least a public address (usually a hash of the user's public key) and a private key that the user needs for transaction signing. [15].

Chapter 4

Quantum-safe cryptography

In recent years, there has been a lot of buzz around quantum computers. Quantum computers are machines that use quantum mechanical approaches to solve mathematical problems that regular computers find difficult or intractable. Quantum computers are entirely vulnerable to classical cryptography techniques. The majority of modern cryptography techniques are built around difficult mathematical functions. The design of cryptographic algorithms is dependent on computational complexity assumptions. The transition to quantum computing would render present IT infrastructure entirely insecure, necessitating the design and implementation of quantum-safe or quantum-resistant cryptographic techniques. Shor's approach will significantly cut the time required for prime factorization. All encryption techniques that are based on the assumption that extracting the private key with the available computer resources is computationally impossible to fail. A large-scale quantum computer can utilize this method to break many of the public-key cryptosystems already in use in modern applications. This would be devastating because it would put the security and integrity of all digital communications across the Internet at risk.

4.1 Quantum threat

The idea of devices based on quantum mechanics was explored in the early 1980s by physicists and computer scientists. This type of computation involves a real paradigm shift at the hardware level, and consequently a whole new architecture at the software level. This change has been necessitated by the continuous shrinking of the size of the transients, which (already in the orders of nanometers) are approaching to cross the limit where the laws of quantum mechanics are no longer negligible and the laws of classical physics no longer coherently describe reality. Standard computers rely on the capability to store and process data. They use the states 0 and 1 to control individual bits that hold information in binary form. Instead, the fundamental unit of information in a quantum computer, known as a quantum bit or qubit, is not binary. A qubit can exist not only in a state corresponding to the logical values 0 or 1, as in a classical bit state, but also in a superposition of those classical states. In specifically, three quantum mechanical properties are exploited in quantum computing to modify the state of a qubit:

- *Superposition.* This is one of the fundamental laws of quantum physics that qubits exploit. It refers to a collection of states that we would ordinarily describe separately.

- *Entanglement.* It is a quantum phenomenon that cannot be observed in the classical world, in which entangled particles function as a single system. In more detail, a pair of particles is entangled when each particle's quantum state cannot be characterized independently of the quantum state of the other particle - entanglement applies to both pairs and groups of particles.

- Interference. Finally, interference owing to phase can occur between quantum states, just as it can between two waves. Quantum interference is analogous to wave interference in that when two waves are in phase, their amplitudes increase; when they are out of phase, their amplitudes cancel. The superposition feature of qubits causes interference in the case of quantum interference.

As a result, while traditional computers rely on classical representations of computational memory, quantum computing may convert memory into a superposition of many classical states. While this means that quantum computers give no extra power over classical computers, they do bring greater power in terms of time and size complexity of tackling some issues for which we do not have enough computing capacity to solve. Many of the public-key cryptosystems now employed in contemporary applications as shown in the table 4.1 will be broken by a large-scale quantum computer. This would be disastrous because it would compromise the security and integrity of all digital communications on the Internet. Quantum computers exist now, but they are rough and flawed machines that will require significant technological advancement before they can be used on a large scale. Despite the fact that present experimental quantum computers lack the processing capacity to break any practical cryptographic scheme, several cryptographers are developing new algorithms in anticipation of a time when quantum computing would pose a threat. This is the situation with Shor's algorithm, which was previously discussed. This method, developed by Peter Shor in 1994, is a quantum algorithm capable of factoring integers. This approach was further modified to solve the discrete logarithm problem in a finite field as well as an elliptic curve group. Because those issues give security to the most commonly used public key algorithms - RSA, DSA, and ECDSA, respectively - they become vulnerable to Shor's algorithm.

Cryptographic Algorithm	Type	Purpose	Impact from Large Scale Quantum Computer
AES	Symmetric Key	Encryption	Larger key sizes needed
SHA-2, SHA-3	———	Hash Functions	Larger output needed
RSA	Public Key	Signature, Key ensablishment	No longer secure
ECDSA, ECDH (Elliptical Curve Cryptography)	Public Key	Signature, Key Exchange	No longer secure
DSA (Finite Field Cryptography)	Public Key	Signature, Key Exchange	No longer secure

Table 4.1. Impact of quantum computing on common cryptographic algorithms.

4.2 Approaches for quantum-safe cryptography

Discussions on quantum computers and cryptography often focus around two major aspects of cryptography that are expected to be resistant to assaults by large and powerful quantum computers: quantum key distribution and Post-Quantum Cryptography.

4.3 Quantum Key Distribution

Quantum Key Distribution (QKD) refers to quantum protocols that use quantum and classical channels (e.g., optical fibers and wireless channels) to co-create private symmetric keys between two parties by codifying private key bits into quantum states. If any eavesdropper intercepts and observes these quantum states, the information they contain (i.e., the bits of the key) is transformed, causing the key to be corrupted and the eavesdropper to be identified. However, because to the deterioration of the quantum states storing the keys, ground-based key exchanges utilizing optical fibers are restricted to a few hundreds of kilometers nowadays. The scalability of these networks is dependent on the development of quantum repeaters, which necessitate the use of extremely powerful quantum memory. This is currently a work in progress. For these reasons, QKD has been ruled out as a viable option for providing quantum safety to blockchain networks today. However, given the National Security Agency (NSA), NIST, and the European Telecommunications Standards Institute (ETSI), among others, have announced that quantum cryptography (such as QKD) would be the sole option for long-term safe encryption, this may change in the future. [16]

4.4 Post-quantum cryptography

Post-Quantum Cryptography (PQC) is a new field of encryption built to be secure against quantum attackers. [17] As a result, the objective of PQC is to create cryptographic systems that are safe against both quantum and conventional computers while also interoperating with existing communications protocols and networks. The necessity for building cryptosystems whose security is based on several hard mathematical problems that cannot be solved by a large-scale quantum computer is genuine. Despite the fact that present experimental quantum computers lack the processing capacity to break any practical cryptographic scheme, several cryptographers are developing new algorithms in anticipation of a time when quantum computing would pose a threat. Another reason to begin designing new cryptographic algorithms is because we don't know when today's traditional cryptography may fail, and it's difficult and time-consuming to withdraw and replace old encryption from production applications. [18]

4.5 PQC Algorithms

Encrypt, decrypt, sign, and verify are the four essential processes in cryptography that must be completed to ensure the security of transactions and data. Only authorized parties can view the plaintext communication once it has been decrypted and encrypted. The encrypted communication cannot be read without the necessary cryptographic key. This implies that even if an attacker intercepts the communication, he will be unable to read it. The use of digital signatures and verification ensures that the sender is truthful and that

the message was not altered during transmission. Because only the genuine sender may generate a digital signature with his or her private key, the digital signature verifies the message author's validity. The objective of PQC designers is to improve the efficiency and usability of the new PQC algorithms' operations. First and foremost, a PQC algorithm must be designed to be resistant to both quantum and conventional computers. There are various kinds of mathematical problems that are thought to be resistant to quantum computer assaults and have been utilized to build public key cryptosystems. The unique class of mathematical issues exploited by the individual method can be used to classify PQC techniques. Of all the various approaches that exist here, we have reported only those related to the algorithms studied in the thesis:

- *Lattice-based cryptography.* Their security is based on the notion that certain lattice-related optimization problems cannot be solved efficiently: these are known as computational lattice problems.
- *Code-based cryptography.* Error-correcting codes are used in these cryptographic systems. The McEliece public key encryption technique, for example, is predicated on the difficulty of decoding a generic linear code.
- *Hash-based cryptography.* It suggests using hash functions to digitally sign documents. Hash-based systems are fully dependent on the features of ordinary hash functions. As a result, they are thought to be among the most quantum-resistant. XMSS and SPHINCS are two instances of hash-based algorithms.

These new cryptosystems, based on these PQC algorithms, must communicate with various communication protocols and networks. The issue is that present post-quantum systems have a number of restrictions. All post-quantum techniques have either larger public keys, longer ciphertexts or signatures, or slower runtime as compared to standard RSA, DSA, and ECDSA methods. To interoperate with diverse protocols, we must create better signature and public key encryption techniques that can work with fewer keys and ciphertexts or signatures. Furthermore, many PQC techniques are based on mathematical issues that are relatively novel in terms of cryptography, hence they have undergone less cryptanalysis.

The primary goal of PQC research is to make those schemes usable and flexible while also analyzing and approving them by security experts in order to develop fast and secure implementations for both high-performance servers and small embedded devices, as well as to integrate those schemes into existing network infrastructure and applications. Unfortunately, as is clearly visible in the table 4.1 several of these algorithms still have technical shortcomings when compared to current cryptographic methods, which may limit their practical utility.

PQC techniques, for example, frequently need more computer resources than contemporary cryptographic algorithms. This implies they may be slower or less efficient in terms of time and resources than standard cryptography systems. Moreover, some PQC algorithms need the development and distribution of extremely large public and private keys, which may pose a performance and data size problem to present security methods.

4.5.1 Lattice-based

Table 4.1 shows the key and signature sizes for all NIST Round 4 PQ standardization candidates. The most of PQ signature techniques increase the length of keys and signatures

		Size (bytes)		CPU time (lower is better)	
	PQ	Public key	Signature	Signing	Verification
Ed25519	❌	32	64	1 (baseline)	1 (baseline)
RSA-2048	❌	256	256	70	0.3
Dilithium2	✅	1,312	2,420	4.8	0.5
Falcon512	✅	897	666	8*	0.5
SPHINCS⁺128s	✅	32	7,856	8,000	2.8
SPHINCS⁺128f	✅	32	17,088	550	7

Figure 4.1. Performance characteristics of NIST's chosen signature schemes compared to Ed25519 and RSA-2048. Source: [19]

used in protocols dramatically. The most promising are the lattice-based ones, which may potentially be considered for direct replacement of the currently utilized systems. The majority of PQ signature approaches significantly lengthen the keys and signatures used in protocols. The most promising are the lattice-based ones, which might be used as a straight replacement for the present systems. Dilithium and Falcon are the NIST PQ signature contenders that use lattice-based encryption. [20]

Dilithium and Falcon

All of these schemes use substantially bigger signatures than those currently in use. Falcon in the scientific community is recognized as the best performing algorithm. It does, however, have a flaw that this chart 4.1 does not highlight: it requires rapid constant-time double-precision floating-point arithmetic to achieve acceptable signature performance. Let's dissect that. Constant time indicates that the length of the operation is independent of the data handled. If the time it takes to make a signature is dependent on the private key, the private key may frequently be retrieved by measuring the time it takes to create a signature. Writing constant-time code is difficult, but cryptographers have figured it out for integer arithmetic throughout the years. Falcon is the first major cryptographic method to employ double-precision floating-point arithmetic. It wasn't apparent whether Falcon could be implemented in constant-time at first, but Falcon was brilliantly implemented in constant-time for multiple distinct CPUs, requiring various innovative workarounds for particular CPU instructions. Despite this accomplishment, Falcon's timelessness is based on fragile foundations. The next generation of Intel CPUs may have an optimization that disrupts Falcon's constant-time behavior.

This gets us to Dilithium. When compared to Falcon, it is easier to install and has greater signature performance. However, its signatures and public keys are substantially bigger, which is a concern. For example, we transmitted six signatures and two public keys to each browser that visited this website. We'd be looking at an extra 17kB of data if we replaced them all with Dilithium2. Thus, as shown in the figure 4.3 simply upgrading to Dilithium may quadruple your Transport Layer Security (TLS) handshake times. [19]

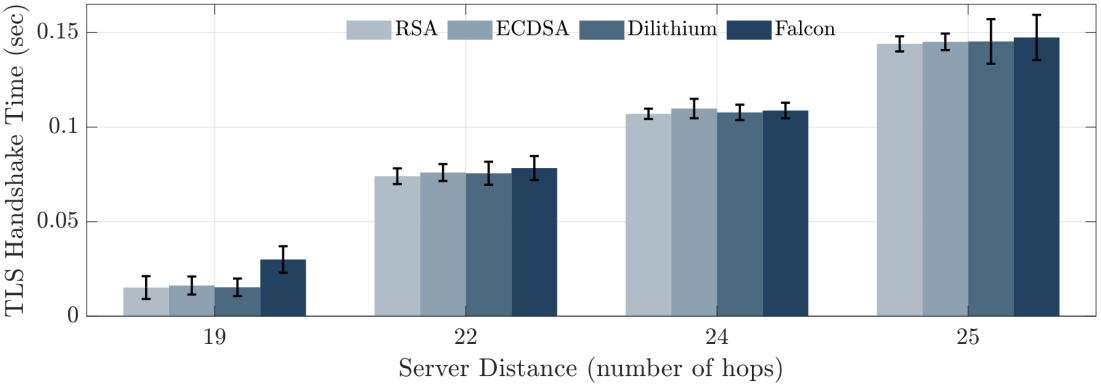

Figure 4.2. TLS 1.3 Handshake time per client-server distance using classic vs NIST Level 1 Dilithium & Falcon certificates. Source: [20]

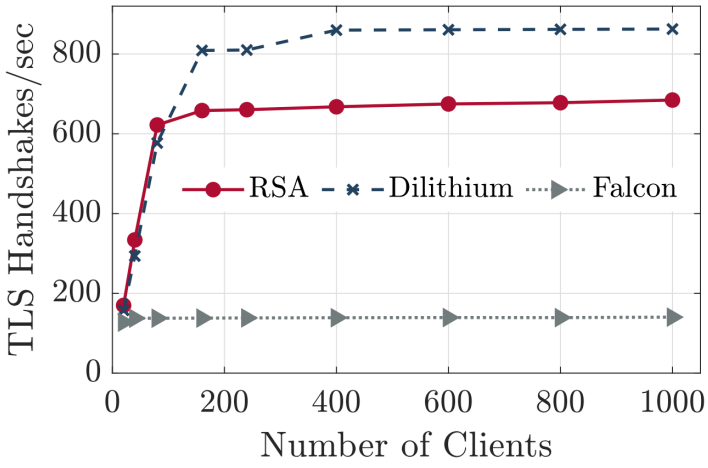

Figure 4.3. Handshakes/s per total clients. Source: [20]

4.6 NIST PQC Project

The urge to compete against the eventual prospect of a large-scale quantum computer is driving an increase in interest in implementing PQC techniques. Recent advances in quantum computing have caused huge security concerns among IT experts. The capacity of a quantum computer to tackle integer factorization and discrete logarithm issues effectively poses a danger to current encryption, key encapsulation mechanism, and digital signature techniques.The Key Encapsulation Mechanism (KEM) is a cryptographic mechanism for producing and distributing a secret key with the use of a public key. In practice, a KEM enables two parties to produce a secret shared session key without knowing one other's private key. This is achieved through the use of an asymmetric encryption technique in which one party uses its public key to encrypt a randomly generated secret key and the other party uses its private key to decrypt the secret key. The created secret key can then be used for symmetric encryption of data sent between the two parties. A digital signature, on the other hand, is a technique of validating the validity of a digital communication or document, assuring that the sender is who he or she claims to be and

that the message's content has not been altered since it was signed. This is done through the use of a digital signature technique, in which the sender generates a message signature using his or her own private key, which is then transmitted along with the message. The recipient can then use the sender's public key to validate the message's signature and validity. The National Institute of Standards and Technology (NIST) in the United States launched a public project to standardize post-quantum public key encapsulation and signature mechanisms, while the European Telecommunications Standards Institute (ETSI) formed a Quantum-Safe Working Group to make proposals for real-world implementation of those algorithms.

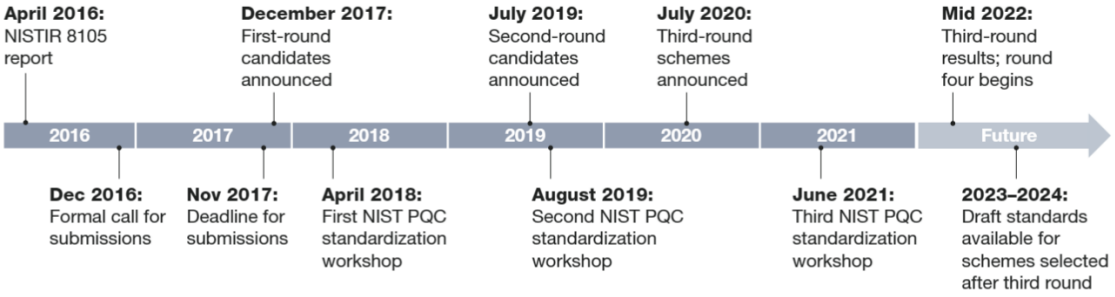

Figure 4.4. The notable events during the course of the NIST PQC standardization process are shown, from its inception in 2016 to the present day. Source: [21]

In 2016, NIST issued an open request for quantum-resistant encryption algorithms. NIST has identified four candidate algorithms for standardization after thorough consideration during the third phase of the NIST PQC Standardization Process. For the majority of use cases, NIST will suggest two principal algorithms for implementation: CRYSTALS-KYBER (key-establishment) and CRYSTALS-Dilithium (digital signatures). Moreover, the FALCON and SPHINCS+ signature schemes will be standardized.

Public-Key Encryption/Kem	**Digital Signature**
CRYSTALS-Kyber	CRYSTALS-Dilithium
	Falcon
	SPHINCS+

Table 4.2. Algorithms to be standardized

Even though such algorithms will be regarded solid in the future when the standardization process is completed, the acceptance of PQC will also be dependent on the success of the transfer of communication protocols and applications to embrace these new algorithms. The fundamental issue is that implementing such schemes in Public Key Infrastructure (PKI) protocols and use-cases might be difficult for today's Internet due to the key size cost and substantial delay associated with these schemes' high computing performance.

4.7 PQC Transition

There is an approaching requirement for PQC deployment. This section will concentrate on the primary motivations for conducting research on PQC methods. Figure 4.4 displays a timeline of upcoming major PQC-related events. This timeline is not to scale and is

made up of three concurrent sequences of events. The top red timeline in Fig. 4.4 depicts the two most significant quantum risks and when they become serious. The first type of attack, known as a Store-Now-Decrypt-Later (SNDL) attack, is already in use. It is analogous to attackers acquiring important encrypted information today, keeping it, then decrypting it later when LFT quantum computers become accessible.

