Security Assessment of Authentication and Authorization Mechanisms in Ethereum, Quorum, Hyperledger Fabric and Corda

Marie-Jeanne LAGARDE
marie-jeanne.lagarde@epfl.ch, EPFL
Decentralized and Distributed Systems Lab (DEDIS)
Under the supervision of
Pr. Bryan FORD
Youness BENCHRIFA
at Orange Cyberdefense during the autumn semester of 2018.

March 14, 2019

Abstract

Recently, enterprises have been seeing an increasing interest in blockchain and are beginning to investigate possible industry use cases. In this context, the question of blockchain security does not only include consensus schemes that have been widely studied but also access control. The topics of authentication and authorization mechanisms are key concepts of access control schemes that have not been properly addressed with regard to popular industry oriented blockchain technologies.

We propose a detailed description of authentication and authorization mechanisms existing in Ethereum, Quorum, Hyperledger Fabric and Corda. Information is retrieved from official documentation and completed by several experiments that test the expected behaviors, describe unclear or lacking implementation aspects, evaluate the frequency of some attacks and disclose official sample scripts default parameters. We define a general threat model for blockchain authentication and authorization schemes and assess the security of each platform with regard to this model. Finally, we provide hardening guidelines in order to help users prevent the stated vulnerabilities when possible.

Keywords: Blockchain, Ethereum, Quorum, Hyperledger Fabric, Corda, authentication, authorization, access control, membership protocol, remote access, security vulnerabilities, attacks.

Contents

List of Figures

List of Tables

Acknowledgment

I would like to thank Professor Bryan Ford for his support and useful advice, and for accepting to supervise me. This master thesis gave me a great opportunity to feed my curiosity and to explore blockchain security in deep.

I further want to thank Anne Kent, Sylvain Brost and Louis-Alexis Brenac for offering me the chance to conduct this master thesis in industry at Orange Cyberdefense in the Conseil&Audit department, Youness Benchrifa for being a thoughtful supervisor, and all my colleagues for their flawless support.

Finally, I address a special thank to my family and my friends who were always supportive during my studies and showed interest in my work.

Chapter 1

Introduction

1.1 Motivation

In 2008, Satoshi Nakamo published the paper *Bitcoin: A Peer-to-Peer Electronic Cash System* [1] which presents a solution to the double-spending issue for digital currency via a distributed database called blockchain that combines cryptography, game theory, and computer science. This solution opened the way to the development of numerous blockchain platforms seeking to answer different needs. Even though it is still in early adoption stage, blockchain technology enjoys significant success in industry as it allows companies to cut costs, and to increase traceability and transparency. However, these newly designed technologies are far from being mature, and almost every implementation still presents vulnerabilities. Further, the lack of documentation along with the blockchain designers' expectations of their platform being used by qualified experts often lead to insecure deployments. The sole purpose of this technology makes it an ideal target for adversaries aiming to access and manipulate valuable and often confidential data. This research has been written in the context of a master thesis in industry in the company Orange Cyberdefense (OCD), it aims to identify potential vulnerabilities regarding **authentication and authorization schemes** of the industry most popular blockchain platforms.

1.2 Scope

This thesis focuses solely on four blockchain platforms: Ethereum, Quorum, Hyperledger Fabric and Corda. For each platform, we investigates the design, the implementation and the security of the authentication and authorization schemes. The scope of this thesis includes the evaluation of access control default parameters for selected official sample scripts, remote access authentication protocols and corresponding hardening guidelines. Out of scope of this thesis is any security issue affecting other platforms than the ones listed above or other components of the listed platforms. Further, we do not consider physical attacks (such as shoulder surfing) and social engineering attacks in the scope of this thesis.

1.3 Research Objectives

The objective of this thesis is to provide a clear overview of **the design, implementation and vulnerabilities of the authentication and authorization schemes** existing in Ethereum, Quorum, Hyperledger Fabric and Corda. Further, it aims to provide **hardening guidelines** to prevent their exploitation.

1.4 Research Questions

As mentioned before, this research addresses the issue of authentication and authorization in the context of four blockchain platforms: Ethereum, Quorum, Hyperledger Fabric and Corda. As a consequence, we aim to answer for each platform three main questions with a few sub-questions:

- **Research Question 1 (RQ1)**: How are designed and implemented the authentication and authorization mechanisms?
  - **Research Sub-question 1 (RSQ1)**: How is the network access permissioned and how are the roles granted?
  - **Research Sub-question 2 (RSQ2)**: How are transactions senders authenticated, and do transactions require specific authorization to be sent and accessed?
  - **Research Sub-question 3 (RSQ3)**: Does the platform offer remote APIs? If yes, how are authorizations granted and how is the remote user authenticated?
- **Research Question 2(RQ2)**: What are the vulnerabilities of the previously described authentication and authorization mechanisms?
- **Research Question 3 (RQ3)**: Which recommendations shall be given with regard to the security of the authentication and authorization mechanisms?

1.5 Thesis Outline

The remaining content of the thesis is organized in the following way: first, chapter 2 gives a theoretical background of blockchain technology, public key cryptography and identification, authentication and authorization concepts. In chapter 3, we provide an overview of the current related research work while in chapter 4 we describe the methodology and justifications for the thesis structure, the platforms choice and the conducted experiments. Chapter 5 addresses the first research question (RQ1) by describing the implementation and the design of the authentication and authorization scheme of each platform. Chapter 6 answers the second research question (RQ2) by analyzing the vulnerabilities of the previously described authentication and authorization schemes. The methodology and results of the experiments that helped answer the previous two questions are described in chapter 7. Chapter 8 consequently presents hardening guidelines corresponding to previously highlighted vulnerabilities in response to research question 3 (RQ3) and chapter 9 discusses general limitations and gives a summary of further research work. Finally, chapter 10 concludes the

research reported in this thesis. If you are already familiar with the theoretical background, you may jump directly to Chapter 3.

Chapter 2

Theoretical Background

This chapter presents theoretical background which is required to understand the security analysis and the experiments conducted in this thesis. The following concepts are explained: blockchain and permissioning models, public key cryptography, identity authentication and authorization.

2.1 General Blockchain Overlook

2.1.1 Main Components

The concept of the blockchain answers the need for a trustworthy ledger that does not rely on any designated centralized authority. This technology implements a simple solution which consists in distributing trust by providing each participant with an identical copy of the ledger. It mainly relies on three components which are described below: a distributed ledger, a consensus protocol and a membership protocol.

Distributed Ledger
A distributed ledger in the context of blockchain is a growing list of records, called blocks, which are linked one to another using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data as described in Figure 2.1. Once recorded, the data in any given block cannot be altered retroactively without alteration of all subsequent blocks (each subsequent block is a function of the previous one).

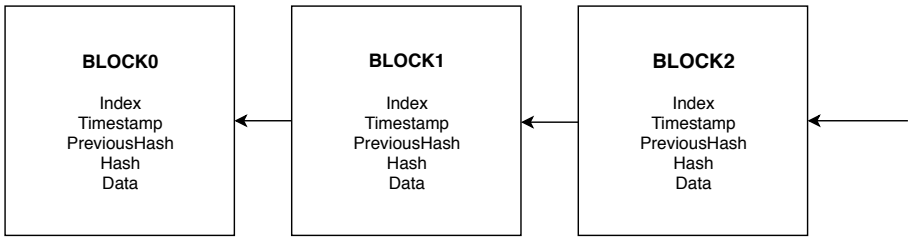

Figure 2.1: Example distributed ledger structure

Consensus Protocol
In order to keep a fixed replicated ledger, blockchain requires participants called **nodes** to agree using a consensus. There exist several approaches to the consensus problem with each providing different levels of guarantees against faulty and malicious participants.

Membership Protocol
Blockchain requires a membership protocol to maintain the distributed participant network. This protocol defines how participants join, leave and maintain interaction with the distributed network.

2.1.2 Blockchain Permissioning Models

We distinguish two models: permissioned and permissionless blockchains.

Permissionless Blockchain
A permissionless blockchain is open to anyone to read, initiate transactions and participate in the consensus.

Permissioned Blockchain
Permissioned blockchains maintain an access control layer to allow certain actions to be performed only by certain identifiable participants.

2.2 Public Key Cryptography

2.2.1 Definition

Definition 1: *Public-key cryptography, or asymmetric cryptography, is a cryptographic system that uses pairs of keys: public keys which may be disseminated widely, and private keys which are known only to the owner. The generation of such keys depends on cryptographic algorithms based on mathematical problems to producing one-way functions. Effective security only requires keeping the private key private; the public key can be openly distributed without compromising security. [2]*

Figure 2.2 illustrates how Bob uses Alice's public key to send her an encrypted document and how Alice decrypts it using her private key.

2.2.2 Public Key Certificate

Definition 2: *In public key cryptography, a certificate is an electronic document that aims to prove the ownership of a public key.*

2.2.3 Public Key Infrastructure

Definition 3: *A public key infrastructure (PKI) binds public keys to entities, enables other entities to verify public key bindings, and provides the services needed for ongoing management of keys in a distributed system. [3]*

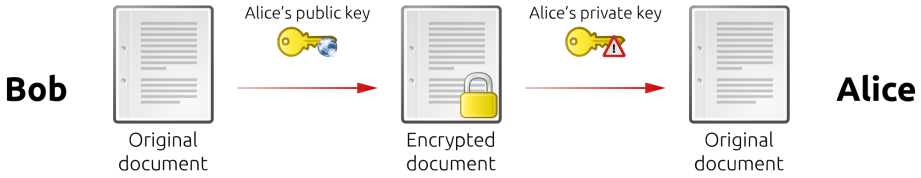

Figure 2.2: Public key cryptography scheme. Extracted from Asymmetric Cryptography by Wikimedia Commons, URL: `https://commons.wikimedia.org/wiki/File:Asymmetric_Cryptography.svg`. Retrieved March 5, 2019

PKI aims to provide trust in distributed environments. Each PKI possesses a Certificate Authority (CA) which issues a public key certificate for a requested identity, confirming that the identity has the appropriate credentials. A digital certificate typically includes the party's public key, optionally attributes of the party holding the corresponding private key (e.g. roles), an operational period along with the CA's own digital signature. The CA also issues Certificate Revocation List (CRL) where are listed certificates that have been revoked.

2.2.4 Digital Signature

Definition 4: *A digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender (authentication), that the sender cannot deny having sent the message (non-repudiation), and that the message was not altered in transit (integrity). [4]*

The use of public key cryptosystems to provide digital signatures was first suggested by Diffie and Hellman [5]. The original scheme is the following: the sender encrypts a hash of the message with his private key and concatenates it with the message, optionally including his certificate. When receiving the packet, the sender decrypts the received hash with the sender's public key, and compares it with the hash of the received message. If they are equal, then the message has been sent by the known sender, it was not altered and the sender cannot deny having sent it.

2.3 Identification, Authentication and Authorization

The definitions that follow originate from the ICSA whitepaper of M. E. Kabay [6].

2.3.1 Identification

Definition 1 *Identification is the process that enables recognition of a user described to an automated data processing system. This is generally by the use of unique machine-readable names.*

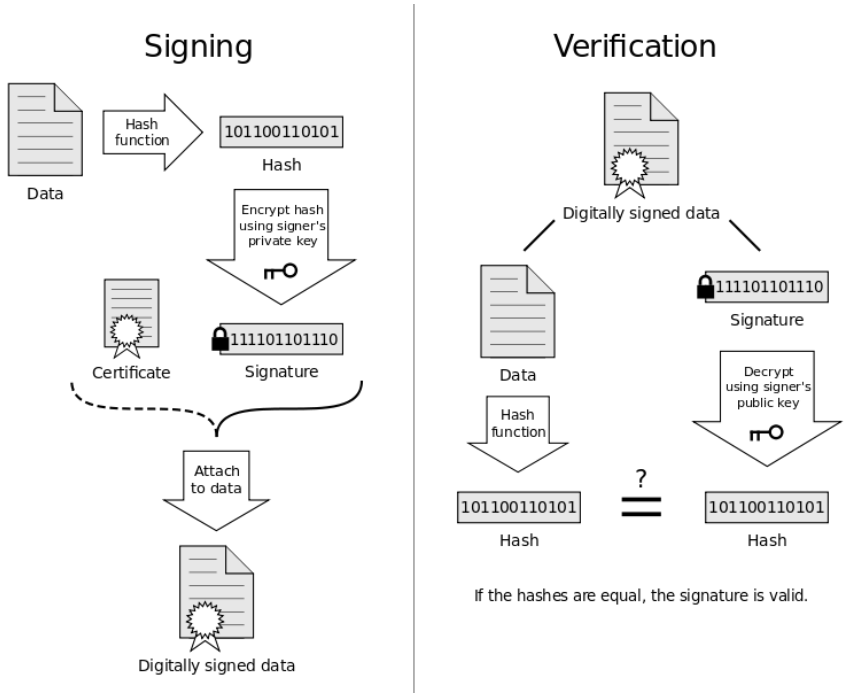

Figure 2.3: Digital Certificate Scheme. Extracted from *Digital Signature* by Wikimedia Commons, URL: `https://commons.wikimedia.org/wiki/File:Digital_Signature_diagram.svg`. Retrieved March 5, 2019

In clear, identification is the ability to uniquely identify a user in the system. In the real world, this would relate to the combination of the first name and last name when talking about the identity of an individual. In a computer network, it is often a username. An identity needs to be ascertained to verify that the user claiming the identity is not lying, this process is called authentication.

2.3.2 Authentication

Definition 2 *Authentication is the act of verifying the claimed identity of an individual, station or originator.*

To continue the previous metaphor, authentication in real life would correspond to an individual showing his ID card to prove its identity. In a computer network, verification relies on a factor which varies depending on the applications and makes it possible to positively verify an identity. The factors are listed here under.

- **What the user knows:** an information that only the user possesses (e.g. a password)
- **What the user is:** this factor refers to biometric property of the user

(e.g. retina or voice recognition)

- **What the user has:** an asset that only the user has in his possession (e.g. a key or a certificate)

Authentication methods for each factor are widely described in the Cert-EU Security Whitepaper [7].

2.3.3 Authorization

Definition 3 *Authorization is the granting to a user, program, or process the right of access.*

Some systems require authorization on top of authentication for users to access a resource, perform an action or access a file. It is defined through an access control policy which specifies user access rights and privileges to resources. Once authenticated, each action performed by the user is subject to the access control policy and either accepted or denied.

Chapter 3

Related work

The following work has been gathered via a thorough research described in the methodology section (4.2).