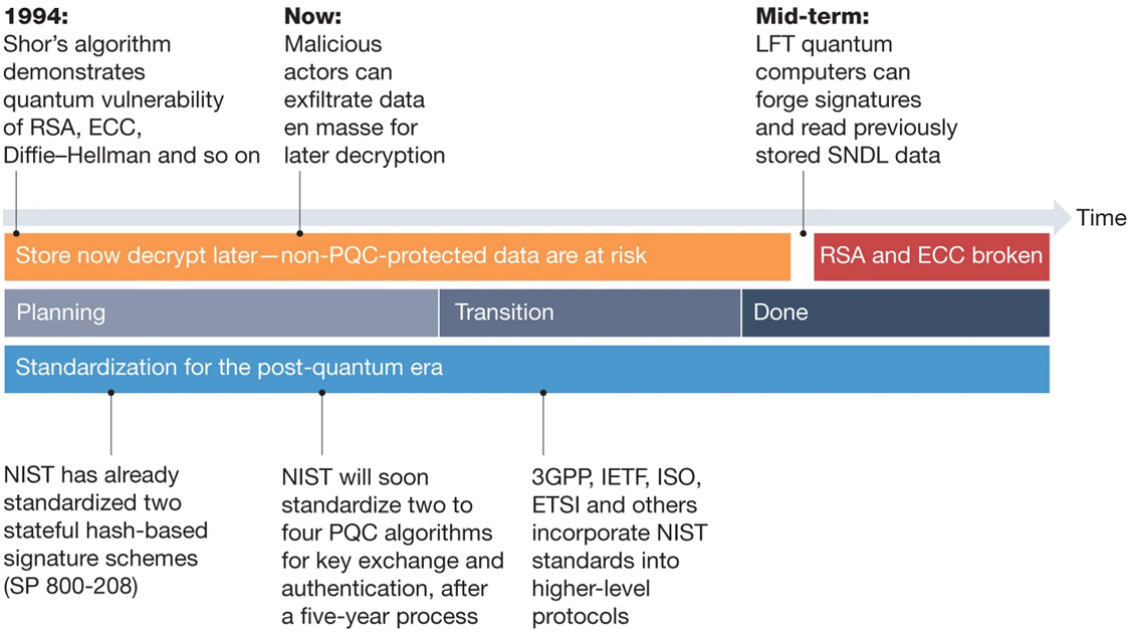

Figure 4.5. The three timelines can be thought of as: the threat to cryptography (top), the steps organizations should pass through during the migration (middle) and the process of standardization (bottom), which is led by multinational standards bodies. Source: [21]

The SNDL attack presumes that this knowledge will be useful in the future. The capacity to break RSA and ECC, the two most widely used public key techniques for encrypting information today, is referred to be the second quantum menace. Adversaries would be able to fake RSA and ECC digital signatures, posing hazards to systems that rely on them, such as secure web browsing, zero trust architectures, and cryptocurrencies. The two acts required by organizations in migrating to PQC are depicted in the middle grey timeline in Fig. 4.4. The first is concerned with strategic planning and technical experimentation for this transition, while the second is concerned with the practical implementation of PQC in production systems. We underline that the strategic planning phase must be accomplished far before LFT quantum computers can attack RSA and ECC successfully (that is, a process whose start should not be further delayed).

Finally, the bottom blue timeline in Fig. 4.4 addresses standardization processes organized by relevant government and industrial bodies, with a particular emphasis on the NIST PQC process to determine the fundamental security of proposed PQC candidates.

4.8 Open Quantum Safe project

The Open Quantum Secure (OQS) project is a collaborative effort to create and assist the incorporation of quantum-resistant cryptographic algorithms into current protocols and applications. The project's goal is to assist companies and people in preparing for the arrival of quantum computers, which are projected to severely weaken the security of many

present cryptographic systems. The OQS project is led by the Institute for Quantum Computing at the University of Waterloo and is funded by a number of corporations, including Microsoft, Cloudflare, and Cisco. The main team of the project consists of expertise in encryption, quantum computing, and software development. Many post-quantum cryptographic methods, including lattice-based, code-based, and multivariate-based cryptography, have been created by the OQS project. These algorithms are being developed to withstand assaults from both conventional and quantum computers. The Open Quantum Safe (OQS) project intends to create and test quantum-resistant encryption by including post-quantum techniques into the liboqs library. Douglas Stebila and Michele Mosca created the open source C library liboqs. It contains implementations of PQC algorithms to help with deployment and testing in real-world contexts. The library is implemented using a standard interface based on NIST submission package implementations. Liboqs began with only key exchange algorithms and eventually evolved to include signature methods. The project's source code is organized in a simple and modular manner, making it easier to comprehend and develop. This section, presented in a hierarchical layout, offers an overview of the OQS source code structure. The *'src/'* subdirectory contains the project's source code, and is organized into subdirectories based on the different cryptographic algorithms implemented by the project. The *'algorithms/'* subdirectory contains subdirectories for different families of algorithms, such as key encapsulation mechanisms (*'kem/'*) and signature schemes (*'sig/'*). The *'tests/'* subdirectory contains tests for the project, organized into unit tests and integration tests. The *'examples/'* subdirectory contains example code demonstrating how to use the project's algorithms in different programming languages, such as C, C++, and Java. The *'scripts/'* subdirectory contains various scripts used to run benchmarks and other tasks.

The OQS also provides benchmarking data for multiple quantum-resistant algorithms, which are utilized in this work to compare the runtime behavior and memory usage of the algorithms. The runtime behavior and memory consumption statistics of the algorithms are obtained using Amazon Web Service (AWS) and the CPU Model Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50 GHz [22]:

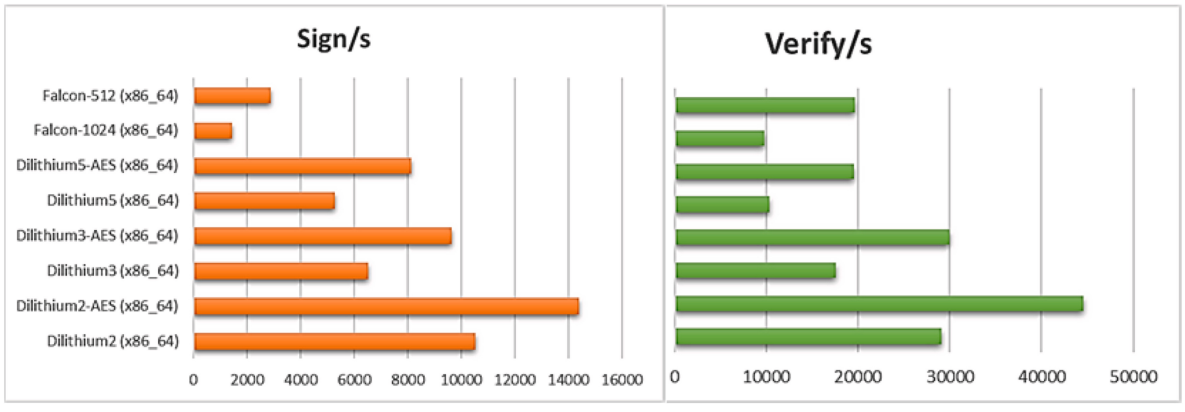

Figure 4.6. Sign and Verify operations per second per algorithm (Digital Signature using Lattice Based Algorithms). Source: [22]

- *Dilithium*: Dilithium asserts that it has the shortest public key and signature size of any lattice-based signature system that just uses uniform sampling.
  - Lattice-based digital signature algorithm based on the Fiat-Shamir paradigm.

  - The security is based on decisional Module Learning With Errors (MLWE) assumption, which suffices to show that the public key does not leak any information about the secret key.
  - It uses pseudorandomness and truncated storage techniques improve the performance. The scheme does not use floating-point arithmetic, which is an advantage.
  - Highly efficient and relatively simple in implementation.

- *Falcon*: The security of Falcon relies on the difficulty of finding a small non-zero vector in the lattice.
  - Lattice-based signature scheme utilizing the "hash-and-sign" paradigm.
  - The theoretical security is based on the proof of unforgeability in the QROM, based on the hardness of the SIS Problem over NTRU lattices.
  - The verification process is fast and requires low bandwidth. It is the best choice for some constrained protocol scenarios.
  - NIST has confidence in its security. Because of its low bandwidth, it is a preferred choice for certain applications.

Chapter 5

Hyperledger Technology

Hyperledger is an open source blockchain platform for developing corporate blockchain applications. Hyperledger is a set of tools and libraries developed by the Linux Foundation for establishing private and secure blockchain networks suited for usage in the corporate setting. Hyperledger is becoming increasingly popular among businesses wanting to use blockchain solutions for data and transaction management due to its flexibility and ability to adapt to varied demands. This initiative aims to develop a system that might be used in areas such as finance, banking, the Internet of Things (IoT), and supply chains, among others. This chapter will go through the characteristics of Hyperledger, its primary components, and how they may be utilized to build customized blockchain applications.

5.1 Hyperledger for Blockchain Applications

The primary purpose of Hyperledger was not to construct a blockchain with a currency and push it into the crypto market. Its true purpose is to create technologies that harness the structure that gave birth to Bitcoin, attempt to decouple the money, and use the blockchain method of operation in other scenarios. This was marketed as a chance to try to increase the efficiency of businesses and industries in various scenarios. In other words, Hyperledger was a blockchain initiative that served general needs rather than a specific use case. One of the primary applications for which Hyperledger is presented as a technology is to provide more secure solutions for financial and supply chain systems. Therefore, the companies behind the project invested in research and development and succeeded in creating a framework capable of meeting these needs. Today, Hyperledger is one of the most extensively used blockchain systems for addressing these demands. Not only that, but it does so in a setting that can be public (as with any blockchain), semi-public, or private, illustrating the platform's adaptability to satisfy a variety of demands. Many projects have been implemented to achieve this feat; we choose to focus our attention on IBM's fabric framework.

5.2 Fabric

The Hyperledger Fabric project, developed by IT giant IBM, is one of the most prominent implementations of the Hyperledger platform. Fabric is a blockchain technology that is modular and adaptable to the demands of corporate use cases. Fabric's key features include code modularity, which enables for the creation of custom solutions and

the addition of new functionality; scalability, which allows for the handling of large volumes of transactions; and security, which is provided by a certificate-based access system and a customizable consensus mechanism. Fabric also includes a collection of standard components that may be used to build private and secure blockchain networks, such as a distributed ledger, a smart contract system, and an identity management system. Fabric is intended to be a highly customizable and scalable platform with a wide range of access and authorization options. All of these characteristics make it ideal for dealing with high data traffic conditions, intensive and broad use of information access, and, most significantly, adjusting to specific developments required by industry in its many areas.

Among the properties of Hyperledger Fabric we can mention[1]:

- Its permission architecture works through unique cryptographic permissions. Each permission can have granular access to blockchain data, maintaining access to different parts of it in a controlled manner.[1]
- It uses a modular working architecture, so its capabilities can be adapted or expanded as needed.[1]
- It can use different consensus protocols, adapting them for security, speed or privacy as required.[1]
- It can run smart contracts, which can be written in programming languages such as Go, Java, JS and can also run an Ethereum Virtual Machine (EVM) file and run smart contracts written in Solidity, acting as a bridge between Fabric and Ethereum development.[1]
- High-speed peer network due to its specially designed gossip protocol for low latency in P2P networks.[1]
- High fault tolerance and interchangeable BFT protocols.[1]

5.3 Order-Execute Architecture for Blockchains

All previous blockchain systems, permissioned or not, follow the order-execute architecture. This means that the blockchain network organizes transactions first, using a consensus process, and then sequentially executes them on all peers. For example, a permissionless blockchain based on PoW, such as Ethereum, combines consensus and transaction execution as follows:

- every peer (i.e., a node that participates in consensus) assembles a block containing valid transactions (to establish validity, this peer already pre-executes those transactions);
- the peer tries to solve a PoW puzzle;
- if the peer is lucky and solves the puzzle, it disseminates the block to the network via a gossip protocol;

[1]Jose Segura, What is Hyperledger?, Nov 19, 2020, https://academy.bit2me.com/it/que-es-hyperledger

- every peer receiving the block validates the solution to the puzzle and all transactions in the block. Effectively, every peer thereby repeats the execution of the lucky peer from its first step. Moreover, all peers execute the transactions sequentially (within one block and across blocks).

Figure 5.1. The traditional architecture for building blockchains and resilient replicated services, where transactions are ordered first and executed later. Source: [23]

5.3.1 Limitations of Order-Execute

Because the order-execute architecture is conceptually simple, it is also extensively employed. However, when employed in a general-purpose permissioned blockchain, it has several downsides. The three most important ones are discussed next.

Sequential execution

The blockchain's effective throughput is limited because transactions must be executed sequentially on all peers. Because throughput is inversely related to execution latency, this could constitute a performance constraint for all but the most basic smart contracts. A denial-of-service (DoS) attack, which significantly lowers the performance of such a blockchain, might simply introduce smart contracts that take an inordinate amount of time to complete. A smart contract that executes an infinite loop, for example, has a deadly effect that cannot be identified automatically since the halting problem is intractable. The literature on distributed systems provides numerous methods for improving efficiency over sequential processing, such as simultaneous execution of unrelated processes. Sadly, similar strategies have yet to be properly utilized in the blockchain context of smart contracts. For instance, one challenge is the requirement for deterministically inferring all dependencies across smart contracts, which is particularly challenging when combined with possible confidentiality constraints [24]. Furthermore, these techniques are of no help against DoS attacks by contract code from untrusted developers.

Non-deterministic code

Non-deterministic transactions are another significant issue for an order-execute architecture. This is commonly handled by writing blockchains in domain-specific languages (for example, Ethereum Solidity), which are expressive enough for their purposes but constrained to deterministic execution. Such languages, however, are difficult to create for the implementer and necessitate additional programming knowledge. Instead, writing smart contracts in a general-purpose language (e.g., Go, Java, C/C++) is more appealing and speeds up the adoption of blockchain solutions. Unfortunately, generic languages provide numerous challenges to ensuring deterministic execution. Even if the application

developer does not explicitly include non-deterministic activities, hidden implementation details can have the same disastrous consequence (e.g., a map iterator is not deterministic in Go).

Confidentiality of execution

Many permissioned systems, according to the public blockchain plan, run all smart contracts on all peers. Many of the proposed use cases for permissioned blockchains, however, require confidentiality, which means that access to smart-contract logic, transaction data, or ledger state can be restricted. Although cryptographic approaches such as data encryption, advanced zero-knowledge proofs, and verifiable computation can help establish confidentiality, they often come at a high cost and are not practical in practice. Fortunately, instead of running the same code everywhere, it is sufficient to propagate the same state to all peers. As a result, the execution of a smart contract can be limited to a subset of peers who are trusted for this task and can attest for the results of the execution. This design deviates from the norm [24].

5.4 Execute-Order-Validate Fabric's Architecture

Fabric is a novel blockchain architecture that aims for resilience, adaptability, scalability, and confidentiality. Fabric is the first blockchain technology to facilitate the execution of distributed applications written in common programming languages, allowing them to be executed uniformly across numerous nodes, providing the impression of execution on a single globally distributed blockchain computer. As a result, Fabric is the world's first distributed operating system for permissioned blockchains. Fabric is built on a groundbreaking execute-order-validate paradigm for distributed execution of untrusted programs in an untrusted environment. It separates the transaction flow into three steps, which may be run on different entities in the system [24]:

- *executing a transaction* and checking its correctness, thereby endorsing it (corresponding to "transaction validation" in other blockchains);
- *ordering* through a consensus protocol, irrespective of transaction semantics;
- *transaction validation* per application specific trust assumptions, which also prevents race conditions due to concurrency.

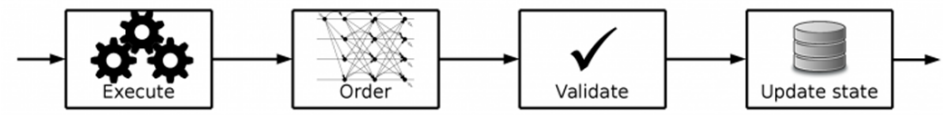

Figure 5.2. The transaction flow in Hyperledger Fabric, in which transactions are executed and endorsed first, before ordering them and validating that they do not conflict. Source: [23]

5.4.1 Fabric Architecture

Fabric is a permissioned blockchain distributed operating system that runs distributed applications written in general-purpose programming languages (e.g., Go, Java, Node.js). It secures its execution history in an append-only replicated ledger data structure and does not include crypto-currency. Fabric introduces the execute-order-validate blockchain architecture, which differs from the traditional order-execute approach. In a nutshell, a distributed application for Fabric consists of two parts:

- A *smart contract*, called chaincode, which is program code that implements the application logic and runs during the execution phase. The chaincode is the central part of a distributed application in Fabric and may be written by an untrusted developer. Special chaincodes exist for managing the blockchain system and maintaining parameters, collectively called system chaincodes
- An *endorsement policy* that is evaluated in the validation phase. Endorsement policies cannot be chosen or modified by untrusted application developers. An endorsement policy acts as a static library for transaction validation in Fabric, which can merely be parameterized by the chaincode. Only designated administrators may have a permission to modify endorsement policies through system management functions. A typical endorsement policy lets the chaincode specify the endorsers for a transaction in the form of a set of peers that are necessary for endorsement; it uses a monotone logical expression on sets, such as "three out of five" or "$(AB) \vee C$." Custom endorsement policies may implement arbitrary logic.

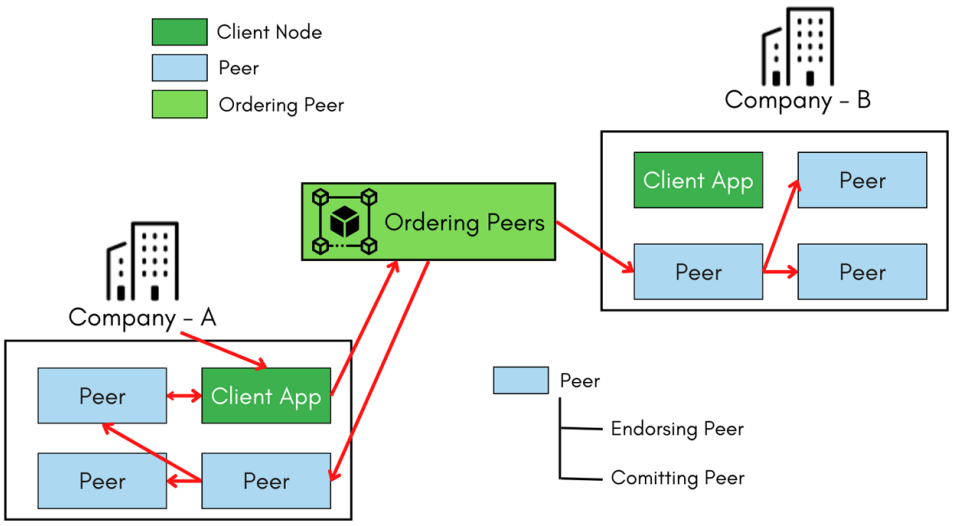

Figure 5.3. Fabric Network

Transactions are sent by a client to the peers designated by the endorsement policy. Each transaction is then conducted by particular peers, and the results are recorded; this is also known as endorsement. Following execution, transactions enter the ordering phase, which employs a pluggable consensus process to provide a completely ordered series of endorsed transactions organized into blocks. These are sent to all peers, with the (optional) assistance of gossip. In the validation phase, each peer checks the state changes from endorsed transactions in relation to the endorsement policy and the consistency of

the execution. The transactions are validated in the same sequence by all peers, and the validation is deterministic. A Fabric blockchain, as we see from the figure 5.3 is made up of nodes that create a network. Because Fabric is permissioned, all nodes in the network have an identity given by a modular membership service provider (MSP). Nodes in a Fabric network perform one of three functions [24]:

- *Clients* submit transaction proposals for execution, help orchestrate the execution phase, and, finally, broadcast transactions for ordering.
- *Peers* execute transaction proposals and validate transactions. All peers maintain the blockchain ledger, an append-only data structure recording all transactions in the form of a hash chain, as well as the state, a succinct representation of the latest ledger state. Not all peers execute all transaction proposals, only a subset of them called endorsing peers (or, simply, endorsers) does, as specified by the policy of the chaincode to which the transaction pertains.
- *Ordering Service Nodes (OSN)* (or, simply, orderers) are the nodes that collectively form the ordering service. In short, the ordering service establishes the total order of all transactions in Fabric, where each transaction contains state updates and dependencies computed during the execution phase, along with cryptographic signatures of the endorsing peers. Orderers are entirely unaware of the application state, and do not participate in the execution nor in the validation of transactions. This design choice renders consensus in Fabric as modular as possible and simplifies replacement of consensus protocols in Fabric.

5.4.2 Membership Service Provider

Hyperledger Fabric is always picked to establish a consortium network to solve a business challenge. Participants are often corporate entities. Organizations are used by Hyperledger Fabric to represent these participating entities. Each organization in Fabric is distinguished by its Membership Service Provider (MSP). On the one hand, MSP establishes credentials for all entities inside an organization. MSP, on the other hand, symbolizes the organization that will engage in a consortium network. Technically, the consortium network is made up of all organizations' MSPs. MSP is used in PKI. In a normal PKI system, each entity has a private key as well as a digital certificate (certificate) that contains the public key as well as important information about the entity. A Certificate Authority (CA) signs and issues this certificate, which serves as the entity's identification. When an entity uses its private key to sign a message, it creates a signature on that message. Everyone who obtain this message, signature and the signer's certificate, can[2]

- indicate that this signature can only be created by the owner of the private key (through public key from certificate)[2]
- determine the identity of the private key holder (through information recorded in the certificate)[2]

[2]KC Tam, Two Ways to Generate Crypto Materials in Hyperledger Fabric: Cryptogen and CA Server, May 13, 2020, https://kctheservant.medium.com/two-ways-to-generate-crypto-materials-in-hyperledger-fabric-cryptogen-and-ca-server-36d3c3e2daad

- trust that the signer is who he claims to be (by checking the CA's signature on the certificate, with the premise that the CA's certificate is trusted).[2]

This signing and verification procedure occurs throughout a consortium network. Each organization in a consortium network knows the MSP of the other organizations, and confidence in this consortium network is developed from a "who's who" standpoint.

5.4.3 Execution Phase

Clients sign and transmit the transaction proposal to one or more endorsers for execution during the execution phase. Remember that every chaincode provides a set of endorsers implicitly via the endorsement policy. A proposal includes the submitting client's identity (as determined by the MSP), the transaction payload in the form of an operation to be executed, parameters, and the chaincode's identifier, a nonce to be used only once by each client, and a transaction identifier derived from the client identifier and the nonce. The proposal is simulated by the endorsers by running the operation on the provided chaincode, which is placed on the blockchain. The chaincode is executed in a Docker container that is separate from the main endorser process. Without synchronization with other peers, a proposal is simulated against the endorser's local blockchain state. Furthermore, endorsers do not save the simulation findings to the ledger. The peer transaction manager (PTM) maintains the state of the blockchain in the form of a versioned key-value store, in which subsequent modifications to a key have monotonically increasing version numbers. [24]. As a result of the simulation, each endorser generates a value writeset consisting of the simulation's state updates (i.e., the modified keys along with their new values), as well as a readset representing the proposal simulation's version dependencies (i.e., all keys read during simulation along with their version numbers). Following the simulation, the endorser cryptographically signs a message called endorsement, which comprises readset and writeset (together with metadata such as transaction ID, endorser ID, and endorser signature), and returns it to the client in the form of a proposal response. The client gathers endorsements until they meet the chaincode's endorsement policy, which the transaction invokes. This, in particular, necessitates that all endorsers, as stated by the policy, provide the same execution outcome (i.e., identical readset and writeset). The client then creates the transaction and sends it to the ordering service. The whole process is summarized in the figure 5.4.

5.4.4 Ordering Phase

When a client has enough approvals for a proposal, it creates a transaction and presents it to the ordering service. The transaction payload (i.e., the chaincode operation with parameters), transaction information, and a set of endorsements are all part of the transaction. During the ordering step, a total order is established on all submitted transactions per channel. In other words, despite flawed orderers, ordering atomically broadcasts endorsements and so achieves agreement on transactions. In addition, the ordering service groups numerous transactions into blocks and returns a hash-chained sequence of blocks containing transactions. Grouping or batching transactions into blocks enhances broadcast protocol performance, which is a well-known approach used in fault-tolerant broadcasts. The ordering service guarantees that all blocks sent on a single channel are completely ordered. More particular, ordering assures that each channel has the following safety properties [24]:

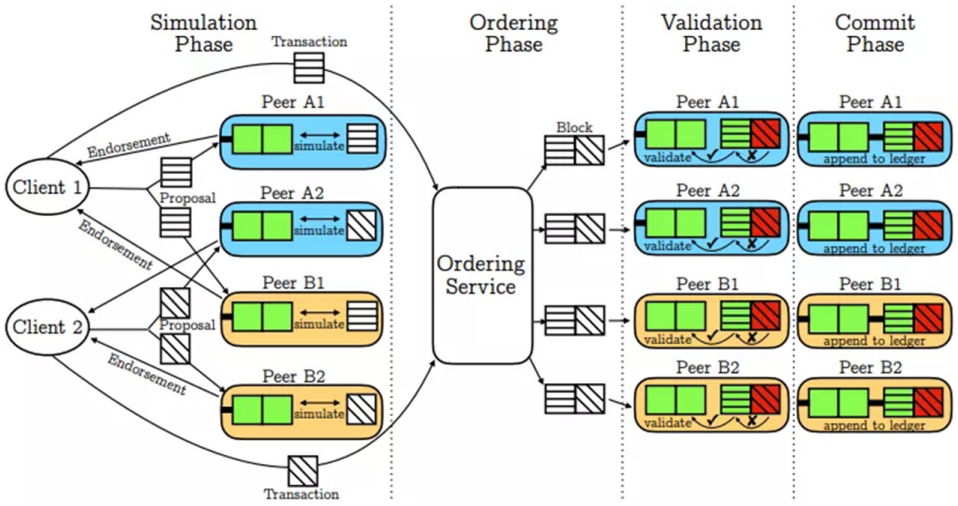

Figure 5.4. High-level workflow of Fabric. Source: [25]

- *Agreement*: For any two blocks B delivered with sequence numbers and B' delivered with s' at correct peers such that s = s', it holds B = B'.
- *Hash chain integrity*: If some correct peer delivers a block B with number s and another correct peer delivers block $B' = ([tx1, ..., txk], h')$ with numbers+1, then it holds h' = H(B), where H(·) denotes the cryptographic hash function.
- *No skipping*: If a correct peer p delivers a block with $numbers > 0$ then for each $i = 0, ..., s - 1$, peer p has already delivered a block with number i.
- *No creation:* When a correct peer delivers block B with number s, then for every $tx \in$ B some client has already broadcast tx.

5.4.5 Validation Phase

Blocks are distributed to peers directly through the ordering service or via rumor. The validation phase, which consists of three successive phases, is subsequently entered by a new block [24]:

- The *endorsement policy evaluation* occurs in parallel for all transactions within the block. The evaluation is the task of the so-called validation system chaincode (VSCC), a static library that is part of the blockchain's configuration and is responsible for validating the endorsement with respect to the endorsement policy configured for the chaincode. If the endorsement is not satisfied, the transaction is marked as invalid and its effects are disregarded.
- A *read-write conflict* check is done for all transactions in the block sequentially. For each transaction it compares the versions of the keys in the *readset* field to those in the current state of the ledger, as stored locally by the peer, and ensures they are still the same. If the versions do not match, the transaction is marked as invalid and its effects are disregarded.
- The *ledger update phase* runs last, in which the block is appended to the locally stored ledger and the blockchain state is updated. In particular, when adding the

block to the ledger, the results of the validity checks in the first two steps are persisted as well, in the form of a bit mask denoting the transactions that are valid within the block. This facilitates the reconstruction of the state at a later time. Furthermore, all state updates are applied by writing all key-value pairs in *writeset* to the local state.

Fabric's default VSCC permits monotone logical expressions across the collection of endorsers selected for expressing a chaincode. The VSCC evaluation confirms that the collection of peers, as indicated by valid signatures on transaction endorsements, meet the expression. However, different VSCC regulations can be defined statically.

5.5 Chaincode

Hyperledger Fabric is a blockchain platform built for enterprise applications. It provides a secure, efficient, and adaptable architecture for executing network transactions. Chaincode, a smart contract that defines the business logic for network transactions, is a key component of Hyperledger Fabric. Chaincode is a program written in a high-level language like Go or JavaScript that defines the business logic for network transactions. It is in charge of performing transactions, accessing the ledger state, and changing it as needed. Chaincode is deployed on the network by network administrators and is executed when a transaction is submitted by endorsing peers. Chaincode has various advantages for the network. It enables the secure and efficient implementation of complicated business logic, as well as the execution of transactions in a regulated and secure environment. Furthermore, chaincode offers a great level of versatility because it may be implemented in a variety of high-level languages and simply changed as needed.

Users of Hyperledger Fabric frequently interchange the words smart contract and chaincode. A smart contract, in general, describes the transaction logic that governs the lifecycle of a business entity stored in the world state. It is then packaged as chaincode and published to a blockchain network. Consider smart contracts to be the rules that control transactions, whereas chaincode governs how smart contracts are packaged for deployment.

A smart contract, as shown in the diagram 5.5, is a domain-specific software that relates to certain business operations, whereas a chaincode is a technical container for a group of connected smart contracts.

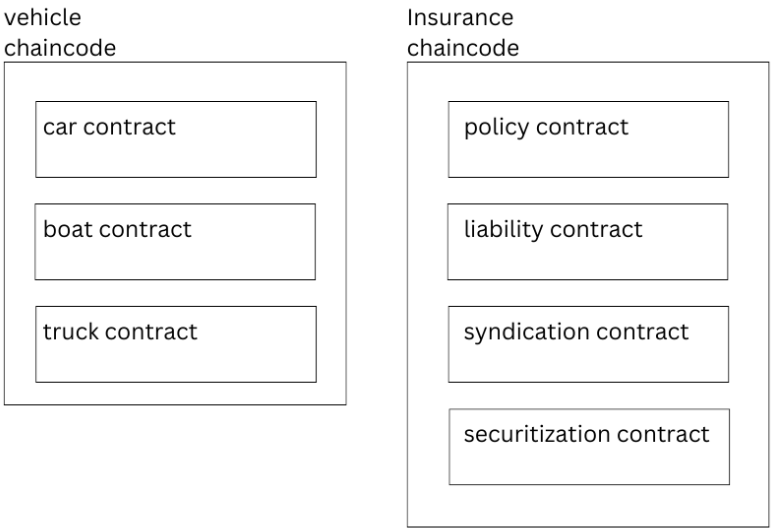

Figure 5.5. A chaincode contains the definition of a smart contract. Within the same chaincode, multiple smart contracts can be defined. All smart contracts included within a chaincode are made available to applications when it is deployed. Source: [26]

5.5.1 Ledger

At its most basic, a blockchain immutably records transactions that update ledger states. A smart contract programmatically accesses two independent parts of the ledger: a blockchain, which immutably records the history of all transactions, and a world state, which maintains a cache of the current value of these states, as it is typically the present value of an object that is required.

Smart contracts primarily put, get, and remove states from the global state, but they can also query the immutable blockchain transaction record.

- A *get* often indicates a query to retrieve information about a business object's current status.
- In the ledger world state, a *put* often creates a new business object or alters an existing one.
- A *delete* often denotes the deletion of a business object from the ledger's current state, but not its history.

Many APIs are available to smart contracts. Importantly, whether transactions create, read, update, or destroy business objects in the global state, the blockchain keeps an immutable record of these changes.

5.5.2 Endorsement

An endorsement policy is associated with each chaincode and applies to all of the smart contracts established within it. A highly significant endorsement policy specifies which entities in a blockchain network must sign a transaction generated by a certain smart contract in order for that transaction to be declared valid.

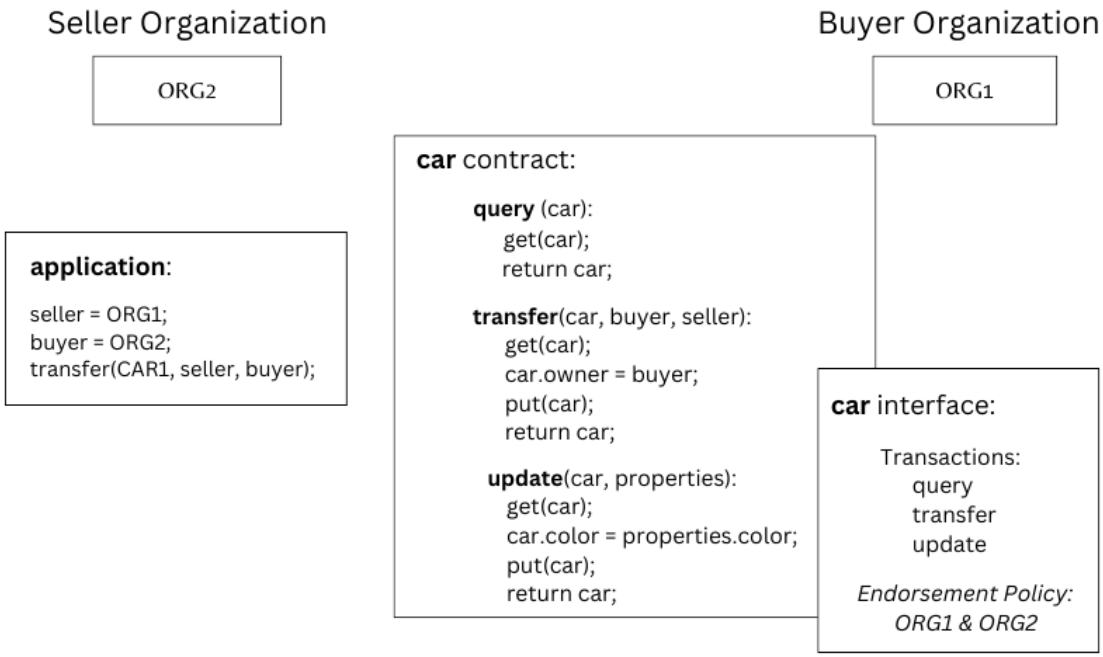

Figure 5.6. Every smart contract comes with an endorsement policy. This endorsement policy specifies which entities must approve smart contract-generated transactions before they may be identified as valid. Source: [26]

An endorsement policy can state that a transaction must be signed by three of the four entities involved in a blockchain network before it is considered valid. All transactions are added to a distributed ledger, whether legitimate or invalid, but only valid transactions alter the world state. If an endorsement policy requires more than one organization to sign a transaction, then the smart contract must be performed by a sufficient number of organizations in order to generate a valid transaction. In the preceding example 5.6, a smart contract transaction to transfer an automobile would need to be conducted and signed by both ORG1 and ORG2. Endorsement policies distinguish Hyperledger Fabric from other blockchains such as Ethereum or Bitcoin. Any node in the network can generate valid transactions in these systems. The Hyperledger Fabric model is more realistic; transactions must be confirmed by trusted organizations in a network. A valid issueIdentity transaction, for example, must be signed by a government organization, and a car transfer transaction must be signed by both the buyer and seller. Endorsement policies are intended to improve Hyperledger Fabric's modeling of these types of real-world interactions.

Finally, endorsement policies are a subset of Hyperledger Fabric policies. Other policies can be defined to limit who can query or change the ledger, as well as add and remove network participants. In general, policies should be agreed upon in advance by the consortium of enterprises in a blockchain network, though they are not set in stone. Indeed, policies can define the ground rules for how they can be changed. Furthermore, although this is a more complex issue, custom endorsement policy rules can be specified

in addition to those provided by Fabric.

5.5.3 Valid transactions

When a smart contract executes, it operates on a peer node owned by a blockchain network organization. The contract reads and writes the ledger using a set of input parameters known as the transaction proposal and associated program logic. Changes to the world state are captured as a transaction proposal response (or simply transaction response), which provides a read-write set including both the read states and the new states to be written if the transaction is valid. When the smart contract is executed, the world state is not updated.