3.1 General Authentication Attacks

Subsection 2.3.2 lists the possible factors on which depends the authentication: knowledge, possession, inference (biometric-based). Those factors are often combined in two-factor or three-factor authentication schemes. We look for scientific literature to evaluate the possible attacks, leaving apart the inference factor as none of our selected platform uses it.
Knowledge-based identification being the most widely used scheme, it has also been the more exposed to attackers and is a large area of interest of researchers. Among those researchers, Towhidi et. al. [8] describe knowledge-based authentication attacks according to Common Attack Pattern Enumeration and Classification (CAPEC) standards. We later use the categories listed in the paper to classify the potential attacks on knowledge-based authentication schemes existing in the platforms we study: **password brute force, sniffing attack, spoofing attacks, authentication bypass**. The paper also describes social engineering attacks, physical security attacks and shoulder attacks that we do not consider in the scope of our thesis.
A more general scholar research paper from Jesudoss A [9] summarizes possible authentication attacks and proposes various countermeasures. Figure 3.1 presents an extract of the summary of attacks and countermeasures presented in the paper that is relevant for our thesis. Additionally, the paper from Maria Nickolova and Eugene Nickolov [10] about threat model for user security in e-learning system also provide a generic authentication section that describes the case when **an attacker masquerades as a legitimate end user: brute force attacks, dictionary attacks, login spoofing attacks, key management attacks, replay attacks, man-in-the-middle attacks, session-hijacking attacks and non-repudiation attacks**. Finally, Dobromir Todorovs book [11] on *Mechanisms of User Identification and Authentication* provides us with useful definitions.

Attack	Countermeasure	Authentication Mechanism	Advantage	Disadvantage	Additional Hardware
Eavesdropping	Encryption	SSL	Secured Online	Costly	No
	Token-based - CAS	RubyCAS, SecurID	Single-Sign on	Less control over navigation control	No
	Authentication Protocols	Kerberos	Single-Sign on, mutual authentication	Migrating users to Kerberos database is difficult	No
Man-in-the-Middle Attack	Encryption	SSL	Confidentiality	Performance	No
	Mutual Trust	CA – Certificates	Speed & Security	Expired & Cost	No
	Hashing	HMAC	No need of SSL	Inconsistent	No
Replay Attacks	Dynamic unique data such as TimeStamp, OTP, Nonce	OTP/NONCE SSL	OTP - Two factor authentication	OTP - dependent on addl technology, spoofable	Yes
Phishing Attacks	Mutual Authentication	Digital Certificates	Protects from Impostor	Vendor support, algorithm strength	No
	Avoid download from unreliable source	Digital Signatures	Non-repudiation, prevents imposter	Compatibility, cost	No
	Check for Padlock icon	HTTPS	Confidentiality	Performance	No
Brute Force Attacks	Tarpitting	Biometric	Unique	Data gets changed	Yes
	IDS	Honeypot	Simplicity	Risk	No
	Test Human	CAPTCHA	Avoids bots	Difficult to read	No
Dictionary Attack	Strong passwords	Hashed with SALT value	Makes guessing harder	Slow	No

Figure 3.1: Comparative Analysis of Attacks, Countermeasures, Authentication Methods - Merits & Demerits, extracted from *A survey on authentication attacks and countermeasures in a distributed environment* by Jesudoss A. URL: `http://www.ijcse.com/docs/INDJCSE14-05-02-061.pdf`. Retrieved March 5, 2019

3.2 Common Vulnerabilities of Authentication and Authorization Schemes in Distributed Networks

Identifying related works on authentication and authorization protocols in distributed networks was a first step for a top-down approach as blockchain is a distributed system at its core. Distributed networks being a popular subject on computer science, the challenge of securing their authentication and authorization schemes has been an area of broad interest. A comprehensive specification for security in a distributed system is provided by Gasser et. al. [12]: it concentrates on **distributed systems specific vulnerabilities, defines the term of secure channel** which was previously introduced by Birrell et. al. [13] and enumerates its properties and possible architectures. Further, it explores the concepts of **delegation, certificates and revocation for distributed systems**. A more recent paper from Akhter and Haque [14] analyzes the security of different types of authentication and authorization protocols and defines the potential security threats as being: **interception, interruption, modification, fabrication**. An unauthorized party accessing a distributed network is called an interception, a data or service made unavailable or being destroyed is called an interruption, an unauthorized change of data or service is called a

modification and an unwanted creation of data or job is called a fabrication. This literature constitutes a baseline for defining a threat model for blockchain authentication and authorization schemes.

3.3 Public Cryptography Vulnerabilities

As all the blockchain platforms we study use public cryptography in their authentication process, we focus on finding scientific material about public cryptography potential vulnerabilities and challenges. Similarly to distributed systems, public-key cryptography has received considerable attention over the past decades and numerous papers and security analysis have been produced, from general purpose to specific applications. David Pointcheval [15] assesses the security of practical schemes together with their reductionist security proofs. His critical analysis of **Plaintext-RSA and DL-Based digital signature scheme** provides helpful content for our thesis. The paper points out the possible vulnerability of Plain-RSA against existential forgery and possible non-generic attack on elliptic curve (seen later in 3.4.2). His analysis is based on the definition of possible digital signature scheme attacks described by Goldwasser et. al. [16] which distinguishes **key-only attacks** where the attacker only knows the sender's public key and **message attacks** where the attacker is able to examine pair of combined message and corresponding digital signatures before his attack. Further, it discerns **totalbreak** where the attacker discovers the victim's trapdoor information (private key), **universal forgery** where the attacker possesses an algorithm that produces correct digital signatures, **selective forgery** where the attacker is able to get a correct digital signature for a chosen message and **existential forgery** where the attacker is able to forge a digital signature for at least a message that he does not choose.

3.4 Research on Vulnerabilities of Blockchain Authentication and Authorization Schemes

3.4.1 Generic Work

Nowadays, most scientific literature on blockchain security deal with consensus schemes and language flaws. Authentication and authorization is often mentioned as it is the case in Prof. Bryan Ford article [17], but rarely carefully analyzed.

3.4.2 Platform Specific Work

Popular permissionless blockchain platforms such as Bitcoin or Ethereum have grabbed the attention of the researchers within the last years. Indeed, those platforms are mostly used for financial exchanges and represent an opportunity for hackers to make considerable profits. As a consequence, financial exchange blockchain platforms have been largely targeted and those attacks made it possible to identify several vulnerabilities including in the mechanisms that we are looking at in our thesis. Here follows an exhaustive description of related work we found relevant for the security of authentication and authorization schemes

regarding the platforms Ethereum, Quorum, Hyperledger Fabric and Corda.
Ethereum was the most documented platform among the four, which can be quickly understood when reading the previous explanations. Some **vulnerabilities have been pointed out in the Ethereum ECDSA algorithm** [18]. They rely on the choice of **the curve SECP256K1** (which is also used in Bitcoin) used for the key generation which suffers from both mathematical and implementation vulnerabilities. Another Ethereum vulnerability of the client remote access has been identified after the conduction of several hacks: it resides in the often **weak or misconfigured remote APIs authentication and authorization schemes** (Remote Protocol Control (RPC) and Web Services (WS)) which are exploited to remotely access the node's client [19].
As far as permissioned blockchain platforms are concerned, few security reviews have been found. Plausible reasons for this absence of relevant studies are: restricted number of operational deployments, lack of feedback resulting from enterprise deployments and a smaller number of users.
Regarding Hyperledger Fabric, a few reviews have been found including one on security: a *Security Assessment Management Report* was conducted by Nettitude on v1.1 [20], Christan Cachin conducted on his own a brief study of the Architecture of the Hyperledger Fabric blockchain platform [21] with a short overview of used certificate mechanisms, and participated along with Androulaki et. al. in a broader architecture review of the platform **trust and membership service** [22]. To the best of our knowledge, Corda and Quorum do not benefit from independent security peer reviews, and similarly to Hyperledger Fabric our main source of information to analyze the security of their authentication and authorization schemes emanates from their own official documentation.

Chapter 4

Methodology

4.1 Roadmap

This chapter describes the methodology followed to produce this thesis. Figure 4.1 presents a roadmap which explains the different phases. This thesis emanates from the will of the French company Orange Cyberdefense which aimed to conduct a broad study on the security of main blockchain technologies used in industry. The thesis shall focus on two specific interesting security aspects of four chosen platforms and demonstrates the findings. As described, the first phase consists in **broad a literature search** (explained in detail in section 4.2) with main goals to identify the four industry most popular platforms and two essential security aspects on which the security analysis shall focus. The second phase deals with **the choice of platforms and security aspects to study in deep** (explained later on 4.3 and 4.4). Next, **the authentication and authorization schemes of each chosen platform are carefully studied**, based on the official documentation and additional research. To complete this study, **numerous experiments are conducted** to provide lacking information about the schemes and to test uncertain behaviors. Subsequently, **a fine grain study of schemes vulnerabilities based on a threat model that we define is conducted for each chosen technology**. This finally leads to **production of hardening guidelines** for more secure deployments, to a **critical discussion** of the limitation of our thesis, and to a **conclusion** on the security of the authentication and authorization schemes of Ethereum, Quorum, Hyperledger Fabric and Corda.

4.2 Literature Search

A large-scale literature search was conducted to first identify the industry most popular blockchain platforms and then to select two crucial areas of interest. Finally, finer-grained research was carried out to gather relevant material for the security analysis. The identification of industry most popular platforms relied mostly on **surveys conducted by large-scaled industry consulting and research companies** such as Gartner, Ernst&Young and on **economic newspapers** such as Forbes (detailed bellow in 4.3). **Scientific content** on

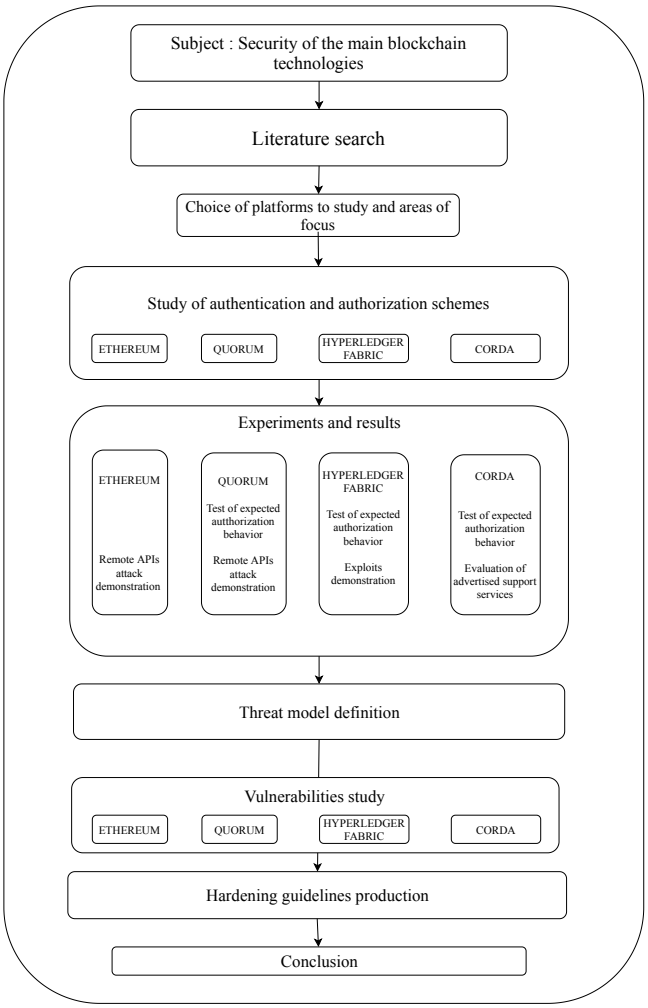

Figure 4.1: Thesis methodology roadmap

blockchain security was searched on both Google Scholar and IEEE. Many papers referenced other useful publications from various authors which enabled the construction of a large knowledge basis. Permissioned blockchain being a least popular subject than open blockchains, addition material was searched on both **technology review blogs (for instance Medium[1]) and security vulnerabilities forums**. Finally, the **official platform documentation and GitHub projects** were gathered to further determine the expected system behavior.

4.3 Justification for Selected Platforms

The goal of this master thesis is to assess the security of the main blockchain technologies from an industry perspective. Conscious of the impossibility of being exhaustive when covering this subject, this research required as a first step to choose on which blockchain initiatives to focus. The criteria for this selection were the following: **high potential for industry purposes, current adoption rate, forecasted long-term adoption and community support**. When analysing the market, it appears that industry initiatives play a preponderant role. As a consequence, we decided to focus on the three main industry initiatives highlighted by the EY *Global blockchain benchmarking study* [23]. Each industry initiative supports one or several platforms among which we choose to study: Hyperledger Fabric in the context of the Hyperledger Project supported by The Linux Foundation, Corda supported by the company R3 and finally both the permissionless platform Ethereum and the permissioned platform Quorum promoted by the Ethereum Enterprise Alliance. The later choice was led by the idea that even if Ethereum is not properly industry oriented, we will be able to take advantage of the similarities between Ethereum and Quorum to transpose to Quorum vulnerabilities that have been already identified in Ethereum. Indeed, the security of Ethereum has been more widely studied due to its popularity. Further, Ethereum is also used by large industry companies to deploy business applications, as is it the case for the insurance company AXA with the Fizzy application[2].

4.4 Justification for Selected Areas of Interest

Trade and industrial secrets are crucial to preserve an economic advantage against competitors in industry. Thus, the prevention of industrial espionage brings the following question to the industrial tables: "Who can access and modify the company data and how secure are the protection systems?" Transposing this question into a blockchain concern, an essential area of interest appears to be **the security of permissioning** systems thought the authentication and authorization processes. Even though the analysis of the system tolerance to faulty process and bug freeness remains necessary, we notice that this area has already been more widely studied as more scientific material about consensus tolerance to faulty processes is available compared to our chosen areas of interest. Further, the design and implementation of the permissioned platforms we

[1] https://medium.com/
[2] https://fizzy.axa/fr/

chose are not very well detailed apart in their official documentation which is often imprecise. In brief, the belief that this thesis would constitute a consequent input to securing deployments of blockchain initiatives in industry leads to the selection of authentication and authorization as areas of interest.

4.5 Justification for Deployment Setups

In order to conduct experiments, selection of platform versions and sample scripts was required. The experiments conducted in the thesis are based on the two following assumptions:

- **Hypothesis 1 (H1):** The most deployed platform versions are the most likely to be targeted by attackers.
- **Hypothesis 2 (H2):** Users tend to adapt their systems from existing official sample scripts rather than build them from scratch.

As a consequence, the experiments are based on **popular demos deployed on the latest available release of the most widely used version of the platform**. A difficulty was to assess the popularity of an example script against an other, as GitHub does not display the number of downloads. Subsequently, we looked for the most referenced demos on technology blogs and forums and any other additional source of information available. Finally, the deployments were **constrained by the available computer resources**. The experiments were conducted either on a MacBook Pro 2013 on macOS Sierra with 4Gb of RAM or on a ThinkPad on Windows 7 with 20Go of RAM deployed behind a company firewall. Consequently, some problems were encountered during the deployment of containers and virtualization technologies (Vagrant, Oracle VM VirtualBox) and local deployment were privileged when possible.