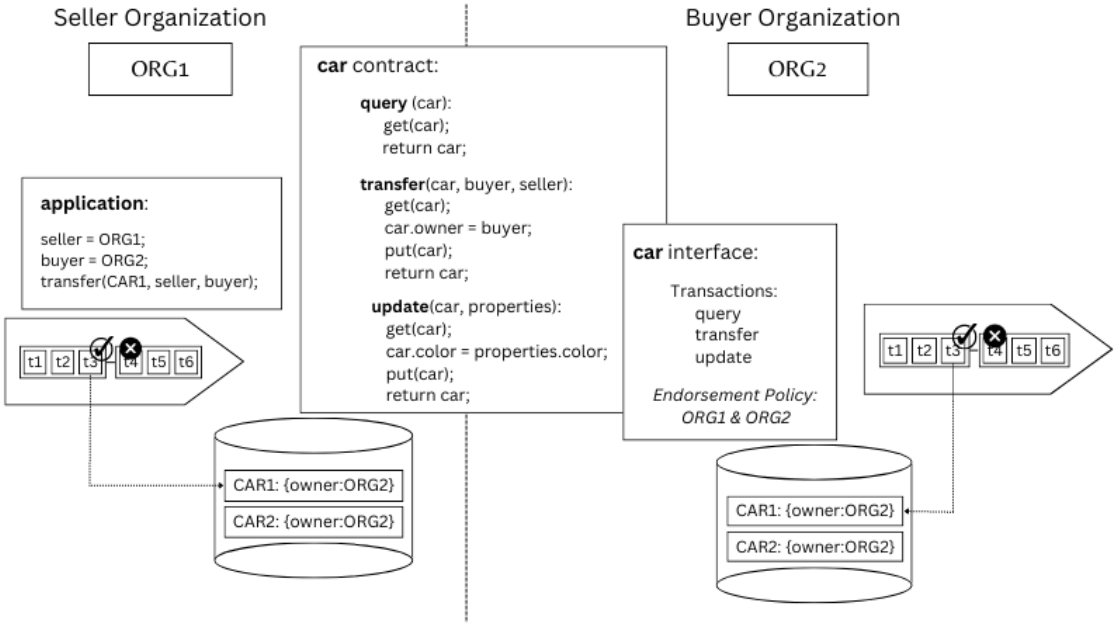

Figure 5.7. Every transaction has an identity, a proposal, and a response that has been signed by a group of organizations. All transactions, legitimate or invalid, are recorded on the blockchain, but only valid transactions contribute to the world state. Source: [26]

Examine the automobile transfer transaction. You can see transaction t3 for an automobile transfer between ORG1 and ORG2. Take note of how the input is signed by the application's organization ORG1, and the output is signed by both organizations defined by the endorsement policy, ORG1 and ORG2. These signatures were produced using each actor's private key, and they signify that anybody in the network may verify that all players in the network agree on the transaction information.

Each peer validates a transaction that is disseminated to all peer nodes in the network in two stages. First, the transaction is reviewed to confirm that it has been signed by enough organizations in accordance with the endorsement policy. Second, it is verified that the

current value of the world state matches the read set of the transaction when it was signed by the endorsing peer nodes; that no intermediary change has occurred. A transaction is considered as legitimate if it passes both of these conditions. All transactions, legitimate or invalid, are added to the blockchain history, but only valid transactions result in a change to the global state.

5.5.4 Channels

Through channels, Hyperledger Fabric enables an organization to join in numerous, different blockchain networks at the same time. An organization can participate in a network of networks by connecting to several channels. Channels allow for effective infrastructure sharing while ensuring data and communication privacy. They are autonomous enough to assist businesses in separating their work traffic with various counterparties, yet integrated enough to allow them to coordinate independent activities when necessary.

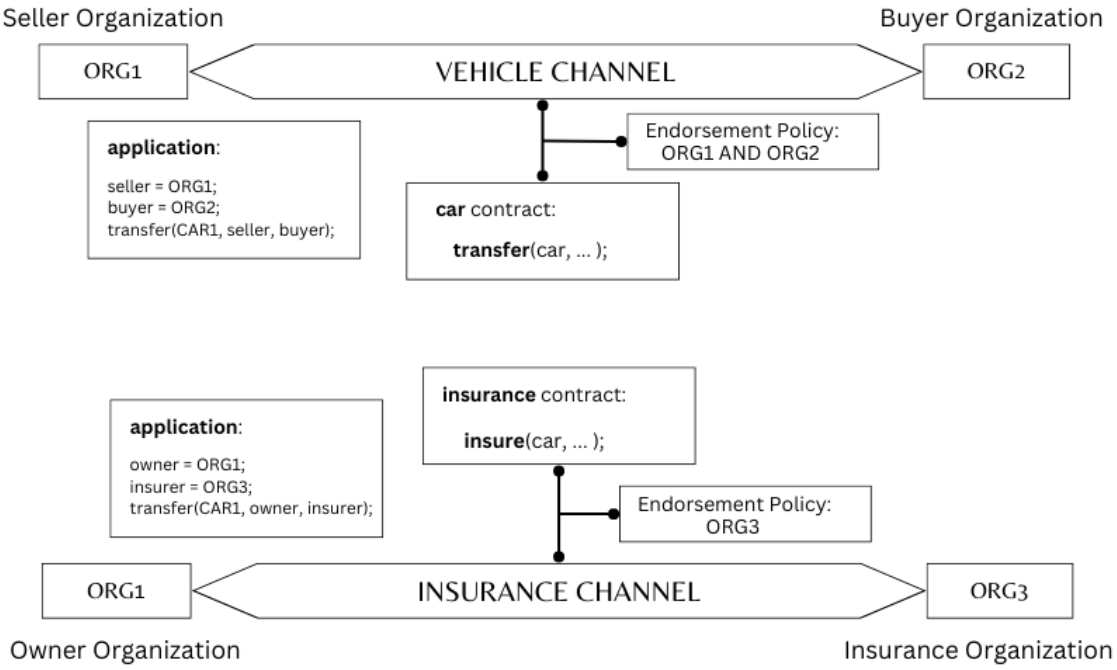

Figure 5.8. A channel is a completely separate communication system that connects a group of organizations. When a chaincode definition is committed to a channel, all of the smart contracts contained within the chaincode are made available to the channel's applications. Source: [26]

While the smart contract code is deployed on an organization's peers as part of a chaincode package, channel members can only execute a smart contract after the chaincode has been defined on a channel. The chaincode specification is a struct containing the parameters that determine how a chaincode works. The chaincode name, version, and endorsement policy are among the criteria. By endorsing a chaincode specification for their company, each channel participant agrees to the parameters of a chaincode. The

definition can be committed to the channel once a sufficient number of organizations (a majority by default) have authorized the same chaincode definition. Channel members can then execute the smart contracts contained within the chaincode, subject to the endorsement policy provided in the chaincode description. The endorsement policy applies to all smart contracts established within the same chaincode in the same way.

In the preceding example, a car contract is created on the VEHICLE channel, and an insurance contract is defined on the INSURANCE channel. The chaincode definition of automobile provides an endorsement policy that requires transactions to be signed by both ORG1 and ORG2 before they can be deemed legitimate. The insurance contract's chaincode specification stipulates that only ORG3 must endorse a transaction. ORG1 participates in two networks, the VEHICLE channel and the INSURANCE network, and can coordinate activities across these two networks with ORG2 and ORG3. Before deploying the smart contract to trade on the channel, channel participants can agree on the governance of a chaincode using the chaincode specification. Based on the preceding example, both ORG1 and ORG2 wish to support transactions that trigger the automobile contract. Because the default policy requires a majority of organizations to accept a chaincode definition, both organizations must approve. If ORG1 and ORG2 accept different chaincode definitions, they will be unable to commit the chaincode definition to the channel. This procedure ensures that a transaction originating from the automobile smart contract is accepted by both companies.

Chapter 6

Quantum Computing against Blockchains

Blockchain systems differ from conventional cryptosystems in that they are not only intended to secure information assets. A blockchain is a ledger and thus a valuable asset. There are three encryption techniques used in modern cryptography: symmetric cryptography, asymmetric cryptography, and hash functions. In symmetric cryptography, a single secret key is used to encrypt and decode a communication. It cannot be utilized in the context of blockchain since the message is delivered to many nodes in the network, and they may all read the message, even though the nodes should only be able to check the authenticity of the new data. This is where asymmetric cryptography, often known as public-key cryptography, comes in. A sender uses a private key to sign the transaction he wants to be recorded on the blockchain, creating a digital signature, and then broadcasts the transaction with a public key. Asymmetric encryption algorithms such as RSA or ECC cryptography, for example, are used to generate private/public key pairs that safeguard data assets stored on blockchains. The associated security is based on the difficulty of factoring in the case of RSA or the discrete logarithm issue in the case of ECC.[27] Hash functions accept any length input and return a fixed length string, with the critical characteristic that it cannot be reversed to get the original data through the hash output.

In a standard banking system, public- and private-key cryptosystems are employed to enforce data confidentiality, integrity, and access constraints. However, the data is detached from the key-pair. A central authority, for example, can quickly cancel the validity of a cryptographic key if it is lost or compromised. A new key-pair can be generated and associated with the data. The continuing integrity and secrecy of the data is ensured by revoking the key in a timely way. If a data breach happens, servers can be taken down and/or backups used. If an account is hacked, there are typically systems in place to assist the rightful owner to reclaim the account.[27]

In contrast, there is no central authority in a blockchain system to control users' access keys. The owner of a resource holds the private encryption keys by definition. There are no backups available offline. The blockchain, a cryptographic system that is constantly online, is regarded as the resource-or, at the very least, the official description of it. If a key is lost, the secured data asset is irreversibly lost. The data asset can be irreversibly stolen if the key or the device on which it is held is hacked, or if a vulnerability can be exploited. In summary, the secured resources in blockchains cannot be readily isolated from the encryption method being employed. As a result, blockchain systems are especially sensitive to breakthroughs in quantum technology.[27]

6.1 Impact of Quantum Computing on Blockchain

As previously stated, a quantum computer is based on quantum algorithms that are created particularly for one task. There are two main quantum algorithms that may represent a severe danger to blockchain security. The first is Shor's algorithm, which was introduced in 1992 to solve prime number factorization at an exponential speedup over traditional computers. An attacker using a quantum computer that performs Shor's algorithm may compute the private key using information from the public key that was broadcast along with a transaction. The hostile actor might then use the victim's private key to publish additional transactions. Stewart et al. (2018) investigated transaction-hijacking in a cryptocurrency context, where the attacker could use the computed private key to broadcast conflicting transactions before the original transaction was finalized in a block, spending the same currency as the victim but offering a higher fee to increase the likelihood of miners including the attacker's transaction instead of the original transaction. [20]

The next is Grover's algorithm, which was described in 1996 (Grover 1996), searches unstructured data for the input to a function that equals a desired value. Because it offers a quadratic speedup in computing the inverse of a hash function, it is primarily a danger to hash functions as part of symmetric cryptography. This enables two types of blockchain attacks that were previously thought to be computationally prohibitive due to the difficulty of reversing a hash function.

First, the technique may be used to look for hash clashes in order to change blocks without compromising the chain's integrity. If an attacker wishes to modify a transaction contained within a block, he must ensure that the block's hash remains unchanged such that the previous-block-pointer remains valid in the following block. If a collision is discovered, the attacker can modify transactions contained in the block without disturbing the blockchain. Second, the speedup provided by Grover's technique enables a miner using a quantum computer to mine blocks far quicker than a conventional computer. An attacker might dominate the generation of new blocks in a so-called 51%-attack, where one instance controls more than half of the network's computational power, allowing him to pick which data is added to the blockchain. This is especially problematic in a bitcoin setting, since it allows an attacker to prevent their own spending transactions from being recorded on the blockchain. It would also allow attackers to "rewrite history" by creating a hidden chain of selected or changed blocks, and once the chain was longer than the current main chain, he could broadcast it to the network. Because he controls the majority of the processing power, the forged chain will soon outnumber the current blockchain in length, and because the blockchain requires that the longest chain be acknowledged as the truth, the forged chain will then replace the previous chain. [20]

6.2 Threats and countermeasures

In the context of quantum computing, we are presented with two features of blockchain commitments being rendered invalid.

First, hash inversion is supposed to be computationally complex. If a quantum computer can drastically simplify this, the legitimacy of the upstream blockchain is no longer guaranteed, and the authenticity of entries in the blockchain is challenged. Grover's approach, as indicated above, searches the pre-image to a function value and can do so much quicker than the traditional brute force search of creating each output and comparing it to isolate the generating input.[28]

As a secondary vulnerability, a quantum computer may be able to undermine the security of any feature of a blockchain implementation that employs public/private key cryptography, whether it be in information flow between participants or in digital signatures.[28]

Quantum resistant cryptography, commonly known as PQC, is a discipline that encompasses the examination of (classical) cryptographic methods using a quantum computer. As previously said, there are some insights in this field, but it is still a pretty new topic with a lot of uncertainty and no agreed standards. Despite the fact that standards for quantum resistant cryptography are still being established, we may make some general assumptions about what elements will be necessary when creating systems that use blockchain-based technology.

The first point is about the hashing function itself. Grover's approach, as detailed in the preceding sections, gives a quadratic speedup over classical algorithms for assessing hash functions. Because this is not an exponential speedup like Shor's approach, the intended computational complexity of a function for secure applications can be restored simply by increasing the amount of bits employed in calculation. Due to the quadratic speedup of the method, just twice the number of bits are required.

The second point is about public/private key cryptography. Shor's algorithm is thought to pose a major threat to standard asymmetric encryption methods such as RSA and ECC, but the real influence of quantum computers on asymmetric encryption will be determined by a number of factors and remains unknown. However, while the key size in symmetric encryption can be increased to enhance security, the same is not possible with asymmetric encryption. This is why researchers are working on post-quantum cryptography methods, such as lattice-based and code-based cryptography.

6.3 Consortium blockchain

Consortium blockchain, which is administered by many organizations, only permits system members to access the blockchain. Consortium blockchain has been widely used as a backbone for cross-company and cross-organization scenarios such as medical information sharing, energy trading, e-health, smart homes, and so on. The underlying consortium blockchain mainly leverages public-key/asymmetric cryptography (e.g., RSA, ECDSA) and hash functions (e.g., SHA-256) to independently authenticate consortium members and hash/record transactions between the members to provide increased data integrity and traceability. At the application layer, consortium blockchain users interact securely with one another by utilizing public-key/asymmetric cryptography (e.g., RSA, ECDSA, ECDH), which is required for authenticating users and transferring keys used to further protect private data kept on-chain.[29] However, with the advent of large-scale quantum computers, the associated Shor's algorithm poses a severe threat to standard public-key/asymmetric cryptography. The consortium blockchain's security foundation would be breached.

To address the issue, the National Institute of Standards and Technology (NIST) has launched a call for proposals for PQ public-key cryptosystems, including digital signature algorithms and KEM algorithms, which is currently in its fourth round and is expected to deliver the first national standard in 2024.

6.4 Hyperledger Fabric

Hyperledger Fabric is one of the most successful existing consortium blockchains, and it supports executing distributed applications (i.e., smart contracts, aka chaincodes) on a consortium of organizations and peers in one channel, with business logic programmed as chaincodes written in standard, general-purpose programming languages (e.g., Go, Node.js, Java). Distributed system nodes like as peers, orderers, and clients are deployed in Hyperledger Fabric, and the peers are organized by the organizations to which they belong. Before running Hyperledger Fabric, all chaincodes must be deployed on specified peers [29]. During Hyperledger Fabric execution, peers can access on-chain data using keys, which are similar to member variables connected to the installed chaincodes, and a key can be a single datum or a tuple of data. Because Hyperledger Fabric records all changes and updates to the data and installed chaincodes, any modification can be instantly recognized.

There is a digital signature (CRYSTALS-DILITHIUM) and one KEM (CRYSTALS-KYBER) finalist algorithm in the fourth (current) round of NIST call for PQ algorithms, which will continue to be examined for consideration for standardization at the end of the fourth round. At the same time, there are two digital signature (i.e., Falcon, SPHINCS+) other candidate algorithms that have the potential to be standardized but are unlikely to be implemented by the conclusion of the fourth round.

We planned to analyze this framework for the reasons stated, with the goal of making it the first chaincode-based user authentication and key exchange system on top of the consortium blockchain, which provides fundamental PQ security mechanisms that most blockchain-based applications require. Furthermore, we did a systematic performance study of the system based on our methodology by comparing the changed version to the present official version.

6.4.1 Crypto Analysis

Hyperledger Fabric uses ECDSA to secure the transactions in the network by verifying the authenticity of the transactions and ensuring the confidentiality and privacy of the data. To secure the legitimacy of transactions in Hyperledger Fabric, clients sign them and network nodes verify them. The usage of digital signatures is a fundamental component of Hyperledger Fabric transaction security:

1. Transaction initiation: The client node creates a transaction and signs it with its private key. The signature is generated using the ECDSA algorithm and is appended to the transaction as an attribute.

2. Endorsement policy evaluation: The transaction is then sent to an endorsement policy, which is a set of smart contract functions that specify the conditions for executing the transaction. Endorsers, which are nodes in the network, evaluate the endorsement policy and sign the transaction with their private keys. The signatures from the endorsers are added to the transaction as attributes.

3. Transaction ordering: The signed transactions are then sent to the ordering service, which is responsible for ordering the transactions into blocks and ensuring that the same transaction is not processed twice. The ordering service signs each block with its private key.

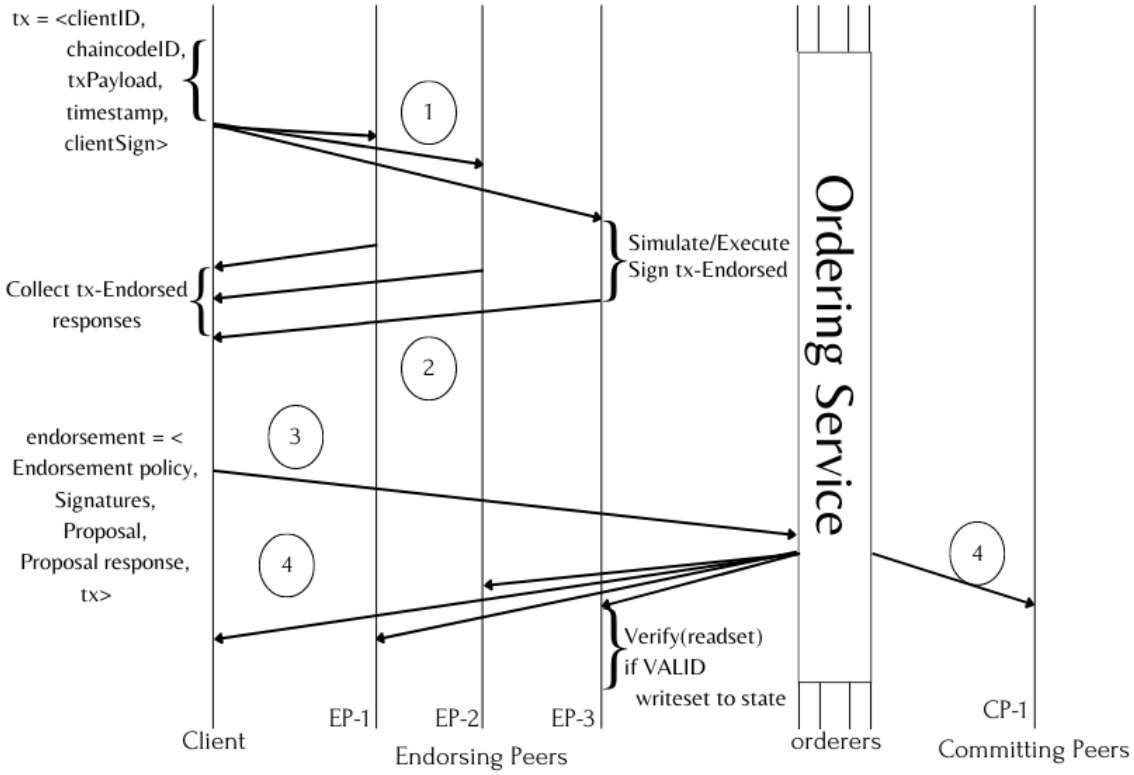

Figure 6.1. Hyperledger Fabric's transaction diagram therefore represents the process of validating, signing and recording transactions within the network. This process ensures the security, transparency, and integrity of the data within the platform.

4. Commitment to the ledger: The signed blocks are then broadcast to the committing peers, which are nodes in the network that maintain a copy of the ledger. The committing peers validate the signatures on the transactions and blocks and commit the transactions to the ledger.

At each step in the transaction process, the signatures are verified using the public keys of the corresponding nodes. This ensures that only authorized entities can initiate transactions, execute endorsement policies, order transactions, and commit transactions to the ledger. The signatures also provide a permanent record of the transaction, ensuring that the integrity of the data is maintained and that transactions cannot be repudiated.

The process of signing a transaction by an identity's private key is the same for all Service of Endorsement, Commit, Query, and Discovery, using first the build() method getting the bytes to be signed. Signing those bytes, then providing the signature on the sign() method before calling to send().[1]

1. generate proposal bytes with the identity's certificate

```
const idx = client.newIdentityContext(user);
const endorsement = channel.newEndorsement(chaincode_name);
```

[1]Hyperledger Fabric Docs, fabric-common: Working with an offline private key, Feb 14, 2023, https://hyperledger.github.io/fabric-sdk-node/release-2.2/tutorial-sign-transaction-offline.html

```
const build_options = {fcn: "move", args: ["a", "b", "100"]};
const proposalBytes = endorsement.build(idx, build_options);
```

2. calculate the hash: There exists multiple hash functions (such as SHA2/3). by default, the fabric client will use 'SHA2' with key size 256.

```
const hashFunction = xxxx; // A hash function by the user's desire

const digest = hashFunction(proposalBytes); // calculate the hash of the
// proposal bytes
```

3. calculate the signature: By default the the fabric client will use ECDSA with algorithm 'EC'.

```
// This is a sample code for signing the digest from step 2 with EC.
// Different signature algorithm may have different interfaces

const elliptic = require("elliptic");
const { KEYUTIL } = require("jsrsasign");

const privateKeyPEM = '<The PEM encoded private key>';
const { prvKeyHex } = KEYUTIL.getKey(privateKeyPEM); // convert the pem
// encoded key to hex encoded private key

const EC = elliptic.ec;
const ecdsaCurve = elliptic.curves["p256"];

const ecdsa = new EC(ecdsaCurve);
const signKey = ecdsa.keyFromPrivate(prvKeyHex, "hex");
const sig = ecdsa.sign(Buffer.from(digest, "hex"), signKey);

// now we have the signature, next we should send the signed transaction
// proposal to the peer
const signature = Buffer.from(sig.toDER());
```

4. use the signature for transaction proposal to peer(s)

```
endorsement.sign(signature);
const proposalResponses = await endorsement.send();
```

Finally, the transaction encryption for digital signatures provided by Hyperledger Fabric is an important component of network security. ECDSA delivers a high level of security and speed, and the verification process assures transaction legitimacy and integrity. We discovered how to replace Hyperledger Fabric's transaction encryption with quantum-secure techniques by analyzing it. The approach utilized during the work is explained in the next chapter.

Chapter 7

System analysis, implementation and security evaluation

Hyperledger Fabric is a variant of the Hyperledger Blockchain project. Hyperledger Fabric allows private, authorized access; network members must sign up through a MSP. The MSP is placed on each channel peer to guarantee that transaction requests submitted to the peer are from an authenticated and approved user identity. A network can have numerous MSPs; however, it is advised that each company have just one MSP. As a result, the whole network will have the same number of MSPs as the number of businesses. The MSP is a pluggable interface that supports multiple encryption schemes. Each entity must have a local MSP that is used to securely store credentials such as private keys utilizing hardware modules offered by the hardware platforms. TLS secures the whole communication between the various entities. Each network user is assigned an identity and an enrollment certificate. The enrollment certificate is made up of two parts: a private key and a public key. The private key is unique to each user and is used to digitally sign transactions delivered to the network. A hardware security module offered by the computer hardware platform is used to securely store the private key. The second component is a signcert, which is a publicly available certificate in the fabric implementation. Each Peer and Ordered has its own private key, public key, and CA signing certificate. A local MSP is linked to the Peer that possesses these IDs.

Based on this information gathered during the system analysis, we decided to direct the focus of the work on identities. These identities use classical asymmetric cryptography and are therefore vulnerable. The idea was to replace the classic algorithms with those supported by NIST. As mentioned above, the modularity of the MSP allows it to support different cryptographic standards. We therefore decided to integrate support for Dilithium and Falcon to the existing interface of the MSP. In addition to the mere practical implementation needed for the testing phase, we identified where to make changes. By defining a developer's manual where we outlined where to work to add any new standardized algorithms.

7.1 Related Work

There has been significant earlier research that has studied various ways to integrate PQC in the blockchain in the search for methods to make Hyperledger Fabric resistant to quantum threats. Campbell's (2019) study[30], for example, investigated the use of the QTESLA signature algorithm in Hyperledger Fabric, whereas S. Chen et al. (2020)[31]

recommended the use of the XMSS signing algorithm. Both research revealed the viability of implementing such algorithms on the blockchain, but they also highlighted several obstacles, such as key size and processing time, which could impair the system's overall performance. J. Hermelink et al. (2021) [32], for example, used a combination of post-quantum algorithms to secure the blockchain, combining XMSS for digital signature and KYBER for key exchange. In general, the majority of the articles followed a similar pattern, testing with various post-quantum algorithms and analyzing their performance and special issues in the context of Hyperledger Fabric. The paper 'A Permissioned Blockchain Safe from Both Classical and Quantum Attacks' by A. Amelia [33] is a key reference point for this master thesis. Amelia has created PQFabric, an implementation of Fabric with hybrid signatures that integrates with the Open Quantum Safe (OQS) library. In PQFabric's implementation, the PKI is protected by migrating from classical to hybrid digital signature primitives that contain a post-quantum counterpart. Amelia's proposed strategy was developed in 2020 taking into account the instability of the OQS project. During this period, the algorithms underwent continuous modifications to keep up with advances in cryptanalysis and changes and optimizations of the parameters of each algorithm. These prerequisites led to the choice of a hybrid approach. By proposing a hybrid quantum-classical signature scheme for transactions, using both post-quantum and classical cryptography. The implementation of Amelia was used as a model in the thesis. However, after the publication in July 2022 by NIST of the algorithms that went through the fourth and final round of standardization, it was chosen to use only the quantum signature.

One of the reasons that prompted us to re-implement a quantum-safe solution of Hyperledger fabric was due to the fact that the changes present on the PQFabric project were no longer working with recent versions of Hyperledger Fabric. The latter is a very active project that has had many code changes in two years. In addition, a custom version of the go-wrapper for the liboqs library was created in PQFabric. Recently, an official version of the wrapper for the library was released and included in the thesis code.

7.2 Proposed Solution

During the initial phase of the framework analysis. In the code, we identified that identities are issued by the CA, along with the CA's signature on them, is called a certificate, and the predominant certificate format is x.509. There are several entities in the blockchain network, such as peers, originators, client applications, and administrators, and only authorized entities can perform operations. The advantage of having all these identities is that they can be used for authentication, validation, signing, and issuing. For example, the user signs a transaction and sends it to the peer; the peer executes the transaction and, when it approves it, affixes the signature. Similarly, the ordering service creates a block and sends it to the peers. Each block will be signed by the ordering service. The idea is to make these cryptographic operations quantum secure. Existing work on the Hyperledger Fabric quantum transition is mostly obsolete. Both because of the continuous changes in the standardization process and the continuous evolution of the Fabric framework. For example, the PQFabric solution, mentioned earlier, makes use of hybrid certificates. Custom X.509 certificates that can support two keys, one classical and one post-quantum. This approach conflicts with one of the constraints we posed at the beginning. Make the number of changes to Fabric's code minimal. This was not possible because the X.509 standard does not currently support multiple signing algorithms in the same certificate. To provide an X.509-compliant solution, we decided to opt for a

single post-quantum key. The constraint of minimizing changes to the framework code was motivated to maintain a good balance between a quantum-safe solution and the needs of the corporate world that already uses this framework, which is hostile to large changes.

Our choice fell on the recent winners of NIST's third round, specifically the two signature algorithms most touted for standardization. Dilithium and Falcon are also highly supported by the community. There are OQS-recognized external projects working to integrate these algorithms into existing protocols such as TLS and X.509. But beyond the algorithms that were included in the code for the testing phase, the contribution of this paper lies in showing the crucial points of the framework to be modified for a quantum-secure digital signature. By letting the choice of the algorithm to be used remain a simple system configuration.

7.3 Architecture

The architecture of Hyperledger Fabric in production, is shown in the figure 7.1. Here you can see the scalability of the framework. This is to stress the need to make the number of changes minimal. In this section, we analyze in detail the developed solution by taking as an example only two organizations with an ordering service made up of only one orederer. In the figure 7.2 we can see the container-based infrastructure we implemented locally.

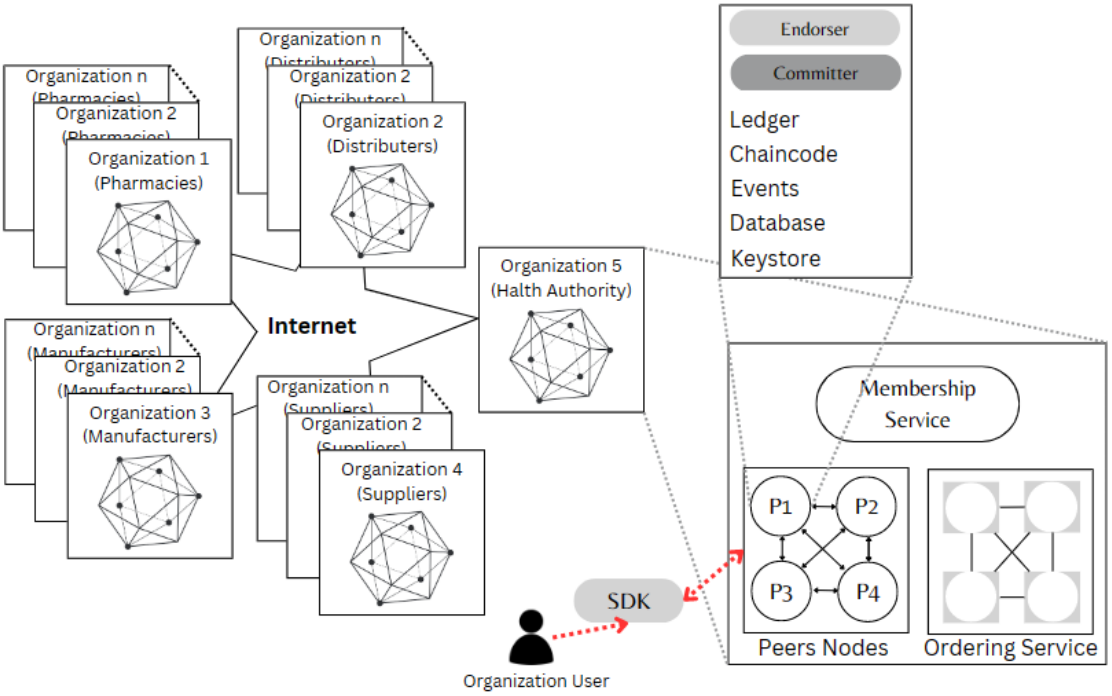

Figure 7.1. The image shows the use of the framework as a scalable solution for the pharmaceutical world, one of the possible applications of blockchain. The implementation for the thesis work tested the framework on a restricted network with only two organizations, with one peer per organization.

Virtualization has been implemented using Docker. This is the solution for developers, and in production there will be some changes at the software component level. This is the test network configuration when using a custom binary that cannot be used in production, but allows developers to work faster to create cryptographic material. In

this configuration, each organization receives two CAs: one for TLS and one for identity (MSP).

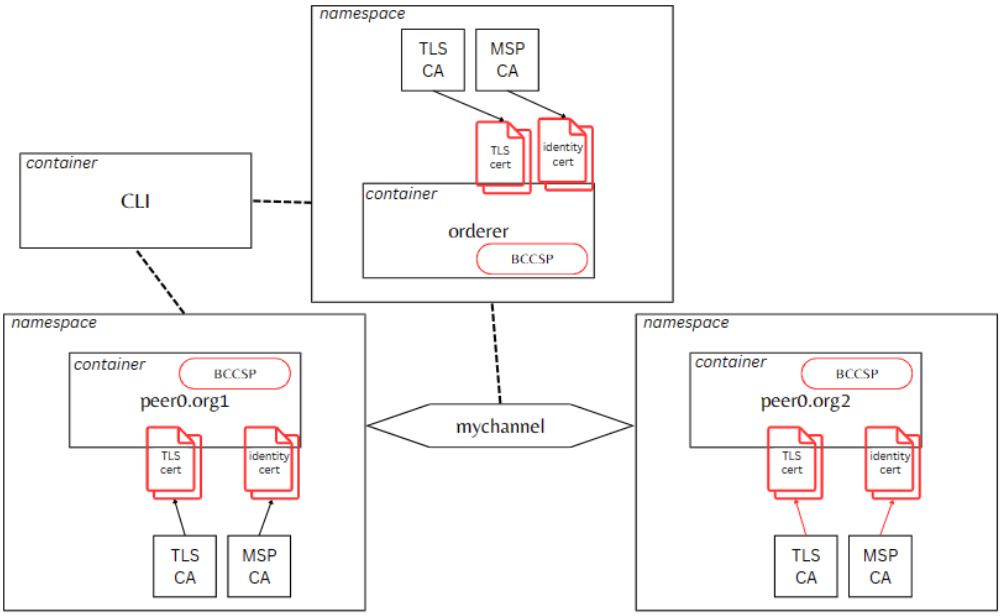

Figure 7.2. In the picture, several nodes of the network can be seen, represented in docker containers. Each node represents a participant in the network and can be an Orderer Node or a Peer Node. The image represents the nodes of the Hyperledger Fabric test network on which we worked. X.509 certificates are issued by the Certificate Authority (CA) and represent a form of digital identity for network participants. The main changes made to the framework are highlighted in red.

The certificates highlighted in red in the figure 7.2 were generated using Dilithium5 and Falcon1024 as the signing algorithm. Certificates used to sign messages exchanged within the channel and TLS certificates used to create TLS connections with quantum-secure digital signatures. The work done by Hyperledger Fabric facilitated the changes. In fact, the source code provides docker-compose files that allow easy configuration of network specifications. Configurations involving TLS have been highlighted in the image 7.3. Several variables set for this peer (container) are noted.

- TLS server authentication is enabled.
- TLS crypto material, including server private key, server certificate, and the certificate of CA which issues the server certificate.
- Volume mapped from the localhost to the path of crypto material inside the container.

With these variables, when peer node start is executed, the peer is ready for communicating with TLS in server authentication. The MSP configuration variables are also specified in the same file. The certificates are loaded inside the container in the paths specified by the volumes. The structure, as it is organized, allows only the peer image to be modified to use quantum cryptography. The remaining sections have been left almost unchanged and ready for deployment.

fabric-samples > test-network > docker > ! docker-compose-test-net.yaml

```
51  peer0.org1.example.com:
52    container_name: peer0.org1.example.com
53    image: hyperledger/fabric-peer:$IMAGE_TAG
54    environment:
55      #Generic peer variables
56      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
57      # the following setting starts chaincode containers on the same
58      # bridge network as the peers
59      # https://docs.docker.com/compose/networking/
60      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_test
61      - FABRIC_LOGGING_SPEC=INFO
62      #- FABRIC_LOGGING_SPEC=DEBUG
63      - CORE_PEER_TLS_ENABLED=true
64      - CORE_PEER_PROFILE_ENABLED=true
65      - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
66      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
67      - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
68      # Peer specific variabes
69      - CORE_PEER_ID=peer0.org1.example.com
70      - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
71      - CORE_PEER_LISTENADDRESS=0.0.0.0:7051
72      - CORE_PEER_CHAINCODEADDRESS=peer0.org1.example.com:7052
73      - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
74      - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org1.example.com:7051
75      - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.example.com:7051
76      - CORE_PEER_LOCALMSPID=Org1MSP
77    volumes:
78        - /var/run/:/host/var/run/
79        - ../organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp:/etc/hyperledger/fabric/msp
80        - ../organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls:/etc/hyperledger/fabric/tls
81        - peer0.org1.example.com:/var/hyperledger/production
82    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
83    command: peer node start
84    ports:
85      - 7051:7051
86    networks:
87      - test
```

Figure 7.3. The image shows the Docker Compose file of a Hyperledger Fabric test network, with particular attention to the fields relating to TLS. These fields are crucial for the configuration of TLS within the network

7.3.1 Node structure

Peers in Hyperledger Fabric are designed to run in Docker containers, which provides a flexible and scalable way to deploy and manage the software components. The container that stores the peer binary is made up of numerous functional and technical components. Initially, it includes the underlying operating system, which might be a Docker image with a Linux distribution. Then there are Hyperledger Fabric-specific components like the Fabric SDK library, which allows clients to connect with the peer via the gRPC interface, and the peer core component, which implements the consensus protocol, transaction management, validation mechanism, and other features. In addition, the container includes other components such as the identity management module and the ledger state storage unit, which allows transactional data to be stored in the distributed ledger. Lastly, to assure the system's scalability, dependability, and security, the container can be coupled with additional components such as database engines, authentication systems, and monitoring tools. In Hyperledger Fabric, the container structure housing the peer binary is meant to facilitate peer installation, setup, and administration. Furthermore, because Hyperledger Fabric is a highly flexible platform, the container structure may be tailored to the unique requirements of the distributed system in which the peer works.