4.6 Justification for Selected Tools

Most of the experiments described in Chapter 7 assess the system expected behavior on each of the platform official sample script and do not require additional tools to provide results. However, for the purpose of some experiments which **evaluate the resistance of studied mechanisms against eavesdropping and replay attacks**, a sniffing tool is required. **Wireshark** is a free and open source packet analyzer which was chosen for its intuitive use along with fine-grained filter options. Finally, graphs were constructed using **Excel** (Microsoft 2010) sheets.

Chapter 5

Authentication and Authorization Schemes

For each platform, we present an **overview of relevant components** and then provide a detailed description of both design and implementation of the existing authentication and authorization schemes. This description is divided in three subsections corresponding to our three research sub-questions: **network permissioning or peer authenticated communication, transactions and remote APIs**. In distributed networks, nodes coordinate their actions and communicate with each other using message-passing, and thus participants authentication relies on message authentication. Some information provided originate from experiments which are detailed in Chapter 7.

5.1 Ethereum

Ethereum is an open-source permissionless blockchain platform that any developer can use to run distributed applications. First, we describe the mechanisms used for nodes authenticated communication, we further present the transaction authentication schemes to address research sub-question 2, and finally the remote APIs authentication and authorization schemes are described answering research sub-question 3. Note that research sub-question 1 is ignored as Ethereum is permissionless. Technical description is based on Ethereum Yellow Paper [24].

5.1.1 Summary

Table 5.1 summarizes the types of the authentication and authorization schemes existing in Ethereum.

5.1.2 Description of Relevant Components

Nodes

Ethereum is structured as a peer-to-peer network where peers which ensure network transactions verification are called nodes.

Node authentication	N\A (permissionless, authenticated communication uses ECC)
Node roles granting	N\A (no roles are defined)
Transaction sender authentication	key-based
Transaction sender authorization	account owner does not require specific authorization
Transaction receiver authentication	N\A (all transactions are public)
Transaction receiver authorization	N\A (all transactions are public)
General remote user authentication	non-existing
Account owner remote authentication	passphrase-based
Remote user authorizations	non-existing, available methods depend on enabled modules

Table 5.1: Summary of Ethereum authentication and authorization schemes

Accounts

Ethereum distinguishes two types of accounts which are used for transactions:

- Contract accounts: controlled by their associated smart contract code, code execution is triggered by transactions or messages (calls) received from other contracts.
- Externally owned accounts (EOA): associated with a pair of keys, they do not rely on any contract code and are used to store the cryptocurrency called Ether.

Each account has a state associated with it, and **the global state of the network is made of the accounts states all together.**

Client

An Ethereum client is required to run a node. There are numerous client versions, three official implementations which possess the same interface coded in different languages are available: Geth (go-ethereum), Eth (C++), Pyethapp (python). As of this date, Geth is the most widely used client (48%) according to the website ethnodes[1] and we decide to conduct our analysis from this client perspective.

Remote APIs

Ethereum official clients provide ways to remotely access a node. Any action performed on a node client can be remotely initiated depending on the enabled modules (see Remote APIs authorization (5.1.5)).
The modules are designed in the following way and aim to be enabled independently:

- `eth`: provides blockchain related options (get transaction receipt, etc.)
- `miner`: provides control functions over mining and DAG
- `net`: provides network management functions (listening, etc.)
- `personal`: provides wallet related API (transfer fund, etc.)
- `ssh`: provides message functions

[1] https://www.ethernodes.org/network/1

- `web3`: provides client version
- `admin`: provides various functions for managing a Geth node (adding a peer, start and stop RPC, etc.)
- `db`: provides functions to interact with the database
- `txpool`: provides information about transaction pool status

In Geth, remote access is implemented via **JSON-RPC** and **web-socket (WS)** which are a **plain text and stateless** remote procedure call protocols.

5.1.3 Authenticated Channel for Message-passing

Ethereum is a **permissionless network** and thus nodes can establish communication with each other without the need of authenticating to a trusted entity. However, an authenticated channel is set up during nodes handshake.

Design
Authenticated communication between nodes is maintained via **Elliptic Curve Cryptography** using **recoverable ECDSA**, material for authenticated communication is derived during the handshake.
Each node is associated with a unique pair of keys generated using the **elliptical curve secp256k1**, the node public key used to verify the signature is called enode.
Internode communication relies on The RLPx Transport Protocol[2] which provides encrypted communication via authenticated channels using both nodes' key pair and a 16-byte long shared secret key. The asymmetric encryption method ECIES (Elliptic Curve Integrated Encryption Scheme) is used in the RLPx handshake to generate authentication key and encryption key in the following way (note that K_b is the receiver public key and k_b the receiver private key):

(1) A random number r and corresponding elliptic curve public key $R = r * G$ and a random initialization vector iv are picked, where G is a generator of the elliptic curve secp256k1
(2) A shared secret $S = P_x$ is computed where $(P_x, P_y) = r * K_B$
(3) Encryption key K_e and authentication key K_m are derived $k_E||k_M = KDF(S, 32)$ where $KDF(k, len)$ is the NIST SP 800-56 Concatenation Key Derivation Function
(4) The encrypted message $R||iv||AES(k_E, iv, m)||MAC(k_M, iv||c)$ is sent, where AES is the AES-128 encryption function in CTR mode and MAC is HMAC using the SHA-256 hash function
(5) The receiver derives the shared key $S = P_x$ from $(P_x, P_y) = k_B * R$
(6) The receiver derives the encryption key K_e and authentication key K_m from $k_E||k_M = KDF(S, 32)$

Implementation
Node keys are generated via the node client, in Ethereum official client Geth,

[2] https://github.com/ethereum/devp2p/blob/master/rlpx.md

this is done through the command `bootnode`.
The RLPx Transport Protocol is implemented on top of TCP.

5.1.4 Transaction Sender Authentication

In this subsection, we aim to address the second research sub-question (RSQ2): in Ethereum, transaction emanates from an account (and not from a node). Note that the sender **does not require specific authorization to invoke a transaction** once the ownership of the account's private key has been proven (authentication). Further, Ethereum being a permissionless network, anyone is able to read the content of a transaction that has been sent to the network. We are left with the question of transaction authentication to answer: how does a user authenticate himself as the owner of a given account?

Design
Ethereum relies on **key-based authentication using recoverable ECDSA signature** to verify the identity of the transaction sender.
Indeed, each identity is associated with a unique account address for which the pair of keys has been generated using the **elliptical curve secp256k1**. The sender is authenticated if a correct account address with respect to the global network state is derived from the transaction signature.

The account address is then generated by taking the right most 20 bytes of the Keccack-256 hash of the generated public key.
To each Ethereum private key, p_{r}, is associated an Ethereum address $A(p_{\mathrm{r}})$ which is defined as the right most 160 bits of the Keccak hash of the corresponding public key:

$$A(p_{\mathrm{r}}) = \mathcal{B}_{96..255}\big(\texttt{KEC}\big(\texttt{ECDSAPUBKEY}(p_{\mathrm{r}})\big)\big) \tag{5.1}$$

Implementation
When receiving a transaction, the account address is extracted by invoking the `ECDSARECOVER` method.

5.1.5 Remote APIs Authentication and Authorization Schemes

In this subsection, we investigate our third research sub-question (RSQ3) which copes with remote APIs authentication and authorization schemes. As explained previously, we choose to describe the remote client access scheme in Ethereum Geth client.

Authentication for Wallet Related Actions

Design
Client remote access **does not require user authentication except for wallet related actions which rely on a passphrase authentication**. To perform a remote operation on a given account, the account needs to be unlocked: it can either be unlocked by default or protected with a passphrase.

Implementation
The module *personal* provides a wallet related API and remotely enables a user to perform all operations on a given account such as querying the balance or transferring funds. The command *unlock*0 is used to unlock by default an account when starting a node. If protected by a passphrase, the remote user needs to unlock the account with the method: *unlock_account*. This method requires as argument an address and a passphrase, and an optional duration. By default, unencrypted key will be held in memory for 300 seconds once the account unlocked. During this time lapse, the account can be used to sign transactions and transfer funds without reentering the passphrase.

Authorization

Design
The user **authorization to remotely call a function of a given module depends on which modules are enabled on the node client**. As no user authentication is required, any function belonging to an enabled module can be accessed via a remote call.

Implementation
In Geth, in order to activate HTTP JSON-RPC and define the enabled modules, the user either has to start Geth with the RPC flag or to authorize it from the console using the `admin.startRPC(rpcaddr, rpcport, rpcpis, rpccorsdomains)` command. The required parameters are the following:

- `rpcaddr`: listening interface. If '0.0.0.0', the node listens on all interfaces
- `rpcport`: defines the default port (8545)
- `rpcapi`: defines which modules of the API are authorized
- `rpccorsdomain`: can be used to bypass the same origin policy (it is a default policy that only allows machine from the same domain)
- `unlock`: permanently unlocks the account attached to this node if set to 0

Similarly, WS can be started with the console or using RPC via `admin_startWS` method using the following parameters:

- `wsaddr`: defines WS-RPC server listening interface
- `wsport`: defines WS-RPC server listening port
- `wsapi`: defines which modules of the API are authorized
- `wsorigins`: defines origins from which to accept web-sockets requests

We assess the default parameters in Ethereum experiment 1 (7.1.3). The results show that **by default RPC and WS APIs are disabled, and that modules enabled by default when starting the services are `eth`, `net`, `web3`.**

5.2 Quorum

Quorum is the open-source permissioned version of the Ethereum platform. It has been adapted to implement private transactions and a permissioning system to control the network access, thus its structure is in many ways similar to Ethereum. The following section analyzes the authentication and authorization processes in logical order, first, relevant components are described providing a better overview of the system, second the question of network permissioning is addressed to determine how a node can participate in the network (RSQ1), then we deal with the mechanisms for transaction authentication and authorization as Quorum allows private transactions (RSQ2) and finally we study the remote APIs authorization schemes (RSQ3). Technical information and schemes presented originate from Quorum Whitepaper [25], Quorum Wiki[3] and Tessera Wiki[4].

5.2.1 Summary

Table 5.2 summarizes the types of the authentication and authorization schemes existing in Quorum.

Message sender authentication	key-based, optionally certificate-based if TLS CA enabled
Node roles granting	N\A (no node roles defined)
Transaction sender authentication	key-based
Transaction sender authorization	account owner does not require specific authorization
Transaction receiver authorization	ACL
General remote user authentication	non-existing
Account owner remote authentication	passphrase-based
Remote user authorizations	non-existing, available methods depend on enabled modules

Table 5.2: Summary of Quorum authentication and authorization schemes

5.2.2 Description of Relevant Components

Quorum being derived from Ethereum, accounts have the exact same structure. To avoid redundancy, one should refer to subsection 5.1.2.
As two different implementations of the Transaction Manager and Enclave exist, we chose to study the most recent one named Tessera. The relationship between nodes, Tessera Transaction Manager and Tessera Enclave is depicted in Figure 5.1 (note that the figure has been modified from the original as the transaction manager and enclave software used is Tessera) and explained here under.

Quorum node

Quorum node is an Ethereum Geth node which has been modified to additionally handle private transactions.

Tessera Transaction Manager

Tessera Transaction Manager is responsible for transaction privacy. It stores and allows access to encrypted transaction data, exchanges encrypted payloads

[3]https://github.com/jpmorganchase/quorum/wiki
[4]https://github.com/jpmorganchase/tessera/wiki

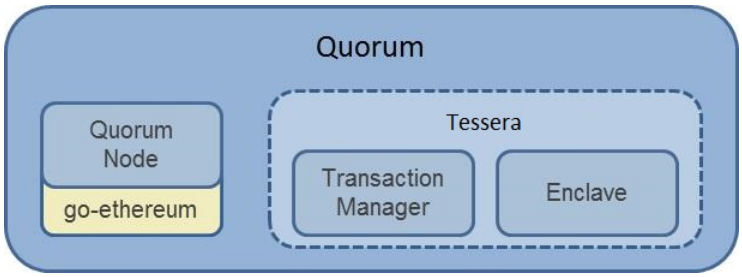

Figure 5.1: Quorum logical architecture diagram. Adapted from Quorum GitHub wiki, URL: `https://github.com/jpmorganchase/quorum/wiki`. Retrieved March 5, 2019

with other participant's Transaction Managers but does not have access to any sensitive private keys. It utilizes the Enclave for cryptographic functionality.

Tessera Enclave

In Quorum, in order to achieve a separation of concerns, much of the cryptographic work including symmetric key generation and data encryption/decryption is delegated to the Enclave. The Enclave manages the encryption/decryption in an isolated way by holding private keys.

Remote APIs

Quorum node's client is a slightly modified version of Ethereum Geth client and possesses the same remote APIs. To avoid redundancy, one shall refer to Ethereum remote APIs (5.1.2). The only difference relies on the additional modules provided for Quorum which handles specific functionalities such as consensus scheme related commands:

- `quorum`: additional API for quorum
- `raft` : addition API for raft consensus
- `istanbul` : additional API for istanbul consensus

5.2.3 Network Permissioning

This subsection addresses research sub-question 1. Quorum being a permissioned blockchain, it ensures that only authenticated nodes can participate in message exchanges. Thus, we describe how message origin authentication is ensured, and how nodes are granted permission or revoked from the network. Note that nodes do not have special roles in Quorum.

Message Sender Authentication

Design
Sender node authentication relies on **key-based authentication using recoverable ECDSA signatures**: each user is assigned a pair of asymmetric

keys generated using the elliptical curve secp256k1 and possess a list of other network nodes' public keys. Messages exchanged are signed using recoverable ECDSA which allows the receiver to **extract the sender's public key from the message signature and to compare it to the list of other permissioned node's public key.** If an entry matches, the node is authenticated by the receiver, otherwise the connection is refused.

Optionally, TLS can be enabled requiring either mutual, client or server authentication. Several modes are available: TOFU, CA and whitelist (TOFU&CA can be combined). The keys for identity and TLS can be either reused or different. If using Tessera to generate TLS keys, the used format is **RSA**. The trustmodes are defined in the following ways:

- TOFU stands for Trust-on-first-use. Only the first node that connects identifying as a certain host will be allowed to connect as the same host in the future. It relies on **key-based authentication**.
- CA: Only nodes with a valid certificate and chain of trust are allowed to connect and thus it requires **certificate-based authentication**
- WHITELIST: Only nodes that have previously connected to this node and have been added to the `knownclient` file will be allowed to connect. This uses **key-based authentication**.