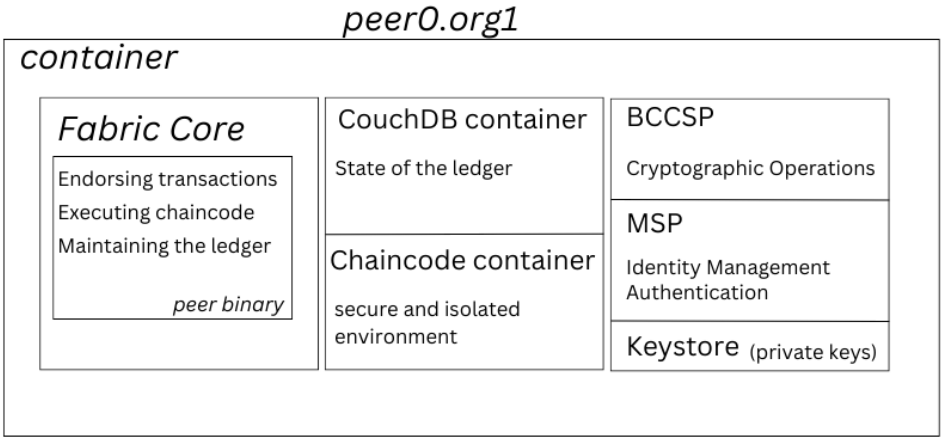

Figure 7.4. The image represents an overview of the software components working together within the Hyperledger Fabric peer container to enable transactions to be executed on the blockchain.

Node's Dockerfile

The peer Dockerfile in Hyperledger Fabric defines the configuration and creation of the Docker image of the container containing the peer. The Dockerfile begins with the "FROM" statement that specifies the base image that will be used for the container, usually a Linux distribution image. Next, the Dockerfile copies the files needed to configure the peer inside the container using the "COPY" instruction. These files include the peer binary, the peer configuration file, the public and private keys for the peer identity, and the TLS configuration file. The Dockerfile then defines instructions for installing the packages necessary for the peer to function, such as dependencies for the Fabric SDK library, support for TLS encryption, and transaction validation libraries. Next, the Dockerfile defines instructions for running the peer within the container. These instructions include starting the peer via the "peer node start" command, configuring the peer to connect to the channel via the configuration file, and enabling transaction logging on the distributed ledger. Finally, the Dockerfile defines the network port used by the peer through the "EXPOSE" statement, which exposes the default port (7051) used for communication with the other nodes in the channel.

The peer's Dockerfile in Hyperledger Fabric is an essential component for creating a customized Docker image for the peer. Customizing the Dockerfile allows the peer to be configured specifically for the needs of the distributed system in which it operates, allowing greater control and flexibility in managing the peer.

7.3.2 Signature Proposal

We propose a quantum signature scheme for transactions and TLS connections, using the chosen post-quantum algorithm Dilithium. Because of their reduced key size, lattice-based signature schemes, particularly those based on the short integer solution (SIS) problem, are more promising. Falcon was discovered to have the smallest signature lengths and key sizes among the tested lattice-based signatures. Dilithium systems were fast, but their signatures and key sizes were massive. For these reasons, we thought that demonstrating that the framework can work with the Dilithium algorithm, which in addition to being chosen by NIST as the most likely to be standardized has been shown to be hostile to integration into existing protocols (e.g., TLS), might be a more relevant outcome.

7.3.3 Identity management

As explained earlier, identity in Hyperledger Fabric is managed through X.509 certificates. The current X.509 standard does not support NIST's PQC algorithms for signing. To overcome this limitation and minimize the amount of modification within the framework code, we decided to use an open source library. This library it's called PQCrypto that is a cryptography project conducted by Chongqing University, China. The goal was to add PQC support to the go language for TLS and X.509, without Fabric's knowledge. This is possible because the PQCrypto library has all the functionality of the standard crypto library provided by golang. In addition to golang's crypto, PQCrypto added liboqs to support NIST's post-quantum signing algorithms, not only supports key generation, signing, and verification, but also supports all operations in X509.go and private key format conversion in PKCS8.go.

7.4 Code Structure

We built our solution on Hyperledger Fabric 2.5, the most recent release. For the implementations of post-quantum cryptographic signature algorithms we used LibOQS 0.7.2, which was the most recent version at the time of writing, although our code is compatible with all versions of LibOQS. Hyperledger Fabric is written in Go, whereas LibOQS is written in C. OQS recently released an official CGO wrapper for LibOQS. We assumed that these quantum-safe cryptographic functions would eventually be incorporated into the Go core library, so we conformed to the API rules used in Go's ECDSA implementation. In our implementation, post-quantum cryptography is exclusively used for message signing and verification, the wrapper provides just KeyPair, Sign, and Verify. The platform's functionality is enabled by two key software components, the Membership Service Provider (MSP) and the Blockchain Cryptographic Service Provider (BCCSP). The MSP implements identity and authorization management for nodes in the Hyperledger Fabric network by utilizing cryptographic certificates and signatures based on standards such as X.509. The BCCSP, on the other hand, provides a suite of cryptographic services, including key generation and management using algorithms such as RSA and Elliptic Curve Digital Signature Algorithm (ECDSA), digital signature creation and validation using the Secure Hash Algorithm (SHA), and encryption/decryption using Advanced Encryption Standard (AES) and other symmetric algorithms. The interaction of the MSP and BCCSP allows network nodes to verify network members' identities, enforce access regulations, safeguard user data through encryption, and assure secure identification and

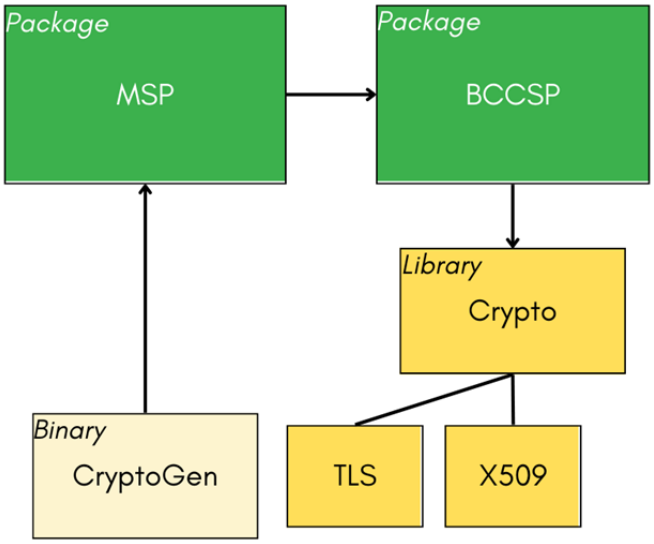

Figure 7.5. Encapsulation between the BCCSP and the MSP software components.

authorization management through the use of secure cryptographic protocols. The following are the changes made to the main software components of the Fabric framework for the quantum-resilient integrity property:

- *Blockchain Cryptographic Service Provider (BCCSP)*: The BCCSP module is intended to offer a consistent cryptography interface to the core Fabric that is independent of cryptographic method or implementation. It is a subset of the more broad Cryptographic Service Provider (CSP). The majority of our changes were to introduce a new key type and its related interface with KeyImport, KeyPair, Sign, Verify, and so on. We also modified the Signer module, which is shared by all BCCSP keys. As shown in the Figure 7.5, bccsp uses the crypto library for TLS and to manage X.509 certificates.

- *Membership Service Provider (MSP)*: the MSP provides an identity and authorization management framework for nodes in the Hyperledger Fabric network through the use of bccsp, which provides secure cryptographic services. No changes were required to the software module of this component, but it was necessary to analyze it to identify all cryptographic functions used to replace ECDSA with DILITHIUM.

- *Cryptogen*: This binary is not an official Hyperledger Fabric component, but it does provide a template for producing the cryptographic material necessary to execute Fabric from its configuration files. Organizations running Fabric may produce this material in whatever way they see fit, but any equivalent of the "cryptogen" binary would require comparable changes to build post-quantum key material and X.509 certificates.

In addition to these changes, as mentioned above, low-level changes were required. Changes that affected the libraries used by the golang language. As can be seen in the Figure 7.6, the crypto library invoked by the BCCSP module was replaced with PQCrypto, thus leaving all functionality related to the X.509 standard and the TLS protocol unchanged.

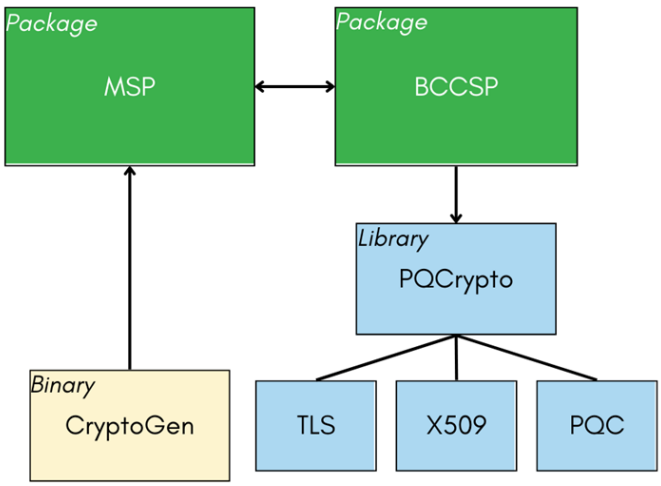

Figure 7.6. Software changes that we applied within our solution.

7.4.1 Software-based approach to BCCSP

The approach we implemented in the code focused mainly, as highlighted in the previous sections, on the BCCSP software component. The Signer interface is one of the main components of the BCCSP, which defines the behavior of objects that can sign digital data. A Signer object is responsible for applying a cryptographic signature to a given piece of data, using a private key that it manages. The signature can be later verified using the corresponding public key. In Hyperledger Fabric, the Signer interface is used by several components, such as the peer, orderer, and client SDK, to sign messages and transactions. Different implementations of the Signer interface can be used depending on the cryptographic algorithm and key management scheme that is chosen for the fabric network.

In our solution, it is possible to set up the use of both Dilithium and Falcon. *dilithium.go* and *falcon.go* files contain the implementation of the Signer interface and call the signing and verification methods implemented in the *liboqs* library. Key generation is handled in the *keygen.go* file where a struct is defined for each algorithm and the *Generatekey()* function implemented in the *liboqs* library is called. The keys are stored in the structs defined in the files *dilithiumkey.go* and *falconkey.go* for later use within the code. When a public key needs to be imported into the BCCSP key store, the *GoPublicKeyImportOptsKeyImporter* interface can be used to provide additional options that may be required to successfully import the key. These options can include things like the key's label and any key attributes or metadata that should be associated with the key. The *GoPublicKeyImportOptsKeyImporter* interface defines the following method:

- *KeyImporter*: This method takes a byte array representing the raw key material to be imported, along with any additional import options that are specified, and returns a *Key* object that represents the imported key.

When dealing with a x509 certificate, the *GoPublicKeyImportOptsKeyImporter* interface can be used to specify options for importing the public key from the certificate into the BCCSP key store. The implementation of the GoPublicKeyImportOptsKeyImporter interface for importing quantum-safe public keys from a x509 certificate it's shown here 7.7:

```go
type x509PublicKeyImportOptsKeyImporter struct {
    bccsp *CSP
}
func (ki *x509PublicKeyImportOptsKeyImporter) KeyImport(raw interface{}, opts bccsp.KeyImportOpts) (bccsp.Key, error) {
    x509Cert, ok := raw.(*x509.Certificate)
    if !ok {
        return nil, errors.New("Invalid raw material. Expected *x509.Certificate.")
    }
    pk := x509Cert.PublicKey
    switch pk := pk.(type) {
    case *ecdsa.PublicKey:
        return ki.bccsp.KeyImporters[reflect.TypeOf(&bccsp.ECDSAGoPublicKeyImportOpts{})].KeyImport(
            pk,
            &bccsp.ECDSAGoPublicKeyImportOpts{Temporary: opts.Ephemeral()})
    case *rsa.PublicKey:
        return &rsaPublicKey{pubKey: pk}, nil
    case *dilithium5.PublicKey:
        return ki.bccsp.KeyImporters[reflect.TypeOf(&bccsp.DILITHIUMGoPublicKeyImportOpts{})].KeyImport(
            pk,
            &bccsp.DILITHIUMGoPublicKeyImportOpts{Temporary: opts.Ephemeral()})
    case *falcon1024.PublicKey:
        return ki.bccsp.KeyImporters[reflect.TypeOf(&bccsp.FALCONGoPublicKeyImportOpts{})].KeyImport(
            pk,
            &bccsp.FALCONGoPublicKeyImportOpts{Temporary: opts.Ephemeral()})
    default:
        return nil, errors.New("Certificate's public key type not recognized. Supported keys: [ECDSA, RSA, DILITHIUM, FALCON]")
    }
}
```

Figure 7.7. In this implementation, the KeyImporter method takes the raw bytes of the x509 certificate, parses it into a Certificate object using the x509 package, and then extracts the public key from the certificate. The public key is then converted into a BCCSP Key object using the bccsp.KeyImporters[TYPE].KeyImport method, which converts the public key to the appropriate format for use with the corresponding struct.

Once all the necessary interfaces have been implemented for the BCCSP to handle the keys of a new algorithm. It is now possible to add this option to the generation of the software-bassed BCCSP instance. The implemented structures for the two new algorithms Dilithium and Falcon are mapped here. For example, the corresponding DilithiumKeyGenerator for key creation is mapped to DILITHIUMKeyGenOpts. This encapsulation of structures, which initially made reading the code more complex, has turned out to be the strength of this framework. In fact, the rest of the code is independent of the algorithms used, and the addition of new algorithms for signing and KEM will mainly consist of a refactoring of the previously explained structures.

Finally, the last two modified files are *keys.go* and *filesks.go*. In both of them, we only added cases to the switches for the newly introduced algorithms. *keys.go* concerns the transformation from public key to PEM and vice versa to derive keys. While *filesks.go* is used to interface with the local keystore and manage keys.

7.5 Integration Challenges

Overall, the Hyperledger Fabric codebase did an excellent job of encapsulating its cryptographic interface in the BCCSP while making minimal assumptions about key or algorithm type. We expect that integrating other codebases with quantum-safe cryptography will need considerable reworking. The two main structures required for the PQC transition are dilithiumPrivateKey and dilithiumPublicKey, both of which implement the BCCSP.key interface, which is how a cryptographic key is represented in the fabric. The part of the source code that required the most changes is the cryptogen binary. This is

not an official support for the fabric framework, but it was necessary for the testing part. Here we encountered functions such as csp.GeneratePrivateKey. ECDSA-specific functions that were rewritten to provide support for DILITHIUM. The changes on cryptogen would not be needed in the production environment because a real CA would be used instead.

Moreover, the quantum solution we provide does not break the encapsulation between BCCSP and MSP. Lower Go's PQCrypto library hides the quantum support while leaving the basic operations of MSP unaffected. Operations such as [33]:

- Extracts cryptographic keys from an X.509 certificate. It does this both on initialization (extracting public keys from its own X.509 certificate and importing the corresponding private key from its keystore) and during signature verification (deserializing the public keys from another node's identity proto).
- Stores the public and private keys for the node directly in an internal proto structure called an *Identity*.
- Provides its own stored keys to the BCCSP when signing a message.

In these three areas, the only changes were in key dimension controls that no longer conformed to the new algorithms used. For each new algorithm, developers must add a new key type to the current code base and implement its interface methods in the BCCSP package. As NIST has recently reached the last round of standardization, we have implemented an algorithm-specific key type that will most likely be confirmed as a standard by 2024. Regarding the interface with GO libraries, operations that require an algorithm identifier, such as key marshaling/unmarshalling or X.509 signature algorithm support with post-quantum algorithms, we have to wait for the release of official libraries. Developers that wait for complete standardization and per-language implementation of post-quantum cryptography will not face this obstacle. However, although NIST solicits comments on the incorporation of these non-standard algorithms into production systems, preserving cryptoagility is another problem. [33]

7.6 Potential extensions

In our thesis work, we evaluated a simplified configuration of Fabric. For example, approval and consent policies in production would be more complex and require more than one signature for approval. Changes to policies and the number of peers would result in more signatures per block. All this would significantly increase the impact of signing and verification timelines. We expect that in the future, performance can be analyzed on larger case studies and with more realistic policies. In addition, to try to make the framework fully resilient to quantum computers, other cryptographic areas need to be modified. KEM functions still use classical algorithms. The idea might be to extend BCCSP with the KYBER algorithm, which seems to be close to standardization by NIST. This could make the whole framework independent of classical algorithms as far as asymmetric cryptography is concerned. In our implementation, we rely on an unofficial library PQCrypto. This is to go against Fabric's code and reduce modifications. This library is recognized by the OQS project, but it is not present in the Go language. It has been added locally and therefore cannot currently become an official solution. This library allowed us to use TLS 1.3 and the X.509 standard with the PQC algorithms without any changes to the code. Speeding up the testing phase and decreasing the workload. This library used the

official format for the other algorithms supported by Go, hoping that in the future with standardization the same format now assumed will be maintained. Unless the Golang and NIST APIs agree, this is likely to be an issue for future integrations as well.

Chapter 8

Performance Evaluation

We tested our solution on a network with one orderer and two peers, each operating in its own Docker container. The client (which was running in a fourth container) submitted all transactions to a single peer, while the second peer served as an endorser. In each of the containers, the source code of the associated binary is reconstructed. In order not to create problems, we had to modify the images provided by Hyperledger Fabric so that all the dependencies needed for PQC could be installed before compilation. A custom image was used with the set of dependencies already installed that we made publicly available on a Docker hub repository.

All the different tests were run on a system running Ubuntu 18.04.6 LTS with 32 GB of RAM and 16 CPU core.

8.1 Test Network Deploy

The benchmark measured the time required to implement a simple test network. For each set of benchmarks, we invoked the './network up' command. When running the ./network.sh up command in a Hyperledger Fabric test network configuration, the following main steps are involved:

- *Creating Docker containers*: The command creates Docker containers for the required network components, such as peer nodes, ordering service and CA.
- *Network configuration*: The network components are configured with the required configuration files, such as network topology, consensus algorithm and network policies.
- *Generation of cryptographic materials*: Required cryptographic materials for network participants, such as digital certificates and cryptographic keys, are generated.
- *Startup of network components*: Docker containers for network components are started to make them ready for use.

The official (In the graphs 8.1 identified with ECDSA) cryptographic configuration only signs transactions with ECDSA defined on the NIST P-256 curve (as specified by the FIPS 186-3) and offers classic 128-bit security. We have compared with the same benchmark run with nodes configured to sign and verify using the quantum (CRYSTAL-Dilithium and Falcon1024) finalist scheme of the third NIST phase as implemented in libOQS 0.7.2..