Implementation
The **permissioning option** is implemented at node level and **shall be identical for all nodes in the network**, i.e., the same network cannot contain permissionless and permissioned configured nodes. Quorum Experiment 1 (7.2.3) shows a nonfunctional behavior for this case.
To enable permissioning, each node needs to be started with the `permissioned` command line flag. In Quorum, the permissioning is managed through the `permissioned − node.json` file which lists enodes in the following style: `enode://remoteky1@ip1:port1`, where remotekey is the public key associated with the node. Every node in the network is required to have its own copy of this file, and **the order in which the nodes are listed must be the same** as shown by the Quorum experiment 1.
For TLS keys, the user can either import its own key pair or as well use Tessera to generate a new one. Those key pairs can be provided in several ways: plain text, inline allowing for the use of Argon2 password-secured private keys by including the corresponding Argon2 settings in the additional config, with Azure key vault or Hashicorp Vault key pairs, and finally using file system.
To generate TLS new key pairs, the command Tessera `tessera − keygen` must be used. When looking at the source code, we discover that this key generation relies on RSA with a key size of 2048.
When dealing with externally generated pairs of keys, no information is given about the supported formats. However, when looking on available forums, we can make the hypothesis that it supports both Elliptic Curve Cryptography (ECC) and RSA schemes. The configuration of TLS communication requirements is handled through Tessera in the `configfile`, and authenticated communication is enabled when the TLS parameter is set to `STRICT`.

Granting Permission

In order to grant permission to a new node, its public key must be added in each node's list. However, the official documentation does not provide any information about the dynamicity of this process. Quorum Experiment 2 (7.2.4) proves that a new node can be dynamically granted access to the network. When TLS is enabled, the new node needs to be authenticated when establishing a connection with another network node. Similarly, no information is given about the dynamicity of TLS modes when adding a new node, as a consequence some experiments are conducted and summarized in table 5.3. Quorum experiment 4 (7.2.5) highlights the fact that whitelist mode does not allow dynamic node addition. TOFU mode enables any new node to connect as long as its IP has not been associated prior to another key pair, and CA modes require the new node to provide a valid certificate (signed by the CA root of trust).

TLS mode	CA	CA & TOFU	TOFU	Whitelist
Dynamically add a new node	Yes	Yes	Yes	No
Dynamically revoke a node	Yes via CRL	Yes via CRL	No	No

Table 5.3: Results of the experiments on TLS modes for dynamic node addition and revocation capabilities

Revoking a Node

Similarly to granting access to a new node, revoking a node requires a deletion on each node's list and the dynamicity of this process has not been cleared either. Quorum Experiment 3 (7.2.4) proves that a node can be dynamically revoked from the network. If TLS is enabled, a revoked node is required to have its TLS identity also revoked. Once more, no information is given about the dynamicity of this process, and some experiments are conducted to provide an answer. The results are also presented in table 5.3. As shown by Quorum Experiment 5 (7.2.5) and 8 (7.2.8), whitelist and TOFU modes do not allow for dynamic revocation. CA mode relies on CRL to handle node revocation.

5.2.4 Transactions

This subsection addresses research sub-question 2: in Quorum, similarly to Ethereum, transaction emanates from an account (and not from a node), and thus the sender is required to be authenticated as the account owner.

Transaction Sender Authentication

Sender authentication, similarly to Ethereum, relies on **key-based authentication using recoverable ECDSA signatures** which are used to verify that the sender owns the account from which the transaction emanates. Design and implementation are identical to Ethereum and for more information one shall report to Ethereum Transaction description (5.1.4).

Authorization to Access a Transaction

Quorum introduces the concept of transaction privacy and distinguishes public transactions from private transactions. Public transactions are identical to Ethereum transaction and accessible to anyone in the network whereas **private transaction are meant to be accessible by a restricted number of nodes defined by the sender.**

Design
With each transaction is associated an ACL which contains the public keys of the authorized nodes. The mechanism for authorized nodes to recover the transaction relies on public key cryptography, the following example extracted from the official documentation[5] describes a private transaction between A and B and is depicted in Figure 5.2:

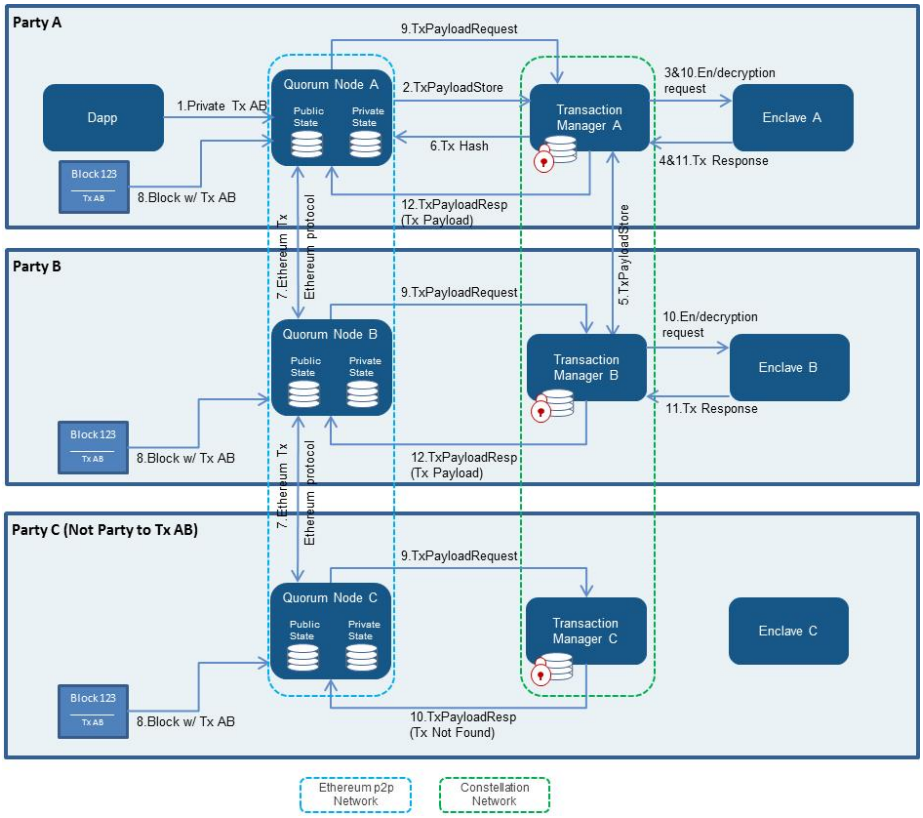

Figure 5.2: Quorum private transaction processing. Extracted from Quorum GitHub wiki, URL: `https://github.com/jpmorganchase/quorum/wiki`. Retrieved March 5, 2019

1. Party A sends a Transaction to their Quorum Node specifying which nodes should have access to it

[5]https://github.com/jpmorganchase/quorum/wiki/Transaction-Processing

2. Party A's Quorum Node passes the Transaction on to its paired Transaction Manager, requesting for it to store the Transaction payload

3. Party A's Transaction Manager makes a call to its associated Enclave to validate the sender and encrypt the payload

4. Party A's Enclave checks the private key for Party A and, once validated, performs the Transaction conversion. This entails:

    - i. Generating a symmetric key and a random Nonce.
    - ii. Encrypting the Transaction payload and Nonce with the symmetric key from i.
    - iii. Calculating the hash of the encrypted payload from ii.
    - iv. Iterating through the list of Transaction recipients, in this case Parties A and B, and encrypting the symmetric key previously generated with the recipient's public key (PGP encryption)
    - v. Returning the encrypted payload from step ii., the hash from step iii. and the encrypted keys (for each recipient) from step iv. to the Transaction Manager

5. Party A's Transaction manager then stores the encrypted payload (encrypted with the symmetric key) and encrypted symmetric key using the hash as the index, and then transfers the hash, encrypted payload, and encrypted symmetric key that has been encrypted with Party B's public key to Party B's Transaction Manager. Party B's Transaction Manager responds with an Ack/Nack response. Note that if Party A does not receive a response/receives a Nack from Party B then the Transaction will not be propagated to the network. It is a prerequisite for the recipients to store the communicated payload.

6. Once the data transmission to Party B's Transaction Manager has been successful, Party A's Transaction Manager returns the hash to the Quorum Node which then replaces the Transaction's original payload with that hash, and changes the transaction's V value to 37 or 38, which will indicate to other nodes that this hash represents a private transaction with an associated encrypted payload as opposed to a public transaction with nonsensical byte code.

7. The transaction is then propagated to the rest of the network

8. A block containing Transaction AB is created and distributed to each Party on the network.

9. In processing the block, all Parties will attempt to process the Transaction by making a call to their local Transaction Manager to determine if they hold the Transaction (using the hash as the index to look up).

10. A party that does not hold the Transaction will receive a negative answer. Party A and B will look up the hash in their local Transaction Managers and identify that they do hold the Transaction. Each will then make a call to its Enclave, passing in the encrypted Payload, encrypted symmetric key and signature.

11. (The Enclave validates the signature and then decrypts the symmetric key using the Party's private key that is held in The Enclave, decrypts the Transaction Payload using the now-revealed symmetric key and returns the decrypted payload to the Transaction Manager.

12. The Transaction Managers for Parties A and B then send the decrypted payload to the EVM for contract code execution. This execution will update the state in the Quorum Node's Private StateDB only.

Implementation
Access control list is implemented via the transaction field `PrivateFor` in the transaction payloads which contains the authorized node's public keys.
The cryptographic implementation relies either on the jnacl or kalium NaCl depending on the user choice during the deployment, the hash algorithm used is SHA3-512 and the network protocol is standard Ethereum P2P Protocol. To identify a private transaction, a payload field `v` is set to the values 37 or 38.

5.2.5 Remote APIs Authentication and Authorization Schemes

Remote APIs authentication and authorization mechanisms are identical to the one described in the Ethereum section. To avoid redundancy, one shall refer to the Ethereum description (5.1.5) to be provided the answer to research sub-question 3.
Quorum Experiment 9 (7.2.9) exhibits RPC and WS APIs default parameters of Quorum official sample script 7nodes: **RPC API is enabled by default on all nodes, with no restrictions on which connections to accept** and the initialization forces the unlocking of the accounts whereas WS is disabled.

5.3 Hyperledger Fabric

Hyperledger is an open-source permissioned blockchain platform which intends to provide a modular architecture. The following section first presents the architecture of the relevant components to our analysis, it then details the network permissioning (RSQ1), and the authorization mechanism which regulates transactions (RQ2). As any communication between nodes relies on the remote APIs, the research sub-question 3 is already addressed within the first two research questions. Technical details and schemes come from Hyperledger Fabric official documentation[6] and the official Hyperledger Fabric GitHub page[7].

5.3.1 Summary

Table 5.4 summarizes the types of the authentication and authorization schemes existing in Hyperledger Fabric which are described in the following section.

Message sender authentication	Certificate-based
Node roles granting	ABAC
Transaction authorization	ACL
Transaction receiver authentication	N\A (all transactions are accessible by channel members)
Transaction receiver authorization	N\A (all transactions are accessible by channel members)
General remote user authentication	Certificate-based
Remote user authorizations	ABAC

Table 5.4: Summary of Hyperledger Fabric authentication and authorization schemes

5.3.2 Description of Relevant Components

Nodes

There are three types of nodes in Hyperledger Fabric:

- Client or submitting-client: a client that submits an actual transaction-invocation to the endorsers, and broadcasts transaction-proposals to the ordering service.
- Peer: a node that commits transactions and maintains the state and a copy of the ledger. Besides, peers can have a special endorser role.
- Ordering-service-node or orderer: a node running the communication service that implements a delivery guarantee, such as atomic or total order broadcast.

Chaincode

The term chaincode refers to a smart contract.

[6] https://hyperledger-fabric.readthedocs.io/en/release-1.4/
[7] https://github.com/hyperledger/fabric

Transaction

Two types of transactions described below exist in Hyperledger Fabric.

- Deploy transactions: create new chaincode and take a program as parameters. When a deploy transaction executes successfully, the chaincode has been installed on the blockchain.
- Invoke transactions: perform an operation in the context of previously deployed chaincode. An invoke transaction refers to a chaincode and to one of its provided functions.

MSP design

MSP offers an abstraction of a membership operation architecture. It manages the authentication and authorization in the network, and should be considered at different levels:

- Network MSP: defines the networks participant through a list of the organizations MSPs and provide their specific **authorization** (ex: authorization to create a channel).
- Channel MSP: define administrative and participatory rights at the channel level. Every organization participating in a channel must have an MSP defined for it. Peers and orderers on a channel will all share the same view of channel MSPs, and will therefore be able to correctly **authenticate the channel participants**.
- Local MSP (orderers, peers, clients): allow the user side to **authenticate itself** in its transactions as a member of a channel, or as **the owner of a specific role (authorization)** into the system (an org admin, for example, in configuration transactions). Local MSPs are defined for clients (users) and for nodes (peers and orderers).

Hyperledger Fabric provides an optional certificate authority which can be used to generate certificates and keys (described below). However it can be replaced by any CA that can generate ECDSA certificates.

Fabric CA module

Certificate issuing is handled (optionally) through Fabric CA. Fabric CA architecture is described in Figure 5.3.

Communication
Communication between the Fabric CA server and clients (which can either be done via Hyperledger Fabric CA client or through one of the Fabric SDKs) uses REST APIs. TLS is optional, on the server side, the following modes are available: NoClientCert, RequestClientCert, RequireAnyClientCert, VerifyClientCertIfGiven, and RequireAndVerifyClientCert.

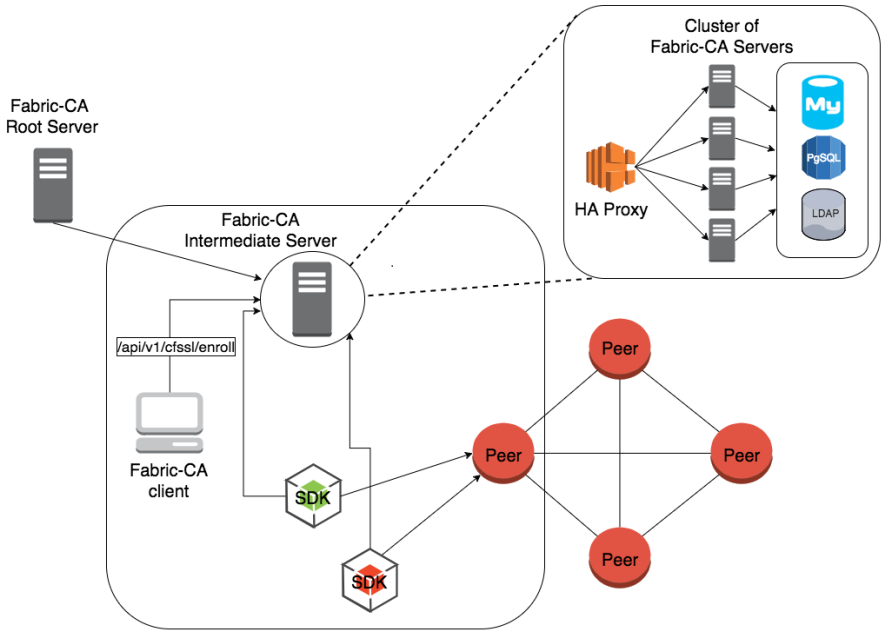

Figure 5.3: Fabric CA overview. Extracted from *Fabric CA User's Guide*, URL: `https://hyperledger-fabric-ca.readthedocs.io/en/release-1.4/users-guide.html`. Retrieved March 5, 2019

Fabric CA provides the following features:

- Registration of identities, or connects to LDAP as the user registry;
- Issuance of certificates;
- Certificate renewal and revocation.

As described in Figure 5.3, LDAP can be configured, in this case, the information is kept in LDAP directly. The CA server structure is flexible, as each CA can either be a root CA or an intermediate CA, and there is no limitation on the number of layers of intermediate CAs.

Fabric CA Initialization
The bootstrap identity should be used to register the first client identity. The user can choose to either generate its own password or let the machine generate one. Once the peer identity successfully registered, the given peer can be enrolled thanks to its ID and password. In order to register a new identity, the identity performing the registration should be currently enrolled and must have the proper authority to register as described in Table 5.5. First, the new identity requires to be registered with an ID and a password, and then to be enrolled to get a valid certificate.

Name	Type
hf.Registrar.Role	List
hf.Registrar.DelegateRole	List
hf.Registrar.Attributes	List
hr.GenCRL	Boolean
hr.Revoker	Boolean
hr.AffiliationMgr	Boolean
hf.IntermediateCA	Boolean

Table 5.5: Fabric CA Identity attributes

The following conditions shall be met :

- The identity that registers the new identity should have "hf.Registrar.Roles" attributes along with the value of the type of attribute being registered (peer, app, user...)
- The affiliation of the registrar shall be a prefix or equal to the affiliation of the registered. If not mentioned, the identity registered is given the affiliation of the registrar.

5.3.3 Channel Permissioning

This subsection partially answers research sub-question 1. In the context of a permissioned distributed network, message origin authentication is required to ensure that only authorized node can participate in the channel. Thus, we describe message sender authentication, permission granting and revoking.

Message Sender Authentication

Design
Message origin authentication protocol uses **ECC-based X.509 certificate and optionally TLS certificates**. For a message sender to be authenticated by a given channel, **it must present a certificate with a verifiable path to exactly one of the roots of trust certificates**. Key generation options if Fabric CA is used are depicted in 5.6. For a given node, TLS is either disabled, one-way (server only) or two ways (server and client). The keys for identity and TLS certificates can either be different or reused.

size	curve	Signature Algorithm
256	prime256v1	ecdsa-with-SHA256
384	secp384r1	ecdsa-with-SHA384
521	secp521r1	ecdsa-with-SHA512

Table 5.6: Key size options for key generation using Fabric CA

Implementation
The list of channel root CAs and intermediate CAs is included in the genesis block of the channel which is broadcast to all channel participants. TLS 1.2 is

used and downgrading to TLS 1.0 is disabled. By default, as shown in Hyperledger Fabric Experiment 1 (7.3.3) TLS client authentication is enabled but client authentication is disabled. As a consequence, when a node or client acts as client, TLS will not require him to be authenticated.

Permission Granting

Design
In order to join a channel, a new node must submit a CSR to one of the channel's root or intermediate CAs. Further, it needs to configure its local MSP with the following information:

- Its root of trust certificate
- (Optional) Intermediate CA certificates
- The certificate of the MSP administrator
- A list of CRL corresponding to each previously listed CA (root or intermediate)
- The TLS root of trust certificate
- (Optional) Intermediate TLS CA
- (Optional) A list of Organizational Unit that member of this MSP must contain in their certificates

Implementation
The process to be provided the required certificate depends on the CAs implementations. Once in possession of its certificate, the new node is required to **build its local MSP by hosting a folder** which contains a file and six subfolders as described in the official documentation[8]:

- A folder admincerts to include PEM files each corresponding to an administrator certificate
- A folder cacerts to include PEM files each corresponding to a root CA's certificate
- (Optional) A folder intermediatecerts to include PEM files each corresponding to an intermediate CAs certificate
- (Optional) A file config.yaml to configure the supported Organizational Units and identity classifications
- (Optional) A folder CRLs to include the considered CRLs
- A folder keystore to include a PEM file with the nodes signing key;
- A folder signcerts to include a PEM file with the nodes X.509 certificate
- (Optional) A folder tlscacerts to include PEM files each corresponding to a TLS root CAs certificate

[8] https://hyperledger-fabric.readthedocs.io/en/release-1.3/msp.html

- (Optional) A folder tlsintermediatecerts to include PEM files each corresponding to an intermediate TLS CAs certificate

Revoking Permission

Design
Node revocation relies on **CRLs** which must be updated at both nodes and channel levels. They must be signed from the channel's root or intermediate that issues the certificate of the node which is revoked.

Implementation
PEM encoded **CRL** file are placed in the CRLs folder of every MSPs, and the channel admin is in charge of updating the channel **CRL**. If using Fabric CA, The fabric-ca-client `gencrl` command can be used to generate a CRL.

5.3.4 Nodes Roles Granting

This subsection completes the answer already provided to research sub-question 1 by explaining node roles granting.

Design
Hyperledger Fabric channel optionally defines specific roles within the channel using **Attribute-Based Access Control** (ABAC) which is based upon identity's attributes contained in certificates.

Implementation
To enable this feature, the file `config.yaml` needs to be configured identically in each local MSP to describe identity classifications (most often it separates peers and clients). Attribute name and values contained in the certificates are then extracted to make an access control decision. Further, the certificate `ROLE` attribute can be used to confer administrative rights at the channel level.

5.3.5 Channel Authentication

A channel do not authenticate itself to its participants. HyperLedger Fabric Experiment 2 (7.3.4) depicts the fact that a channel MSP can be maliciously changed without one of the member organizations being notified. This happens when the majority of organizations decide to eject nodes from another organization, modify the channel MSP and re-add the previously removed nodes.

5.3.6 Transactions Authorization

As described in the description of relevant components, there are two types of transactions in Hyperledger Fabric: deployment transactions and endorsement transactions. Deployment transactions do not require specific authorization.
Note that within a channel, everyone has access to transactions, thus there is **no authorization scheme to access a given transaction**, transactions are treated as regular messages and thus transaction sender authentication is handled similarly to message sender authentication, we are left with the transaction authorization scheme to address research sub-question 2.

Endorsement Transaction Authorization

Design
In Hyperledger Fabric, endorsement transactions authorization scheme relies on **ACL**s: for the transaction to be authorized, it has to be endorsed according to the endorsement policy of the chaincode invoked.
The endorsement transaction operation flow depicted in Figure 5.3.6 is the following:

1. The client creates a transaction and sends it to endorsing peers of its choice
2. The endorsing peer simulates a transaction and produces an endorsement signature
3. The submitting client collects an endorsement for a transaction and broadcasts it through ordering service
4. The ordering service delivers a transaction to the peers

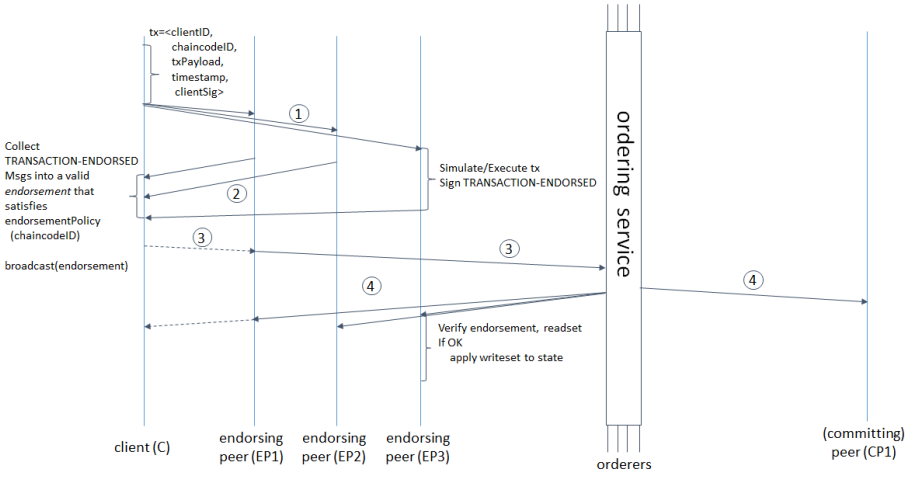

Figure 5.4: Hyperledger Fabric Illustration of one possible transaction flow (common-case path). Extracted from *Hyperledger Fabric Architecture Reference*, URL: `https://hyperledger-fabric.readthedocs.io/en/release-1.4/arch-deep-dive.html`. Retrieved March 5, 2019

Implementation
The chaincode policy that defines the required endorsers is defined when the chaincode is deployed using the -P parameter. As an example the following command line would require both a member of EPFL and a member of Orange-Cyberdefense to endorse the transaction:

```
peer chaincode instantiate -C <channelid> -n testchaincode -P
    "AND('EPFL.member', 'OrangeCyberdefense.member')"
```

The ordering service then verifies via the `VerifyPeerCertificate` method all the required signatures.

5.3.7 Remote API

Irrespective of the kinds of nodes, the endorsers, peers and clients communicate with each other using the gRPC protocol which uses remote APIs. As a consequence, the authentication and authorization schemes existing Hyperledger Fabric remote APIs are the one already described in the section Network and channels access control (5.3.3).

5.4 Corda

Corda is blockchain platform which comes in two flavors: users can either deploy their own network from the open-source code or pay to use the already existing Corda network. To begin with, this section describes the relevant components to our first research questions, then addresses research sub-question 1 (RSQ1) by describing network permissioning and nodes roles granting for both versions of the network. Third, transaction authorization process is explained (RSQ2). Finally, authentication and authorization schemes of Corda remote API are addressed in response to research sub-question 3 (RSQ3). Technical information originates from Corda official documentation[9].

5.4.1 Summary

Table 5.7 summarizes the types of the authentication and authorization schemes existing in Corda.

Message sender authentication	Certificate-based
Node roles granting	ABAC
Transaction sender authentication	Relies on certificate-based node authentication
Transaction authorization	Provided by notaries node
Transaction receiver authorization	N\A (the transactions are only sent to authorized nodes and not broadcast)
General remote user authentication	Password-based
Remote user authorizations	Capability list

Table 5.7: Summary of Corda authentication and authorization schemes

5.4.2 Description of Relevant Components

This subsection describes the relevant components to our first Research Question: the compatibility zone, the CordaX509Utilities class, the transaction scheme, the notion of notaries, the two Corda networks and the remote API.

Compatibility zone

In Corda, a compatibility zone designates a network which is permissioned. It structure is described in Figure 5.5. A Corda network has four types of certificate authorities which are described hereunder:

- The root network CA
- The doorman CA: acts as an intermediate CA
- The node CAs: Each node serves as its own CA in issuing the child certificates that it uses to sign its identity keys and TLS certificates
- The legal identity CAs: Nodes well-known legal identity, apart from signing transactions, can also issue certificates for confidential legal identities

Corda's X509Utilities

Cordas X509Utilities is a module which uses Boucy Castle[10] that is provided to generate key pairs and certificates.

[9] https://docs.corda.net/

[10] A provider for the Java Cryptography Extension (JCE)

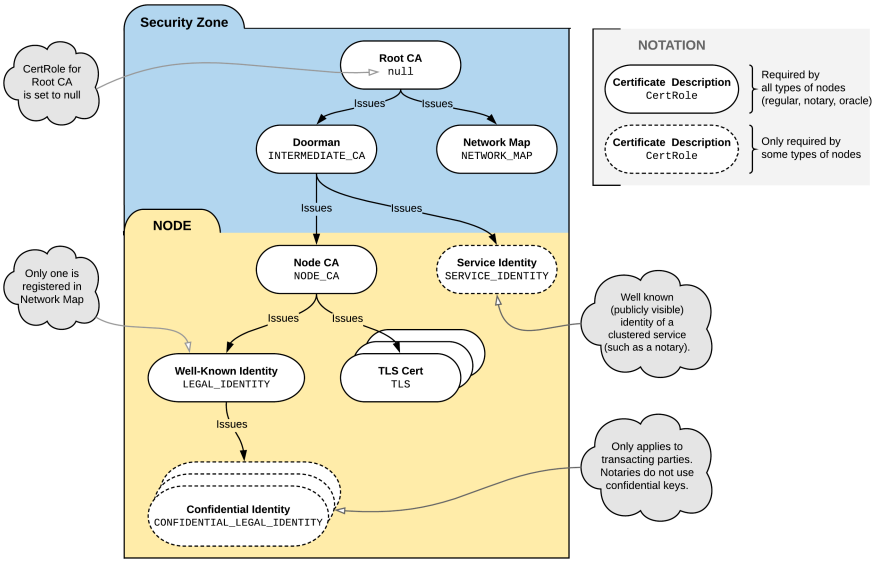

Figure 5.5: Corda permissioning structure. Extracted from Corda Network Certificates, URL: `https://docs.corda.net/permissioning.html`. Retrieved March 5, 2019

Transaction Scheme

Corda follows the UTXO (unspent transaction output) model which is also used in Bitcoin. In brief, each transaction spends outputs from prior transaction and generates zero or new outputs. The ledger is thus updated by marking used transaction outputs as *history*.

Notaries

There exist two types of notaries that validate the transactions:

- Validating: has access to the entire transaction and validate its consistency.
- Non-validating: does not have access to the transaction content but only knows if the input states have not been consumed before.

Independently managed network vs Corda Business network

Corda deployment comes in two flavors: the user either set up his own network by providing it own Root CA and doorman and implementing its network map server, or uses the existing Corda network infrastructure called Corda Business Network which is a publicly available internet of Corda nodes operated by network participants. We try to deploy Corda Business network in Corda Experiment 2 (7.4.4) and we find out that it costs $2500.

Corda Remote APIs

Corda allows remote access to nodes in order for third-party applications to be able to communicate with a given node. It is implemented via a client library that provides a **RPC interface**. This library makes it possible to write clients in any JVM-compatible language.

5.4.3 Network Permissioning

This subsection partially answers research sub-question 1. In the context of a permissioned distributed network, message origin authentication is required to ensure that only authorized node can participate in the network. Thus, we address message sender authentication, permission granting and revoking for both Corda Business network and an independently managed Corda network.

Message Sender Authentication

Design
Corda design integrates **certificate-based authentication using digital signatures** to authenticate message sender: nodes possess both TLS and identity certificates. When receiving an incoming connection, the receiver node verifies the chain of trust of TLS and identity certificates: the signatures of both certificates in the chain must be verified up to the Root CA Certificate. Both TLS and identity keys can be generated with the help of Corda's X509Utilities or imported. The cipher suites supported by Corda's X509Utilities are depicted in table 5.8.

Implementation
Required format for the certificates are the following: an identity X.509v3 certificate and a TLS v1.2 certificate. By default, Corda expects the trust store file (which contains the root CA public key) to be hosted in a certificate folder. Upon the reception of a node message, the receiver verifies the authenticity of the chain of trust with regard to its truststore.

Cipher suite	Default for
Pure EdDSA using the ed25519 curve and SHA-512	identity
ECDSA using the NIST P-256 curve (secp256r1) and SHA-256	root CA, node CA, network map, TLS
ECDSA using the Koblitz k1 curve (secp256k1) and SHA-256	
RSA (3072bit) PKCS1 and SHA-256	
SPHINCS-256 and SHA-512	

Table 5.8: Corda's X509Utilities cipher suites

We analyze the default TLS parameters in Corda Experiment 1 (7.4.3), and find out that by default **mutual authentication is required between nodes**.