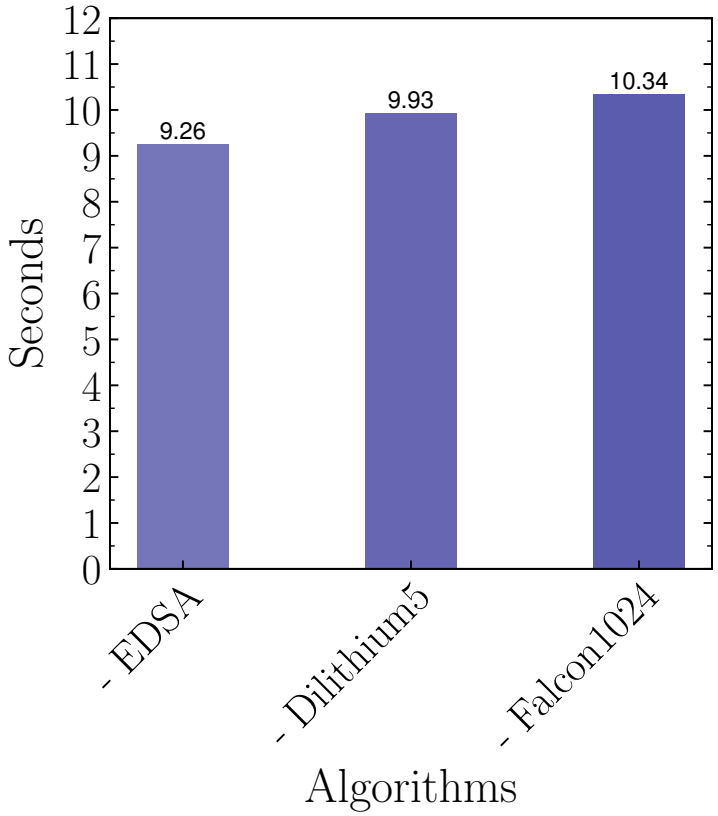

Figure 8.1. Comparison of our solution with Dilithium and Falcon1024 and the official Hyperledger framework for Test Network Deploy.

The results of the network implementation tests gave comparable results for the three algorithms. The data are the result of averaging 50 executions per algorithm of the network.sh script with the up command. This phase generates certificates for each node and ends when all nodes are active and listening on a specific port. Falcon1024 performs slightly less well, assuming that on a machine with floating-point hardware better results would be obtained.

8.2 Channel Generation Test

The benchmark timed the amount of time it took to construct a channel between Org1 and Org2. Channels are built in Hyperledger Fabric by producing a channel creation transaction that is included in a genesis block. In a join request, the genesis block is provided to an ordering service node, and the channel construction transaction describes the channel's initial setup. The construction of the genesis block and the insertion of the channel formation transaction are required phases in the Hyperledger Fabric channel creation process. The genesis block acts as the channel's beginning point, establishing its initial state and setup. The following information is included in the genesis block:

- *Channel Configuration*: The channel configuration defines the channel's policies and membership. This includes the names of the participants, policies for adding and removing participants, and the channel's consensus mechanism.

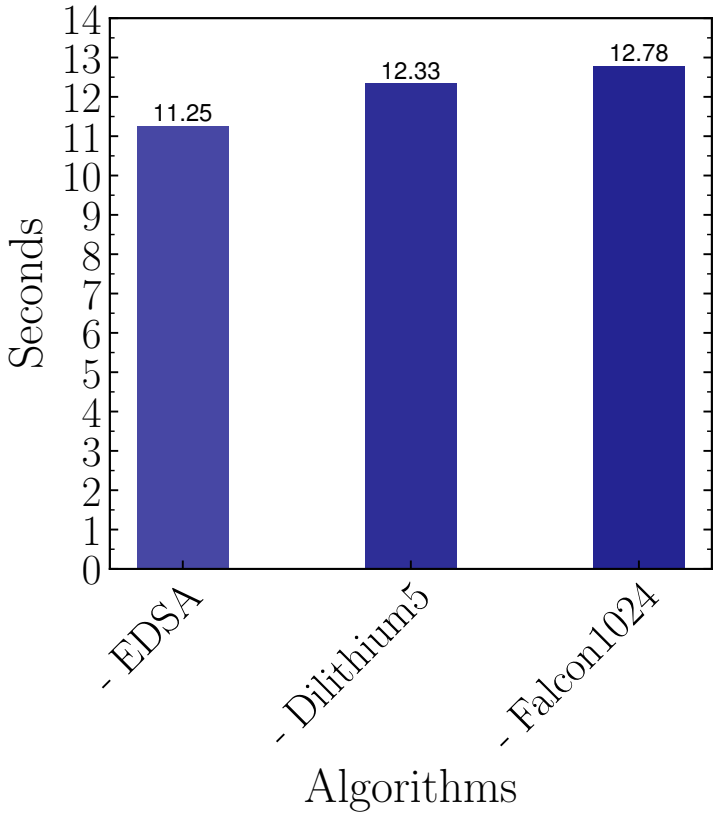

Figure 8.2. Comparison between our solution and official Hyperledger framework for the generation of a channel.

- *Initial Ledger State*: The initial ledger state defines the ledger's starting state, which includes any initial assets or contracts deployed to the channel.
- *Cryptographic Material*: The genesis block also contains cryptographic data used to secure the channel and validate transactions. This includes public keys, certificates, and other cryptographic artifacts used to establish channel trust and security.

It can be seen from the graph 8.2 that the slowest algorithm turns out to be Falcon. The difference could also be due to the small number of times the averaging was performed. The test was repeated 50 times for each algorithm. For a small network such as the one tested, the difference appears minimal. The effect it would have on an implementation that scales like the real cases should be tested.

8.2.1 Transaction performance

The last test that was run involves using the framework to perform normal transactions. Going to simulate the operation of a hypothetical real case of a dealership. For this test, the FabCar chaincode that Hyperledger provides was used to see how it works with practical examples. The transactions we tested fall into two categories. Query-type transactions, where the blockchain is queried for values such as:

- *QueryCar()*: allows query on individual car based on CarID.

- *QueryAllCars()*: simply iterates the record stored in the ledger and put them into a formatted result. No argument is needed for this function.

The other type of transactions are those of the invoke type. Invoke() usually defines some additional actions (functions) when the blockchain code is invoked. Of this type, we have analyzed the following:

- *CreateCar()* is for adding new car record into the ledger.
- *ChangeCarOwner()*: can change the owner of a car specified by CarID.

The goal of the test was to identify how the algorithm for signing and verification affects throughput. The comparison was made on the number of transactions executed per second. Timings were taken by averaging by repeating the transaction 1000 times for each algorithm. Each transaction consists of the following steps:

Step 1. The client requests the execution of a chaincode. In the distributed ledger network, the peers responsible for the execution of a chaincode (called endorsers) are determined when the chaincode is registered in the network. During the execution request, the client sends arguments and the chaincode name to the endorsers.

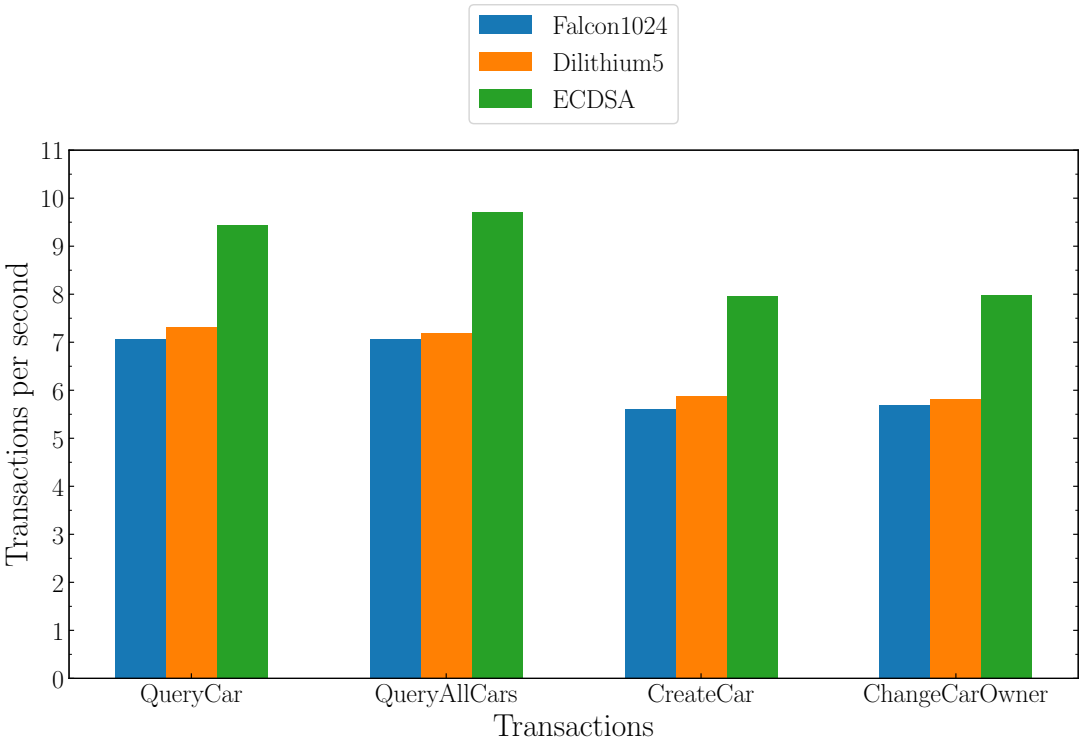

Figure 8.3. Tests performed to monitor the performance of the framework in terms of throughput per second. Each function was executed 1000 times by averaging over the recorded times. It is apparent that ECDSA is the best performing algorithm with Dilithium5 as the runner-up. Although the difference does not appear to be a problem for the size of the test network analyzed, the effect on a system scaling as in a real-world scenario should be analyzed.

Step 2. The endorsers execute a chaincode. Each endorser executes the chaincode specified by the client with the arguments and creates an RW-set signed with its private

key. This signature serves as evidence that the endorser has executed the chaincode. Finally, the endorser sends the signature to the client.

Step 3. The client collects the signatures. To create a transaction, the client must collect signatures from a sufficient number of endorsers. The total number of signatures is specified at the time of the chaincode registration. For this reason, the client performs steps 1 and 2 for the endorsers. Finally, the client creates a transaction with the RW-set and the signatures.

Step 4. The client sends the transaction to an orderer. The client requests a peer called an orderer to reflect the transaction in the ledger. The orderer collects multiple transactions, arranges batches of the submitted transactions into a well-defined sequence, and packages them into a block. The number of transactions in a block depends on either the block size or block generation interval.

Step 5. The orderer distributes the block to all peers. Transactions may conflict with each other because the orderer receives requests from multiple clients in parallel. The orderer checks all the RW-sets in the transactions and stores the block to its ledger, while determining invalid transactions. The invalid transactions are marked as invalid. After the block is finalized, the orderer distributes the block to all peers.

Step 6. The peers store the block to the individual ledger. To prevent unauthorized writing to the ledger, the peers check whether the transactions have been signed by the endorsers and the block has been signed by the orderer. Invalid transactions are ignored, and valid transactions are stored in individual ledgers whose contents are identical for all peers.

Chapter 9

Conclusions

The proposed work aims to provide a redesigning and extension of the Hyperledger Fabric framework. The official version of the framework proposed by IBM is vulnerable to the quantum threat. The modified version that was developed while working on this thesis aims to partially overcome this weakness. We tried to keep the main structure of Fabric mostly intact. This was made possible by the custom library PQCrypto. The library allowed us to integrate PQC into the language in which Fabric is written, Golang. In this way, NIST algorithms could be invoked effortlessly and without Fabric's knowledge. In addition, the library allowed us to support two core protocols of the framework, TLS and X.509, by handling the new algorithms with the same data structures that classic algorithms are handled with in the official libraries. The necessary changes to Fabric's source code mainly involved the creation of data structures and functions that invoke the PQC algorithms. In particular, it is now possible to select from blockchain system configurations PQC algorithms for digital signatures.

Tests conducted on the new version of the framework showed that performance did not decrease significantly with the algorithms we tested. Some critical issues have emerged in terms of key size. This led us to modify some of the checks that Fabric used to perform on key extraction from the keystore, which limited the number of transactions per second.

Future work on this topic could involve extending the 'use of PQC in the code further, for example by modifying the KEM part with algorithms such as KYBER. In this way, the dimensional impacts of post-quantum KEMs could be explored independently on the framework. Moreover, a more direct evolution of the proposed work could also be the adaptation of the solution with the algorithms to the PQC libraries that will soon be a standard.

Bibliography

[1] N. Kannengießer, S. Lins, T. Dehling, and A. Sunyaev, "Trade-offs between distributed ledger technology characteristics", ACM Comput. Surv., vol. 53, may 2020, DOI 10.1145/3379463

[2] M. Rauchs, A. Glidden, B. Gordon, G. C. Pieters, M. Recanatini, F. Rostand, K. Vagneur, and B. Z. Zhang, "Distributed ledger technology systems: A conceptual framework", Available at SSRN 3230013, 2018

[3] X. Fu, H. Wang, and P. Shi, "A survey of blockchain consensus algorithms: mechanism, design and applications", Science China Information Sciences, vol. 64, Feb 2021, pp. 1–15, DOI 10.1007/s11432-019-2790-1

[4] A. Lopez Vivar, A. Castedo, A. Sandoval Orozco, and G. Villalba, "Smart contracts: A review of security threats alongside an analysis of existing solutions", Entropy, vol. 22, 02 2020, p. 203, DOI 10.3390/e22020203

[5] N. El Ioini and C. Pahl, "A review of distributed ledger technologies", On the Move to Meaningful Internet Systems. OTM 2018 Conferences (H. Panetto, C. Debruyne, H. A. Proper, C. A. Ardagna, D. Roman, and R. Meersman, eds.), Cham, 2018, pp. 277–288, DOI 10.1007/978-3-030-02671-4_16

[6] S. Y. Popov, "The tangle", Available at https://www.docdroid.net/mWTNlgd/iota1-2.pdf, August 2015. [Online]

[7] I.-C. Lin, C.-C. Chang, and Y.-S. Chang, "Data security and preservation mechanisms for industrial control network using iota", Symmetry, vol. 14, no. 2, 2022, DOI 10.3390/sym14020237

[8] A. Iftekhar, X. Cui, and Y. Yang, "Blockchain technology for trustworthy operations in the management of strategic grain reserves", Foods, vol. 10, no. 10, 2021, DOI 10.3390/foods10102323

[9] N. Chaudhry and M. M. Yousaf, "Consensus algorithms in blockchain: Comparative analysis, challenges and opportunities", 2018 12th International Conference on Open Source Systems and Technologies (ICOSST), 2018, pp. 54–63, DOI 10.1109/ICOSST.2018.8632190

[10] H. Sheth and J. Dattani, "Overview of blockchain technology", Asian Journal For Convergence In Technology (AJCT) ISSN -2350-1146, Apr. 2019

[11] I. Ahmed and M. A. Shilpi, "Blockchain technology a literature survey", International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 p-ISSN: 2395-0072, vol. 5, no. 10, 2018, pp. 1490–1493

[12] A. Antonopoulos, "Mastering bitcoin", O'Reilly, 2015, ISBN: 9781491902608

[13] Wikimedia, "Bitcoin Block Data", December 2021, https://commons.wikimedia.org/wiki/File:Bitcoin_Block_Data.png

[14] A. Shahaab, B. Lidgey, C. Hewage, and I. Khan, "Applicability and appropriateness of distributed ledgers consensus protocols in public and private sectors: A systematic review", IEEE Access, vol. 7, 2019, pp. 43622–43636, DOI 10.1109/ACCESS.2019.2904181

[15] 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, 2020, pp. 21091–21116, DOI 10.1109/access.2020.2968985

[16] M. Allende, D. L. Leon, S. Ceron, A. Leal, A. Pareja, M. Da Silva, A. Pardo, D. Jones, D. Worrall, B. Merriman, J. Gilmore, N. Kitchener, and S. E. Venegas-Andraca, "Quantum-resistance in blockchain networks", 2021, DOI 10.48550/ARXIV.2106.06640

[17] D. Stebila and M. Mosca, "Post-quantum key exchange for the Internet and the Open Quantum Safe project", Selected Areas in Cryptography – SAC 2016 (R. Avanzi and H. Heys, eds.), Cham, October 2017, pp. 1–24, DOI 10.1007/978-3-319-69453-5 2

[18] D. J. Bernstein, "Introduction to post-quantum cryptography", 2009, pp. 1–14, DOI 10.1007/978-3-540-88702-7_1

[19] Bas Westerbaan, "Nist's pleasant post-quantum surprise", July 2022, https://blog.cloudflare.com/nist-post-quantum-surprise/

[20] N. Kappert, E. Karger, and M. Kureljusic, "Quantum computing - the impending end for the blockchain?", Pacific Asia Conference on Information Systems (PACIS), Dubai, UAE, 2021, Available at SSRN: https://ssrn.com/abstract=4075591, Jun 2021

[21] D. Joseph, R. Misoczki, M. Manzano, J. Tricot, F. Pinuaga, O. Lacombe, S. Leichenauer, J. Hidary, P. Venables, and R. Hansen, "Transitioning organizations to post-quantum cryptography", Nature, vol. 605, 05 2022, pp. 237–243, DOI 10.1038/s41586-022-04623-2

[22] M. Kumar, "Post-quantum cryptography algorithm's standardization and performance analysis", Array, vol. 15, 2022, p. 100242, DOI https://doi.org/10.1016/j.array.2022.100242

[23] IBM, "Behind the architecture of hyperledger fabric", February 2018, https://www.ibm.com/blogs/research/2018/02/architecture-hyperledger-fabric

[24] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains", Proceedings of the Thirteenth EuroSys Conference, New York, NY, USA, 2018, DOI 10.1145/3190508.3190538

[25] Xiao Shi, "Workflow of hyperledger fabric", June 2021, https://blog.51cto.com/shijianfeng/2914477

[26] Hyperledger Fabric Docs, "Architecture reference", 2023, https://hyperledger-fabric.readthedocs.io/en/release-2.5/architecture.html

[27] J. J. Kearney and C. A. Perez-Delgado, "Vulnerability of blockchain technologies to quantum attacks", Array, vol. 10, 2021, p. 100065, DOI https://doi.org/10.1016/j.array.2021.100065

[28] B. Rodenburg and S. Pappas, "Blockchain and quantum computing", Jun 2017, DOI 10.13140/RG.2.2.29449.13923

[29] S. Xu, A. Sun, X. Cai, Z. Ren, Y. Zhao, and J. Zhou, "Post-quantum user authentication and key exchange based on consortium blockchain", 2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS), 2021, pp. 667–674, DOI 10.1109/ICPADS53394.2021.00089

[30] R. Campbell, "Transitioning to a hyperledger fabric quantum-resistant classical hybrid public key infrastructure", The Journal of the British Blockchain Association, vol. 2, 11 2019, pp. 1–11, DOI 10.31585/jbba-2-2-(4)2019

[31] Y. Cao, Y. Wu, W. Wang, X. Lu, S. Chen, J. Ye, and C.-H. Chang, "An efficient full hardware implementation of extended merkle signature scheme", IEEE Transactions

on Circuits and Systems I: Regular Papers, vol. 69, no. 2, 2022, pp. 682–693, DOI 10.1109/TCSI.2021.3115786

[32] J. Hermelink, T. Pöppelmann, M. Stöttinger, Y. Wang, and Y. Wan, "Quantum safe authenticated key exchange protocol for automotive application", 18^{th} escar Europe : The World's Leading Automotive Cyber Security Conference (Konferenzveröffentlichung), 2020, DOI 10.13154/294-7549

[33] A. Holcomb, G. Pereira, B. Das, and M. Mosca, "Pqfabric: A permissioned blockchain secure from both classical and quantum attacks", 05 2021, pp. 1–9, DOI 10.1109/ICBC51069.2021.9461070

Appendix A

User Manual

This appendix describes how to configure the environment used for testing the proposed solution.