Granting Permission

Design
The steps required to be granted the permission to join the network are the following:

1. The new node must **possess the root CA in its truststore, and the address of the doorman and the network map**
2. The node needs to register with its certificate to the doorman by **submitting a certificate signing request (CSR) to obtain a node CA certificate**
3. From the node CA certificate, the node creates and **signs two further certificates, a TLS certificate and a certificate for the node's well known identity**
4. Finally the node builds a node info record containing its address and well-known identity, and **registers it with the network map service**

Implementation for Corda Business network
The implementation of the previously described steps is the following:

1. First, the trust root certificate needs to be requested at by emailing doorman@r3.com
2. A CSR along with participant's pair of keys are sent to the network doorman when launching a node for the first time. The participant then needs to agree on the terms of use.
3. The doorman returns the signed certificate and the participant has been granted permission to join the network. The new node must store the root certificate in `certificates/network − root − truststore.jks`, provide network map and doorman URLs, and store his key in his keystore
4. Finally, the node signs its node IP address and submit it to the Corda Business Network Map, for broadcast to other participant nodes

Implementation for independently managed network
The implementation of the process to be provided a valid certificate and the trust root certificate and to register to the network map is left to the appreciation of the network deployer. The new node must store the root certificate in `certificates/ networkroottruststore.jks`, and provide network map and doorman URLs.

Revoking Permission

Design
Revocation is handled using CRL that must be signed by the certificate's issuer.

Implementation
All CAs must provide a CRL distribution point using HTTP protocol. The CRL provider shall be mentioned in the `tlsCertCrlIssuer` field of node configurations. Before establishing a TLS connection, the receiver node checks the CRL of the sender node. If the check fails because the data is not available, the action aborts if `crlCheckSoftFail` is set to false and succeeds otherwise.

5.4.4 Restricted Corda Business Network Permissioning

When using the public Corda Business network, a sub-network of nodes that have decided to run the same Dapp can be created for business purpose.

Design
A restricted access can be defined within the existing Corda network using ACL. A list of nodes that are authorized to start the given Dapp flow must be stored on an accessible point.

Implementation
A list of authorized X.500 names must be stored in an http server. Each node of the restricted network is required to download and cache the list in a class annotated `@CordaServices`. The permission will be denied to a participant who tries to initiate a flow that requires authorization if his name does not match a X.500 name on the server.

Granting Permission

In order to grant permission to a new node, its X.500 name needs to be added to the list of authorized nodes.

Revoking Permission

If a restricted access is defined with the existing Corda network, the revocation of a node is done by deleting his name from the list on the http server described above.

5.4.5 Nodes Roles Granting

This subsection finalizes the answer provided to research sub-question 1 and explains the mechanism for nodes roles granting in Corda.

Design
Corda specifies the actions that a node is allowed to perform using **ABAC**: identity types are defined in certificates. The types can be any of the following:

- Doorman
- Network map
- Service identity
- Node certificate authority (from which the TLS and well-known identity certificates are issued)
- Transport layer security
- Well-known legal identity
- Confidential legal identity

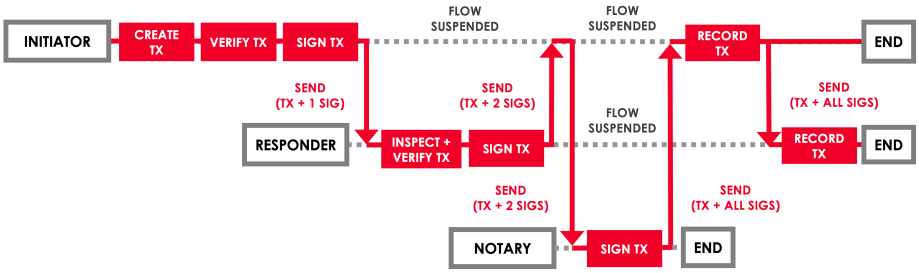

Figure 5.6: Corda flow framework. Extracted from *Corda Key Concepts*, URL: `https://docs.corda.net/key-concepts-flows.html`. Retrieved March 5, 2019

Implementation
Corda certificate custom X.509v3 extension specifies the role the certificate relates to using the OID 1.3.6.1.4.1.50530.1.1. The extension contains a single ASN.1 integer identifying the identity type. A simple check of the extension enables to verify which action the given entity is enabled to perform depending on their identity type.

5.4.6 Transaction Authorization

Transactions are treated as regular messages in Corda, and thus transaction sender authentication is handled similarly to message sender authentication as a given transaction is associated with a node. As transaction are shared directly between stakeholders, authorization to access a transaction must not be addressed. We are left with the question of how the authorization to send a transaction is granted to answer (RSQ2).

Design
In opposition to many permissioned blockchain platforms, **all transactions are private which means that they are only shared between their stakeholders, and authorization to invoke a transaction is provided by notaries.** This process is illustrated in Figure 5.6. For a transaction to be valid, all input states must first be repointed to have the same notary. This is done using a special notary-change transaction that produces an output state identical to the input state but appointed to a new notary.

Implementation
A transaction is sent by invoking the `SendTransactionFlow` method which digitally signs the transaction with the sender private's key. The receiver responds by invoking `ReceiveTransactionFlow` method and validates the transaction if the method `verifyRequiredSignatures`() returns true. To verify that the sender is authorized to perform the transaction, the notary verifies that he did not sign any other transactions consuming the proposed transaction's input states. The notary will then send back the transaction signed with its private key.

5.4.7 Remote API authentication and authorization

This subsection addresses research sub-question 3 and explores how remote users are authenticated and authorized to access a given Corda node.

Authentication

Design
Remote user authentication is supported **by a password authentication scheme** which is managed at node level. **An optional certificate-based client authentication can be set using TLS** (note that mutual authentication is not supported).

Implementation
When attempting to remotely connect to a node, the user is required to provide a username and a password which are compared to valid credentials. Those credentials can be fetched in the following ways:

- From a plain text configuration file
- From an external database with optional in-memory caching
- From hash-encrypted format conforming to the Modular Crypt Format provided by the Apache Shiro framework

If TLS client authentication is required, the remote user is required to provide a valid TLS certificate. It must be issued via the remote node specific RPC certificate for which the root of trust is distributed to remote clients. Those options are managed via the constructor parameter `SSLConfiguration`. Corda Experiment 1 (7.4.3) highlights the fact that by default RPC TLS communication is disabled in the Dapp official example version.

Authorization

Design
The full interface of the node is exposed to remote APIs calls. **Authorization to use Corda remote API uses capability lists**. The remote user possesses a capability list per remote node that he accesses. Thus, the methods that a remote user is allowed to invoke depends on the authorization that he has been granted.

Implementation
The capability list of each user is defined by a set of options nested in the security field of each `node.conf` file, in which each RPC user is listed along with his permission. By default, RPC users are not permissioned to perform any RPC operations.

Chapter 6

Vulnerabilities

In this chapter, we analyze the vulnerabilities of each platform with regard to the authentication and authorization schemes. First, we define the threat model on which our analysis is based. For each technology, we then present a summary of the findings, and decompose our reasoning.

6.1 Threat Model

We define a threat model with the help of related works presented in Chapter 3. We divide this model in four categories: authentication threats, authorization threats, security single points of failure and default parameters vulnerabilities.

6.1.1 Authentication Threats

Authentication attacks occur when an attacker masquerades a legitimate user or modifies the expected behavior of an authentication scheme. The definition of the following authentication threat model has been shaped by the reading of related works from Jesudoss A [9], Maria Nickolova and Eugene Nickolov [10] and Towhidi et. al. [8], and adapted to describe a framework for authentication threats applicable to blockchain platforms. We keep in mind when designing our threat model that permissioned blockchains rely on message sender authentication to ensure that only authorized node can participate in the network. To provide the following definitions, we help ourselves with the Dobromir Todorov's book *Mechanics of User Identification and Authentication* [11].

Password brute force attack

A password brute force attack consists in trying all possible combinations in order to find the password that satisfies the authentication scheme. Note that key brute force attacks are addressed in key compromise attacks category.

Dictionary attack

A dictionary attack is a kind of brute force attack in which the attacker uses a list of commonly used combinations or a dictionary to guess the combination that satisfies the authentication scheme.

Password sniffing attack

In the context of authentication, a sniffing attack consists in intercepting the authentication credentials by capturing the network traffic.

Key compromise attacks

A key compromise attack happens when an attacker discovers an authentication key. This can be caused by weak key schemes, an incorrect use of keys or an inappropriate key storage.

Replay attack

A replay attack consists in an attacker eavesdropping a communication channel in order to replay intercepted data later to impersonate the original sender.

Man-in-the-middle attack

In a Man-in-the-middle attack, an attacker relays and possibly modifies communication between two parties without them knowing that their session has been compromised.

Session hijacking attack

Session hijacking is an attack which consists in taking over an ongoing session and masquerading as an authorized user.

Source non-repudiation attack

A source non-repudiation attack deals with a sender denying committing a given action.

DDoS and DoS attacks

In the context of authentication, DDoS and DoS consist in sending several authentication attempt requests in order to saturate the receiver and disrupt its behavior. The difference between DDoS and DoS relies on the fact that DDoS attempts emanate from different sources whereas DoS uses a single source.

6.1.2 Authorization Threats

Our authorization threats model defines two potential attacks: user elevation of privileges and exploitation of vulnerabilities in the access granting and revoking scheme. The definitions are provided below.

Elevation of privileges

Also called privilege escalation, it consists in exploiting a design flaw or a bug in order to gain privileges that the attacker is not supposed to be granted.

Exploitation of access granting and revoking vulnerabilities

This attack relies on the exploitation of a design flaw or a bug in the access granting and revoking scheme, allowing unauthorized or revoked users to access some resources.

6.1.3 Security Single Points of Failure

We define a security single points of failure as being a part of the system that, if it fails or is successfully attacked, compromises the whole security of the network.

6.1.4 Default Parameters Vulnerabilities

In this thesis, our work relies on the hypothesis (H2) that users tend to adapt their system from existing sample scripts rather than building it from scratch. As a consequence, an insecure script is likely to induce deployments of vulnerable systems. For this reason, vulnerable default parameters are included in our threat model and the vulnerabilities of the default parameters of the official sample script that we have chosen for each platform are assessed.

6.2 Ethereum Vulnerabilities

6.2.1 Summary

This subsection presents a brief summary of Ethereum authentication and authorization schemes vulnerabilities.
The result of the analysis of the node and transaction authentication schemes vulnerabilities are depicted in Table 6.1: we find out that both schemes are vulnerable to key compromise attacks, and that node authenticated communication is also vulnerable to man-in-the-middle and session hijacking attacks due to the protocol broken handshake.
Further, our analysis highlights that **remote users are not required to be authenticated except when preforming wallet related method calls**. This absence of authentication system **exposes client endpoints to DDoS and DoS attacks and remote calls to man-in-the-middle attacks**. The mechanism of remote account owner authentication is passphrase-based and **the result of the analysis of its vulnerabilities are presented in Table 6.2: remote account owner authentication scheme is vulnerable to passphrase sniffing and to brute force and dictionary attacks when RPC or WS personal modules are enabled, remote calls are also vulnerable to man-in-the-middle attacks.** Ethereum experiment 2 and 3 (7.1.4, 7.1.5) depict thank to a honeypot the high popularity of Remote APIs attacks and demonstrate that the attacker's final main purpose is gaining financial benefit.
Ethereum mechanism for remote user authorization might be vulnerable to a kind of elevation of privilege attack called modules enabling if the client node enables the RPC or WS admin modules.
Finally, **default parameters do not show any weaknesses**.

Attack / Scheme	Node authentication	Transaction authentication
Brute force	N\A	N\A
Dictionary	N\A	N\A
Sniffing	N\A	N\A
Key compromise	**Yes**	**Yes**
Replay	No	No
MITM	**Yes**	N\A
Session hijacking	**Yes**	N\A
Source non-repudiation	No	No
DDoS/DoS	**Yes**	N\A

Table 6.1: Ethereum node and transaction authentication schemes vulnerabilities

Attack / Scheme	Remote account owner authentication
Passphrase brute force	**If WS or RPC personal module enabled**
Dictionary	**If WS or RPC personal module enabled**
Passphrase sniffing	**Yes**
Key compromise	N\A
Replay	N\A
MITM	**Yes**
Session hijacking	**Yes if unlocked different from 0**
Source non-repudiation	No
DDoS/DoS	N\A

Table 6.2: Ethereum remote account owner authentication scheme vulnerabilities

6.2.2 Node Authentication

In Ethereum, nodes communicate via the RLPx Transport Protocol which maintains authenticated channels established using node's asymmetric key pair and a share symmetric key as described in 5.1.3. No password is used, so dictionary attacks and password sniffing attacks are not relevant for this section. We study hereafter the remaining threats.

Key compromise - Vulnerable

The research survey conducted by Hartwig Mayer [18] highlights some vulnerabilities of the elliptical curve secp256k1 which is used for node key pair generation in Ethereum.
First, this curve suffers from **a mathematical vulnerability** which makes it vulnerable to a speedup of **Pollards rho attack** on small factors due to its existing automorphism.
As a reminder, Pollard's rho attack as described by Orhon and Taleb [26] is a method to solve the discrete logarithm problem using collisions by getting two differently generated equal points. Collision points are found by having a starting point that goes on a walk.

Further, this curve also presents implementation vulnerabilities: as no ladder implementation for scalar multiplication is available, this causes no constant time computations which can lead to leak information using **channeling attacks**. The previously stated existence of automorphisms makes it weak against **invalid curve attacks** which consist in an attacker uses a twist of the original curve. If the implementation does not check that the point is on the elliptical curve secp256k1, the attack may extract the private key with great success probabilities.

Replay attack - Non vulnerable

The RLPx Transport Protocol uses a nonce which prevents replay attacks.

Man-in-the-middle and hijacking attacks - Vulnerable

The RLPx handshake is considered to be broken. Indeed, an attacker can perform a known plain text attack and then conduct a Man-in-the-middle or hijacking attack. This attack would be performed in the following way:

First, the Counter mode (CTR) turns a block cipher into a stream cipher.
Given (k, IV) where k is a random encryption key and IV is an initialization vector:

For the i message m_i:

The message is encrypted: $c_i = E_k(IV||i) \oplus m_i$

The message is decrypted: $m_i = E_k(IV||i) \oplus c_i$

The knowledge of m_i enables to recover the key stream of this block:

$$E_k(IV||i) = m_i \oplus E_k(IV||i) \oplus m_i$$

The cipher of the other stream (other participant) can thus be forged:

$$c_i = E_k(IV||i) \oplus m_i$$

Source non-repudiation - Non vulnerable

In Ethereum, node authentication relies on ECDSA digital signatures which prevents source non-repudiation attacks.

DDoS and DoS attacks - Vulnerable

Ethereum nodes are vulnerable to DDoS and DoS attacks, also called Eclipse attacks in the context of blockchain as described in the paper *Low-Resource Eclipse Attacks on Ethereum* from Marcus et. al. [27]. Indeed, any node can establish a connection with another node, Ethereum nodes make thirteen outgoing connections by default and if those connections are monopolized by the attacker, the node is cut from the rest of the network.

6.2.3 Transaction Authentication

In Ethereum, authentication is used to verify that the transaction signer owns the private key associated with the public address from where the transaction is sent. It relies on the fact that transactions are signed using recoverable ECDSA signatures. Similarly to node authentication, no password is used, so brute force attacks, dictionary attacks and password sniffing attacks are not relevant for this section. Transactions being sent by nodes, the communication authentication in itself has been already studied and Man-in-the-middle, session hijacking and DDoS and DoS attacks are not applicable. We study hereafter the remaining threats.

Key compromise - Vulnerable

Similarly to node authentication, account keys are generated using the elliptical curve secp256k1 which has been proven to be weak against some attacks. One

shall refer to the authenticated channel key compromise description (6.2.2) to understand the vulnerabilities of this curve.

Replay attacks - Non vulnerable

Ethereum is not vulnerable to replay attacks. Indeed, transactions include a nonce.
After the DAO hack, Ethereum was split in two separate network ETH/ETC which allowed for replay attacks as transaction of one network could be resubmitted in the other network. However, a protection mechanism which includes the ID of the chain has been added to prevent replay attacks.

Source non-repudiation - Non vulnerable

Transactions are signed with a digital signature using ECDSA which prevents source non-repudiation attacks.

6.2.4 Remote User Authentication

Ethereum remote client authentication mechanism presents potential vulnerabilities which are exposed in a paper written by Wang et. al. [19]. It mostly relies on the fact that RPC and WS protocols are **stateless and plain text**. No user authentication is used excepted for wallet related interactions where the user has to authenticate himself by providing a passphrase, this is detailed in the next section. We consider below all remote actions that do not require user authentication. Password brute force attacks, dictionary attacks, sniffing attacks, replay attacks, key compromise attacks, session hijacking are thus not relevant.

Man-in-the-middle attack - Vulnerable

As no authentication is required for the remote user and for the node to which it sends requests, a Man-in-the-middle attack can easily be set up. For instance, a remote user could possibly send requests thinking that he is communicating with a given node ignoring that an attacker maliciously intercepts the traffic and modifies it.

DDoS and DoS - Vulnerable if RPC or WS enabled

RPC and WS being stateless, a node can be targeted by a DDoS or a DoS attack where the attacker floods the node with requests.

6.2.5 Remote Account Owner Authentication

A remote user authenticates himself as the account owner by providing a passphrase. This authentication is handled via RPC or WS which are plain text and stateless. For an account owner to remotely invoke account related methods, the personal module needs to be enabled. **No passphrase enforcement policy is set up.**

Passphrase brute force attack - Vulnerable if WS or RPC personal module enabled

When WS or RPC personal modules are enabled, the remote user can execute APIs methods that require a given account to be unlocked. For that purpose, the remote user is required to enter a passphrase. RPC and WS being stateless, this scheme is vulnerable to brute force attacks.

Dictionary attack - Vulnerable if WS or RPC personal module enabled

Similarly to brute force attacks, the scheme is vulnerable to dictionary attacks when WS or RPC personal module are enabled. This has been observed in the Ethereum Experiment 2 (7.1.4).

Passphrase sniffing attack - Vulnerable

RPC and WS being plain text, if a user unlocks his account using the remote APIs his passphrase transits in clear, an attacker can sniff the network traffic to capture it.

Man-in-the-middle attack - Vulnerable

Mutual authentication mechanism is not implemented: the node does not authenticate itself to the remote user. As a consequence, a Man-in-the-middle attack can be conducted, an attacker can for instance impersonate the node to which the remote user is trying to connect.

Session hijacking - Vulnerable if unlocked different from 0

Once unlocked, a given account stays unlocked for a given duration. An attacker can take advantage of this unlocking period to remotely invoke account related methods.

Source non-repudiation - Non vulnerable

The remote user authenticates himself as the owner of a given account by providing a passphrase which prevents from source non-repudiation attacks.

6.2.6 Elevation of Privileges

Remote APIs suffer from an elevation of privilege vulnerability. If remote users are granted the right to access only certain modules including the admin module, then a module-enabling attack can be conducted. It consists in a remote attacker sending a RPC or WS request to grant all remote users the authorization to use a given module.

6.2.7 Default Parameters Vulnerabilities

Ethereum Experiment 1 (7.1.3) assesses the RPC and WS default parameters of the official Ethereum client Geth. The results show that by default RPC and WS are disabled, as a consequence, the default configuration is neither

Attack / Scheme	Message sender authentication	Transaction sender authentication
Brute force	N\A	N\A
Dictionary	N\A	N\A
Sniffing	N\A	N\A
Key compromise	**Yes**	**Yes**
Replay	No	No
MITM	**If TLS disabled**	N\A
Session hijacking	**If TLS disabled**	N\A
Source non-repudiation	No	No
DDoS/ DoS	**If mutual TLS disabled**	N\A

Table 6.3: Quorum message sender and transaction sender authentication schemes vulnerabilities

vulnerable to the accounts authentication vulnerabilities described nor to the elevation of privilege attack.

6.3 Quorum Vulnerabilities

6.3.1 Summary

This section summarizes the analysis of Quorum vulnerabilities. Being the permissioned version of Ethereum, it presents many similar flaws.
Message sender authentication and transaction sender authentication schemes vulnerabilities are presented in Table 6.3: message sender and transaction sender authentication schemes use the same key generation scheme which is vulnerable to key compromise attacks. Further, if TLS is disabled, message sender authentication protocol is vulnerable to MITM, session hijacking, and mutual TLS is required to prevent DDoS and DoS attacks.
Further, block communication is handled differently from regular communication in Quorum and relies on HTTP or HTTPS depending on TLS settings. **If TLS is disabled, node authentication for block communication does not exist and this exposes the network to the following attacks: replay attacks, MITM, source non-repudiation, DDoS and DoS.**
Remote user authentication only exists for account related actions and is identical to Ethereum and one shall report to Table 6.2. Quorum suffers from the same vulnerability as Ethereum for elevation of privileges with regard to the remote APIs: **an attacker can conduct a module enabling attack if the admin module is enabled to gain privileges. Further, it suffers from an additional vulnerability which allows unauthorized users to access private transactions if an authorized node enables RPC to public internet with the eth module.**
Two vulnerabilities were detected in the privilege granting model: TLS TOFU and whitelist do not allow for certificate revocation which can cause a compromised node to be used for DDoS or DoS attack, and **if the network permissioning file is not identical on each node the behavior of the network is uncontrolled.**
If the same keys are used for node identity and TLS, the compromise of a private key becomes a single point of failure as this would grant

access to an attacker to all the network public transactions. **If the system does not enforce permissioning with a list when using CA mode, the root CA also becomes a security single point of failure.**
Finally, Quorum official sample script default parameters are the weakest possible: TLS is disabled and RPC is enabled on all nodes and exposed to public internet, and accounts are forced to be unlocked permanently. An additional vulnerability relies on the fact that **TOFU mode prevents a node from changing a compromised key pair** as the node would not be able to rejoin the network after a change of key pair.

6.3.2 Node Authentication for General Communication

Quorum uses the existing Ethereum p2p transport layer to communicate transactions between nodes, but communicate blocks only through HTTP. Optionally, TLS can be enabled with either mutual, client or server authentication. No password is used, thus brute force, dictionary and sniffing attacks are not applicable. We explain below the found vulnerabilities.

Key compromise - Vulnerable

Quorum uses the same node keys as Ethereum, and thus suffers from the same vulnerability. For more explanations, one shall refer to subsubsection 6.2.2. However, Tessera uses RSA with 2048 bits to generate TLS keys, this scheme has not been broken yet and thus when TLS is enabled the system remains secure if the same key is not reused for node identity and TLS.

Replay attack - Not vulnerable

The RLPx Transport Protocol uses a nonce which prevents replay attacks.

Man-in-the-middle and session hijacking attacks - Vulnerable if TLS disabled

Quorum uses the same P2P transport layer as Ethereum adapted to reject a non-permissioned node. Thus, if TLS is disabled, an attacker can conduct a Man-in-the-middle or session hijacking attack as described in Ethereum node authentication vulnerabilities (6.2.2). When TLS is enabled, it mitigates MITM attacks using certificate-based authentication.

Source non-repudiation - Not vulnerable

Similarly to Ethereum, the communication protocol uses ECDSA which prevents source non-repudiation attacks.

DDoS and DoS attacks - Vulnerable if TLS mutual authentication disabled

If TLS mutual authentication is disabled, then Quorum nodes are vulnerable to DDoS and DoS attacks similarly to Ethereum (6.2.2).

6.3.3 Node Authentication for Block Communication

In Quorum, blocks are sent via HTTP, and optionally with HTTPS if TLS is enabled. No password is required, thus password brute force, dictionary and sniffing attacks are not applicable. Further, HTTP is stateless thus session hijacking is ignored.

Replay attack - If TLS disabled

Even though block replay attack is interest-free for attackers as block contains transactions which are protected against replay attacks, this attack remains possible if TLS is disabled.

Man-in-the-middle attack - If TLS disabled

If mutual TLS is disabled, an attacker can impersonate the non-authenticated node during the block exchange.

Source non-repudiation attack - If mutual TLS disabled

If mutual TLS is disabled, the block sender node can deny having sent the block and the receiver node can deny its reception.

DDoS and DoS attacks - If client TLS authentication disabled

If client TLS is disabled, an attacker can flood a given node HTTP endpoint.

6.3.4 Transaction Authentication

Quorum transaction authentication is similar to Ethereum, as explained in Ethereum section transaction authentication mechanism does not suffer from any vulnerabilities excepted key compromised. If TLS is disabled, Quorum suffers from the same vulnerability, however, enabling TLS secure the transaction sender authentication with an additional key scheme (RSA by default).

6.3.5 Remote User Authentication

Quorum remote client access authentication is identical to Ethereum remote client access, and thus suffers from the same vulnerabilities: Man-in-the-middle, DDoS and DoS. One shall refer to subsection 6.2.4.

6.3.6 Elevation of Privilege

Quorum remote API authorization mechanism suffers from the same vulnerability as Ethereum remote APIs, thus it is vulnerable to module-enabling attacks as explained in subsection 6.2.6. On top of that, if the RPC or WS eth module is enabled and not restricted to localhost, an attacker can bypass the transaction authorization scheme and connect to a node that has access to a private transaction to retrieve it. The access to private transactions is handled via private key authentication, and the key scheme use to generate keys in Quorum module is vulnerable to the speed up of Pollards rho attack and channel attacks.

6.3.7 Exploitation of Access Granting and Revoking Vulnerabilities

The privilege granting model suffers from several vulnerabilities:

- TLS TOFU and Whitelist modes do support node revocation. Revocation can be done directly in the permissioning file of each node, but that would not prevent revoked nodes from conducting a DoS or DDoS attack.
- In the case where the network nodes possess different versions of the permission file, an inconsistent behavior is triggered as shown in Quorum Experiment 1 7.2.3.

6.3.8 Security Single Points of Failure

We evaluate the system single points of failure: if the same keys are used for node identity and TLS, the compromise of a private key can lead to an attacker accessing all public transactions in the network. Further, If the system does not enforce permissioning but only implements TLS CA, then root CA is a security single point of failure.

6.3.9 Default Parameters Vulnerabilities

Quorum official sample script default parameters present numerous vulnerabilities: by default, TLS is disabled which exposes the system to Man-in-the-middle attacks, key compromise attacks, DDoS and DoS. Further, the remote APIs parameters are the weakest possible as shown in Experiment 9 (7.2.9). By default, RPC is enabled on all nodes with no restriction on the authorized modules and exposed to the public internet, additionally, accounts are forced to remain unlocked.

6.3.10 Additional Vulnerabilities

Quorum Experiment 6 (7.2.6) highlights a vulnerability of the TOFU mode: if a node key gets compromise, TOFU mode prevent the node from changing its key pair. Indeed, the node would be refused access to the network as it would connect from a known host with a different public key associated.

Attack / Scheme	Message sender authentication	Message sender authentication using Fabric CA
Brute force	N\A	**If TLS client authentication disabled**
Dictionary	N\A	**If TLS client authentication disabled**
Sniffing	N\A	**If TLS disabled**
Key compromise	No	N\A
Replay	No	**If TLS disabled**
MITM	No	**If TLS disabled**
Session hijacking	No	**If TLS disabled**
Source non-repudiation	No	No
DDoS/DoS	**If mutual TLS disabled**	**Yes**

Table 6.4: Hyperledger Fabric authentication schemes vulnerabilities

6.4 Hyperledger Fabric Vulnerabilities

6.4.1 Summary

This subsection presents a summary of the identified Hyperledger Fabric vulnerabilities.
Table 6.4 presents a summary of the node, transactions and Fabric CA authentication vulnerabilities: if TLS is disabled, the message sender authentication scheme is vulnerable to MITM, and mutual TLS is required to prevent DDoS and DoS attacks.
The absence of channel authentication presents a threat to the system security which has been demonstrated in the experiment section: the user has no guarantee that the channel that he is using has not been modified by an attack of the majority of the channel organizations as the channel does not authenticate itself to the user.
An externally conducted security Assessment management report highlights that the **implementation lacks of chaincode sandboxing and demonstrates how malicious chaincode can perform elevation of privilege attacks on previous version 1.1 but nothing ensures that this has been fixed.** Further, privilege revocation scheme presents the following vulnerabilities: **TLS certificate revocation is not supported and expiration of identity certificate is ignored by the system.** Additionally, **two possible security single points of failure has been identified depending on the chosen configuration**: the use of a single orderer and the use of a single root CA for both TLS and MSPs.
Finally, the official sample script default parameters present a vulnerable configuration: client authentication is not enabled thus exposing the system to DDoS and DoS and TLS is disabled for Fabric CA.

6.4.2 Node and Transactions Authentication

In Hyperledger Fabric, a transaction is handled as a regular message. As a consequence, the schemes are identical and we provide an unique analysis. A node is authenticated by two pairs of keys: one for the identity and one for the TLS protocol. No password and so this scheme is **not vulnerable to dictionary attacks and sniffing attacks**. We study here after the remaining threats. We remind that by default TLS server authentication is enabled but client authentication is disabled.

Keys compromise - Not vulnerable

Hyperledger Fabric supports ECDSA and the user is able to import his own keys. Fabric CA module is offered to generate keys using either prime256v1 or secp521r1 curves. None of those key schemes have shown evidence of vulnerabilities yet.