A.1 Prerequisites

To set up Hyperledger Fabric on a Linux-based operating system such as Ubuntu, Debian, or CentOS, you will need to have the following prerequisites [26]:

- Install the latest version of git if it is not already installed. (git version 2.17.1)
- Install the latest version of cURL if it is not already installed. (curl 7.58.0)
- Install the latest version of Docker if it is not already installed. (Docker version 23.0.1, build a5ee5b1)
  - Once installed, confirm that the latest versions of both Docker and Docker Compose executables were installed.
  - Make sure the Docker daemon is running.
  - Add your user to the Docker group.
- Install the latest version of Go if it is not already installed. (go version go1.19.3)
  - $GOPATH and $GOPATH/bin are added to $PATH

Once these prerequisites have been set, it will be necessary to follow the steps for installing the PQCrypto library explained in section B.2 of the Developer's Manual.

A.2 Before you begin

Before you can run the test network, you need to build images and binaries. To build Hyperledger Fabric:

```
$ make dist-clean all --always-make
```

A.3 Using the Fabric test network

After cloning the modified Hyperledger Fabric repository github/CosimoMichelagnoli/-fabric, you can use the fabric-samples repository to run a test network. The test network is offered to allow you to run nodes on your own system. The network can be used by developers to test their smart contracts and applications. The network is only intended to be used as a testing tool, not as a blueprint for how to build up a network. It is based on a restricted configuration that should not be used as a model for installing a production network:

- It includes two peer organizations and an ordering organization.
- For simplicity, a single node Raft ordering service is configured.
- To reduce complexity, a TLS Certificate Authority (CA) is not deployed. All certificates are issued by the root CAs.
- The sample network deploys a Fabric network with Docker Compose. Because the nodes are isolated within a Docker Compose network, the test network is not configured to connect to other running Fabric nodes.

A.3.1 Bring up the test network

You can find the scripts to bring up the network in the test-network directory of the *fabric* repository. Navigate to the test network directory by using the following command:

```
$ cd fabric/test-network
```

In this directory, you'll find network.sh, an annotated script that sets up a Fabric network using Docker images on your local system. The network can then be activated by giving the following command. If you try to run the script from another directory, you will have problems:

```
$ ./network.sh up
```

This command creates a Fabric network that consists of two peer nodes, one ordering node. If the command completes successfully, you will see the logs of the nodes being created:

```
Creating network "fabric_test" with the default driver
Creating volume "net_orderer.example.com" with default driver
Creating volume "net_peer0.org1.example.com" with default driver
Creating volume "net_peer0.org2.example.com" with default driver
Creating peer0.org2.example.com ... done
Creating orderer.example.com ... done
Creating peer0.org1.example.com ... done
Creating cli ... done
IMAGE COMMAND PORTS NAMES
fabric-tools "/bin/bash" cli
fabric-peer "start" 7051->7051/tcp peer0org1
fabric-orderer "orderer" 7050->7050/tcp, 7053->7053/tcp orderer
fabric-peer "start" 7051/tcp, 9051->9051/tcp peer0org2
```

A.3.2 Creating a channel

We can use the script to construct a Fabric channel for transactions between Org1 and Org2 now that we have peer and orderer nodes functioning on our workstation. Channels are a private layer of communication between members of a network. Channels can only be used by organizations that have been invited to the channel and are invisible to other network participants. Every channel has its own blockchain ledger. Organizations that have been invited to "join" their peers on the channel in order to keep the channel ledger and validate the channel transactions. You can use the network.sh script to create a channel between Org1 and Org2 and join their peers to the channel. Run the following command to create a channel with the default name of mychannel:

```
$ ./network.sh createChannel
```

If the command was successful, you can see the following message printed in your logs:

```
Channel 'mychannel' joined
```

A.3.3 Starting a chaincode on the channel

After you've established a channel, you may begin interacting with the channel ledger using smart contracts. The business logic that governs assets on the blockchain ledger is included in smart contracts. When deploying a chaincode to a channel, channel participants must agree on a chaincode specification that defines chaincode governance. The chaincode specification can be submitted to the channel after the required number of organizations agree, and the chaincode is ready to use.

After you've created a channel with network.sh, you may start a chaincode on it with the following command:

```
$ ./network.sh deployCC -ccn basic -ccp ../chaincode-go -ccl go
```

The deployCC subcommand will install the asset-transfer (basic) chaincode on peers0.org1 and peer0.org2 before deploying the chaincode on the channel indicated by the channel flag (or mychannel if no channel is specified). If you are deploying a chaincode for the first time, the script will install the chaincode dependencies. The language flag -ccl can be used to install the chaincode in Go, Typescript, or Javascript. The asset-transfer (basic) chaincode is located in the fabric-samples directory in the asset-transfer-basic folder. This category contains sample chaincode that is offered as an example and is used in tutorials to showcase Fabric capabilities.

A.3.4 Interacting with the network

When you've started the test network, you may communicate with it using the peer CLI. You may use the peer CLI to invoke deployed smart contracts, update channels, or install and deploy new smart contracts. Use the following command to add those binaries to your CLI Path:

```
$ export
   PATH=$GOPATH/src/github.com/hyperledger/fabric/build/bin:$PATH
```

You also need to set the FABRIC_CFG_PATH to point to the core.yaml file in the fabric repository:

```
$ export
    FABRIC_CFG_PATH=$GOPATH/src/github.com/hyperledger/fabric/config
```

You can now set the environment variables that allow you to operate the peer CLI as Org1:

```
# Environment variables for Org1

$ export CORE_PEER_TLS_ENABLED=true
$ export CORE_PEER_LOCALMSPID="Org1MSP"
$ export
    CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/
org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
$ export
    CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/
org1.example.com/users/Admin@org1.example.com/msp
$ export CORE_PEER_ADDRESS=localhost:7051
```

The CORE_PEER_TLS_ROOTCERT_FILE and CORE_PEER_MSPCONFIGPATH environment variables point to the Org1 crypto material in the organizations folder. Run the following command to initialize the ledger with assets.

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride
    orderer.example.com --tls --cafile
    "${PWD}/organizations/ordererOrganizations/example.com/orderers/
orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C
    mychannel -n basic --peerAddresses localhost:7051
    --tlsRootCertFiles
    "${PWD}/organizations/peerOrganizations/org1.example.com/
peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses
    localhost:9051 --tlsRootCertFiles
    "${PWD}/organizations/peerOrganizations/org2.example.com/
peers/peer0.org2.example.com/tls/ca.crt" -c
    '{"function":"InitLedger","Args":[]}'
```

If successful, you should see output similar to the following example:

```
-> INFO 001 Chaincode invoke successful. result: status:200
```

You can now query the ledger from your CLI. Run the following command to get the list of assets that were added to your channel ledger:

```
peer chaincode query -C mychannel -n basic -c
    '{"Args":["GetAllAssets"]}'
```

If successful, you should see the following output:

```
[
  {"ID": "asset1", "color": "blue", "size": 5, "owner": "Tomoko",
     "appraisedValue": 300},
  {"ID": "asset2", "color": "red", "size": 5, "owner": "Brad",
     "appraisedValue": 400},
  {"ID": "asset3", "color": "green", "size": 10, "owner": "Jin␣Soo",
     "appraisedValue": 500},
```

```
  {"ID": "asset4", "color": "yellow", "size": 10, "owner": "Max",
      "appraisedValue": 600},
  {"ID": "asset5", "color": "black", "size": 15, "owner": "Adriana",
      "appraisedValue": 700},
  {"ID": "asset6", "color": "white", "size": 15, "owner": "Michel",
      "appraisedValue": 800}
]
```

A.3.5 Bring down the network

When you are finished using the test network, you can bring down the network with the following command:

```
$ ./network.sh down
```

The command will stop and remove the node and chaincode containers, as well as erase the organization's crypto material and chaincode images from your Docker Registry. The program also removes prior runs' channel artifacts and docker volumes, allowing you to run./network.sh up again if there were any issues.

Appendix B

Developer's Manual

This Appendix describes the implementation choices, outlining the main changes to the Hyperledger Fabric source code that led to the thesis work.

B.1 Prerequisites

To develop Hyperledger Fabric on a Linux-based operating system such as Ubuntu, Debian, or CentOS, you will need to have the following prerequisites [26]:

- Install the latest version of git if it is not already installed. (git version 2.17.1)
- Install the latest version of cURL if it is not already installed. (curl 7.58.0)
- Install the latest version of Docker if it is not already installed. (Docker version 23.0.1, build a5ee5b1)
  - Once installed, confirm that the latest versions of both Docker and Docker Compose executables were installed.
  - Make sure the Docker daemon is running.
  - Add your user to the Docker group.
- Install the latest version of Go if it is not already installed. (go version go1.19.3)
  - $GOPATH and $GOPATH/bin are added to $PATH

For Linux platforms, including WSL2 on Windows, also required are various build tools such as gnu-make and C compiler. On ubuntu and it's derivatives, you can install the required toolset by using the command:

```
$ sudo apt install build-essential
```

Other distributions may already have the appropriate tools installed or provide a convenient way to install the various build tools. At this point we can clone the official repository:

```
$ mkdir -p $GOPATH/src/github.com/hyperledger/
$ cd $GOPATH/src/github.com/hyperledger/
$ git clone https://github.com/hyperledger/fabric.git
$ git checkout 12625f54535c5fd691cd50911497c8f4e750006b
```

B.2 Integrating Post Quantum Cryptography into Golang

The changes we wanted to implement in the Hyperledger fabric framework required the integration of post quantum algorithms. This was made possible by the use of the liboqs library. To install liboqs (Open Quantum Safe) library on a Linux OS, you need to have the following dependencies:

```
$ sudo apt install astyle cmake gcc ninja-build libssl-dev
    python3-pytest python3-pytest-xdist unzip xsltproc doxygen
    graphviz python3-yaml valgrind
```

Get the source:

```
$ git clone -b main https://github.com/open-quantum-safe/liboqs.git
$ cd liboqs
```

and build:

```
$ mkdir build && cd build
$ cmake -GNinja .. -DBUILD_SHARED_LIBS=ON
$ ninja
$ sudo ninja install
```

The framework is written in golang, which is why it is needed in liboqs' go-wrapper. liboqs-go is a Go package. The project contains the following files and directories:

- *oqs/oqs.go*: main package file for the wrapper
- *.config/liboqs.pc*: pkg-config configuration file needed by cgo
- *examples*: usage examples, including a client/server KEM over TCP/IP
- *oqstests*: unit tests

Assuming liboqs.so.* were installed in /usr/local/lib (true if you ran *sudo ninja install* after building liboqs). You may need to set the LD_LIBRARY_PATH environment variable to point to the path to liboqs' library directory, e.g.

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

Download/clone the liboqs-go wrapper repository in the directory of your choice, e.g. $HOME, by typing in a terminal/console:

```
$ cd $HOME && git clone https://github.com/open-quantum-safe/liboqs-go
```

Next, you must modify the following lines in $HOME/liboqs-go/.config/liboqs.pc:

```
LIBOQS_INCLUDE_DIR=/usr/local/include
LIBOQS_LIB_DIR=/usr/local/lib
```

so they correspond to your C liboqs include/lib installation directories.

Finally, you must add/append the $HOME/liboqs-go/.config directory to the PKG_CONFIG_PATH environment variable, i.e.

```
$ export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$HOME/liboqs-go/.config
```

If all the previous configurations were successful, it is now time to clone the PQCrypto repository:

```
$ https://github.com/CosimoMichelagnoli/PQCrypto.git
```

Then delete crypto folder, put PQCrypto into $GOROOT/src/ folder, and rename PQCrypto to crypto. At this point, if everything is correct, the framework can be built regularly with the make command without being aware of the modification of the crypto library.

B.3 Blockchain Cryptographic Service Provider

To make Hyperledger Fabric quantum-safe, the Blockchain Cryptographic Service Provider (CSP) needs to be updated to use quantum-safe algorithms for cryptographic functions that are currently based on classical cryptographic algorithms that are vulnerable to quantum attacks. To make signature algorithms quantum-safe in Hyperledger Fabric's Blockchain Cryptographic Service Provider (BCCSP), you must work on the source code related to signature algorithms in the BCCSP package. The source code files for signature algorithms can be found in the BCCSP package at the following location: *fabric/bccsp*. The package define several interfaces, struct, and methods that allow developer to specify options related to cryptographic operation such as key generation, signature, and encryption. The options can be used to customize the behavior of the BCCSP according to the user's specific requirements and use cases. The main interfaces are the following:

- *KeyGenOpts*: This interface defines the options for key generation, such as the key size and the key type.
- *ImportKeyOpts*: This interface allows users to specify options for importing cryptographic keys.
- *Signer*: This interface defines methods for signing and verify messages using a cryptographic key.
- *KeyGenerator*: This interface defines methods for generating cryptographic keys.
- *GoPublicKeyImportOptsKeyImporter*: This interface specifies options for importing public keys into the BCCSP key store.

The *opts.go* file in the BCCSP package of Hyperledger Fabric contains various options and configurations that can be used to customize the behavior of BCCSP. *ImportKeyOpts* is a type of option that allows users to specify options for importing cryptographic keys. The *ImportKeyOpts* interface can be implemented by various structs to provide different options for importing keys. For example, the *X509PublicKeyImportOpts* struct implements the *ImportKeyOpts* interface and provides options for importing public keys from X.509 certificates. Overall, the *ImportKeyOpts* option provides a flexible way for users to specify options for importing cryptographic key according to their specific needs. In our specific case, we extended the *opts.go* file by adding the structs *DILITHIUMGoPublicKeyImportOpts* and FALCONGoPublicKeyImportOpts. The *KeyGenOpts* interface has been implemented for both Dilithium and falcon by means of two dedicated files, *dilithiumopts.go* and *falconopts.go* with the corresponding structs *DILITHIUMKeyGenOpts* and *FALCONKeyGenOpts* inside them.

B.4 Docker Images

Docker images are an important part of the Hyperledger Fabric infrastructure because they enable for the development of isolated and portable environments in which to operate Fabric nodes and applications. Hyperledger Fabric is a distributed ledger platform created with container technology for enterprise use. Fabric components and apps are packaged and distributed using Docker images, which are subsequently deployed as container instances.

Each Fabric component, such as peer nodes, orderer nodes, and certificate authorities, can be packed into a Docker image that includes all the component's dependencies and configurations. Fabric components may be easily deployed and scaled across numerous environments by using Docker images, as the same image can be utilized across multiple deployments.

When the container is created, the binary files required for the component are created. For example, in a peer's container, the peer's source code and dependencies will be built with the make command, including the BCCSP package. The PQCypto library must therefore also be added to the container, otherwise it will not be possible to build the code of the peer and other network components. This problem was solved by creating a custom Docker file and uploading it to Docker Hub. We then modified the affected DockerFiles so that they would take the built libraries from our customized image.

```
FROM golang:1.18.7-alpine3.16 as golang
RUN apk add --no-cache bash binutils-gold pkgconfig gcc git astyle
    cmake ninja libressl-dev py3-pytest py3-pytest-xdist zip libxslt
    doxygen graphviz py3-yaml valgrind
WORKDIR /
RUN git clone -b main https://github.com/open-quantum-safe/liboqs.git \
       && git clone https://github.com/open-quantum-safe/liboqs-go

WORKDIR /liboqs
RUN mkdir build

WORKDIR /liboqs/build
RUN cmake -GNinja .. -DBUILD_SHARED_LIBS=ON -DOQS_BUILD_ONLY_LIB=ON
RUN ninja && ninja install

ENV LD_LIBRARY_PATH /usr/local/lib
ENV PKG_CONFIG_PATH /liboqs-go/.config

WORKDIR /usr/local/go/src/
RUN rm -r crypto
RUN git clone https://github.com/CosimoMichelagnoli/PQCrypto.git crypto
```

B.5 Cryptogen

Cryptogen is a Hyperledger Fabric command-line utility that generates cryptographic content for Fabric network nodes and organizations.

Each node and organization in a Fabric network must have a distinct digital identity, which is represented by cryptographic key pairs. These key pairs are used to protect

network communications, validate transactions, and control access. Cryptogen automates the development of cryptographic material for nodes and organizations, making it easier to generate these key pairs. Cryptogen, in particular, can produce:

- Public and private key pairs for each node in the network, including peer nodes, orderer nodes, and certificate authorities.
- Digital certificates for each node and organization, which are used to verify the authenticity of network communications and transactions.
- Certificate revocation lists (CRLs), which are used to revoke compromised or outdated certificates.

These cryptographic materials are generated by Cryptogen based on a configuration file that determines the number of nodes and organizations in the network, as well as their naming standards and cryptographic algorithms. The created materials are saved as files that the Fabric network components can use during network configuration. Overall, Cryptogen simplifies the process of generating cryptographic material for Hyperledger Fabric nodes and organizations, decreasing the possibility of errors and assuring the Fabric network's security and integrity.

The main changes made to the source code of this binary are the definition of a struct for Dilithium and Falcon as a signer. This struct allows the generated keys to be saved. Keys that are subsequently saved in the keystore and X.509 certificates with the genCertificateDILITHIUM/FALCON() methods. The number of changes has been drastically reduced thanks to the functionality offered by the PQCrypto library, which offers X.509 and TLS management for NIST-selected algorithms.