Transaction replay attack - Non vulnerable

Transaction replay attacks are mitigated by a double mechanism: first, a nonce is used to avoid an attacker from replaying a transaction, second, a node is prevented from re-submitting a transaction that has been endorsed as the endorsement contains read/write set with a unique state identifier for all variables implied. Nodes do not communicate with each other except for transactions, and the other message flow existing in the channel is admin updates. Admin updates cannot be replayed as they are based on the current configuration fetching, once committed the configuration is changed and a new update requires to re-fetch the configuration.

Man-in-the-middle and session hijacking attacks - If TLS disabled

If TLS is disabled, Hyperledger cannot prevent an attacker to replay the traffic between two nodes. However, he would not be able to modify it due to the use of ECDSA.

Source non-repudiation - Non vulnerable

All communication in Hyperledger Fabric use ECDSA which prevents source non-repudiation attacks.

DDoS and DoS attacks - Vulnerable if mutual TLS disabled or attacker is part of the network

A DDoS or DoS attack requires one of the user not to be authenticated via TLS. In the case when TLS is disabled or only server authentication is required, this attack can be launched. Indeed, an attacker will try to connect, establish the transaction and then be denied access as it does not own a valid identity key. Further, if the attacker is part of the network, he can possibly launch this attack but is likely to be revoked from the network by other peers.

6.4.3 Fabric CA module

Hyperledger Fabric offers a module which handles certificate issuing called Fabric CA module described in 5.3. All communication with the CA module is handled via the REST API. The following modes for TLS are available: disabled, client authentication, server authentication and mutual authentication.

Password brute force attack - Vulnerable if TLS client authentication disabled

In order to register a new identity, the identity performing the registration must provide its username and password. If TLS client authentication is disabled,

then Fabric CA module is vulnerable to brute force attacks.

Dictionary attack - Vulnerable if TLS client authentication disabled

Similarly to password brute force, Fabric CA module is vulnerable to dictionary attacks if the client is not required to authenticate itself via a TLS certificate.

Sniffing attack - Vulnerable if TLS disabled

An attacker can sniff one identity's username and password if TLS is disabled by capturing the network traffic.

TLS Key compromise - N\A

No information is available about the required format for TLS keys, and their possible compromise depends on the user's choice.

Replay attacks - Vulnerable if TLS disabled

Fabric CA is vulnerable to replay attack if TLS is disabled. Indeed, the communication is handled via the REST API which is plain text, and thus can be intercepted and replayed.

Man-in-the-middle attack and session hijacking - Vulnerable if TLS is disabled

In the case where TLS is disabled, an attacker can perform a MITM or session hijacking attack as the traffic relies on the REST API which are plain text and stateless.

DDoS and DoS attacks - Vulnerable if client TLS disabled or attacker part of the network

A DDoS or DoS attack requires one of the users not to be authenticated via TLS. In the case when TLS is disabled or only server authentication is required, this attack can be launched.

6.4.4 Channel Authentication

The channel does not authenticate itself to the user. In other words, the user has no guarantee that the network or channel that he is using has not been modified. A possible attack has been described in Hyperledger Fabric Experiment 2 (7.3.4): as a majority of organization is able to eject a chosen organization from a network or channel, the majority could possibly eject an organization, modify the network or channel MSP to add root or intermediate CAs, and re-add the ejected organization which would adopt the new MSP rules without noticing the change.

6.4.5 Elevation of Privilege

Regarding transactions and node authorization, we did not find any flaws in the authorization and chaincode policy schemes. However, if a single peer orderer is deployed and gets compromised, the chaincode policy can be bypassed (for this reason a single orderer is considered a security single point of failure). Further, the Security Assessment management report conducted by Nettitude [20] on version 1.1 depicts a lack of chaincode sandboxing causing possible elevation of privilege attack: chaincodes are executed in containers, however those containers appears not be enough sandboxed as Nettitude was able to write a malicious chaincode capable of performing a nmap scan. As no similar review has been done for the new version, a user shall still be careful about this attack.

6.4.6 Exploitation of Access Granting and Revoking Vulnerabilities

The following vulnerabilities have been found in the network access granting and revoking model:

- There is currently **no support to revoke TLS certificates** meaning that communication between nodes can be established even if the MSP certificate of that given user has been revoked.
- Enrollment certificates (MSP) **do not consider expiration**, meaning that at the MSP level an expired certificate is still considered as a valid identity.

6.4.7 Security Single Points of Failure

The following potential single points of failure have been identified depending on the chosen configuration:

- Single orderer: if the system uses a single orderer, in case of compromise, the attacker is able to censure transactions or endorse malicious transactions.
- Single root CA: if the same CA is used for TLS and MSP certificates, in this case, if the root CA gets compromised, then the attacker could generate both a TLS and MSP certificate and act like a new node that would satisfy all security requirements in the channel.

6.4.8 Default Parameters Vulnerabilities

Hyperledger Fabric Experiment 1 (7.3.3) describes the default node and orderer TLS parameters: TLS is enabled but client authentication is not required in the default parameters. This default configuration makes the network vulnerable to DDoS and DoS attacks. Further, Experiment 2 (7.3.4) shows that by default TLS is disabled on both Fabric CA client and server exposing it to all attacks of our threat model (excepted key compromised as no key is used in this case).

Attack / Scheme	Message sender authentication	Remote user authentication (RPC)
Brute force	N\A	**Yes**
Dictionary	N\A	**Yes**
Sniffing	N\A	**If TLS disabled**
Key compromise	**Yes**	**Yes**
Replay	**If TLS disabled (No for transactions)**	**If TLS disabled**
MITM	**If TLS disabled**	**If TLS disabled**
Session hijacking	No	**Not vulnerable if TLS enabled, N\A otherwise**
Source non-repudiation	No	No
DDoS/DoS	**If mutual TLS disabled**	**Yes**

Table 6.5: Corda message sender and remote user authentication schemes vulnerabilities

6.5 Corda Vulnerabilities

6.5.1 Summary

This section presents a summary of Corda authentication and authorization scheme vulnerabilities. The explanations for the reasoning are described in the following subsections.
Table 6.5 presents a summary of the message sender and remote user authentication schemes vulnerabilities to the attacks defined in the subsection threat model (6.1.1): **message sender authentication scheme is exposed to replay attacks, DoS and DDoS attacks if TLS is disabled. Remote user authentication scheme is vulnerable to brute force, dictionary, key compromise, DDoS and DoS attacks in any case, and to sniffing, replay, MITM and session hijacking attacks if TLS is disabled.**
Further, the study of potential elevation of privileges attack shows that **nodes are vulnerable to malicious Dapps** that would allow a contract to instantiate classes in the JVM that it is not authorized to access. Thus, node admins are required to trust the Dapps that they install. **Three security single points of failure have been identified, the last one depending on the chosen configuration**: the single root CA, the single doorman and deployment with a single notary.
Further, the assessment of default parameters shows a secure TLS node configuration as mutual TLS authentication is enabled by default but `crlCheckSoftFail` parameter is set to true, **allowing a bypass of the CRL check if the list is not available.** Further, the RPC parameters are insecure as TLS is disabled by default for communication between a remote user and a node, exposing the authentication scheme to brute force, dictionary, sniffing, replay, MITM and session hijack, DDoS and DoS attacks.
Finally, an additional vulnerability which prevents users to change compromised keys is described.

6.5.2 Node and Transactions Authentication

In Corda, similarly to Hyperledger Fabric, a transaction is handled as a regular message. As a consequence, the schemes are identical and we provide a unique analysis. A node is authenticated by two pairs of keys: one for the identity and one for the TLS protocol. No password being used, this scheme is **not vulnerable to password brute force, dictionary attacks and sniffing**

attacks. We study here after the remaining threats. We remind that by default TLS is enabled for node communication.

Key compromise attacks - Vulnerable

By default when using Corda utility to generate key pairs, for root network CA keys, doorman CA keys, node CA keys, TLS keys and network map (CN) keys, NIST P-256 curve (secp256r1) is used. Further, Corda generates by default node identity and confidential identity key pairs with Edwards-curve (ed25519), which has been proven to be weak. Indeed, Yolan Romailler and Sylvain Pelissier from Kudelski Security demonstrated that this EdDSA signature scheme is vulnerable against fault attacks. As a reminder, a fault attack aims at disturbing the normal behavior of a device making it output erroneous results or bypass certain operations. Indeed, this attack targets the hardware which processes the digital signature. **As a consequence, an attacker could potentially attack the sender's hardware, then brute force the error location and value, recover half of the secret key and can thus brute force the missing part**. No attack of this kind has been advertised on signatures using NIST P-256 curve.

Replay attack - Vulnerable if TLS is disabled for communication

Transactions in Corda implement a security against replay attack as the notary verify that he did not sign any other transactions consuming the transaction's input states. Regarding regular communications, replay attack is mitigated by virtue of the mutual authentication TLS protocol and thus communication is vulnerable to replay attack only if TLS is voluntarily disabled.

Man-in-the-middle attack and hijacking attacks - Vulnerable if TLS is disabled

When TLS is voluntarily disabled, the authentication scheme is vulnerable to MITM and session hijacking attacks as an attacker can relay the traffic. However, messages and transactions are still signed with identity keys which also prevent malicious modification.

Source non-repudiation attack - Not vulnerable

All communications and transactions in the network use digital signature which prevents source non-repudiation attacks.

DDoS and DoS attacks - Vulnerable if mutual TLS is disabled or attacker is part of the network

When mutual TLS is disabled or if the attacker is part of the network, there is no specific mitigation against DoS and DDoS attacks.

6.5.3 Remote User Authentication

Corda allows remote access to nodes via the RPC API. TLS is disabled by default on the Corda Dapp official example as explained in Corda Experiment

1 (7.4.3). We remind that password based authentication is used for Corda RPC API authentication, with optional TLS node authentication (remote user authentication using TLS is not supported).

Brute force attack - Vulnerable

RPC is a stateless protocol, which allows an attacker to conduct a brute force attack to guess the remote user's username and password. If TLS is enabled, only the node is required to authenticate itself and thus an attacker can conduct a brute force attack.

Dictionary attack - Vulnerable

Similarly to brute force attacks, dictionary attacks can be conducted under the same conditions.

Sniffing attacks - Vulnerable if TLS disabled

RPC is plain text, as a consequence, if TLS is disabled the traffic is not encrypted and can be sniffed. This allows for the attacker to retrieve username and passwords.

Key compromise attacks - Vulnerable

All TLS keys in the network use the same format. This attack has already been addressed and one shall refer to previously described node and transaction authentication key compromise attack (6.5.2).

Replay attacks - Vulnerable if TLS disabled

TLS prevent replay attacks, however, when disabled, no mechanisms in the remote API is enforced to prevent replay attacks.

Man-in-the-middle attack - Vulnerable if TLS disabled

If TLS is disabled, an attacker can intercept the traffic and modify the node's answer. When TLS is enabled, the remote connection is protected against MITM attacks.

Session hijacking attack - Not vulnerable is TLS enabled, N\A otherwise

Remote user is authenticated by a username and password which are required for each request (no default unlocking duration time), and thus cannot be hijacked. If TLS is disabled, no session is established and this attack in thus not relevant.

Source non-repudiation attack - Not vulnerable

RPC clients must authenticate to the node using credentials and thus cannot repudiate their packets. Further, all interactions with an RPC user are also logged by the node.

DDoS and DoS attacks - Vulnerable

RPC being a stateless protocol, the system will not limit the number of authentication attempts. RPC TLS does not use mutual authentication and thus allow for anyone to establish a connection with a given network node.

6.5.4 Elevation of Privilege

Regarding transactions and nodes, we did not find any flaws in the authorization scheme. Similarly, remote users are restricted by a capability list. However, a node itself is vulnerable to an elevation of privileges attack conducted by a Dapps, as Corda does not implement specific security controls to prevent privileges escalation. This could allow a malicious contract to instantiate classes in the JVM that it is not authorized to access. Corda requires Dapps to be trusted by the node administrator.

6.5.5 Exploitation of Access Granting and Revoking Vulnerabilities

For the Corda Business network, an obvious threat relies on the management of the privileges granting. The decision of granting or removing a participant depends on a third party, here R3.
For both networks, if the doorman gets compromised, it threatens the security of the entire network. Indeed, if an attacker takes over the doorman server, he is able to grant access to any participant. Further, the single root CA also represents a danger for the network, similarly, if it gets compromised, permission can be granted to anyone. Finally, the uniqueness of the network map is a vulnerability as its compromise might cause some participants to be maliciously revoked from the network.

6.5.6 Security Single Points of Failure

Several single points of failure have been identified, depending on the chosen deployment design:

- Single root CA: as the network possesses a single Root CA, if it gets compromised then the entire network is compromised.
- Single doorman: similarly, if the doorman gets compromised, the attacker can revoke any node CAs in the network via a CRL publication.
- Single notary configuration: If a single notary structure is deployed, the compromise of this notary will cause the entire authorization scheme to be compromised.
- Single network map: the uniqueness of the network map represents a single points of failure. A malicious attacker that takes over the network map is able to censor some network participants.

6.5.7 Default Parameters Vulnerabilities

Corda Experiment 1 (7.4.3) which consisted in determining the default parameters of the official Corda Dapp example presented the following results: by default mutual authentication is required between nodes. However, for remote users using RPC is disabled by default which induces several vulnerabilities shown in the summary vulnerability authentication Table 6.5. Further, `crlCheckSoftFail` default value is set to true, meaning that if CRL list is not available the communication is established even if the certificate has not been proven valid.

6.5.8 Addition Security Vulnerabilities

Issue CORDA-1678[1] prevents user from changing a compromised key pair, as NetworkMapClient throws a NullPointerException when trying to publish a NodeInfo corresponding to a X509 name that has been registered before with a different public key.

[1] https://r3-cev.atlassian.net/browse/CORDA-1678

Chapter 7

Experiments and Results

In this chapter, we conduct experiments which aim to demonstrate potential vulnerabilities, to determine the systems expected behavior and to evaluate demos default parameters with regard to the authentication and authorization schemes. For each platform, we process in the following way: first, we provide a reminder of the experiment to conduct, second we describe the test environment setup, finally we summarize each conducted experiment and present the results.

7.1 Ethereum

7.1.1 List of Conducted Experiments

We aim to conduct the following experiments:

1. Evaluate the default RPC and WS parameters when deploying a Geth node using the official tutorials.
2. Evaluate the frequency and strategy of RPC attacks by constructing a honeypot. The unlock duration being 300 seconds, we intend to measure the likelihood of an attack occurring within this lapse of time.
3. Calculate the frequency of successful attacks and the overall profitability with regard to piracy attacks.

7.1.2 Test environment Setup

We choose to deploy Ethereum Geth client as argued in subsection 5.1.5. For that purpose, we followed the official tutorial[1] for Mac. This deployment requires the installation of Homebrew, and of both Go and GMP external libraries. The latest Geth version (v1.8.23) is then downloaded from the official Ethereum go-Ethereum page[2].

[1] https://ethereum.gitbooks.io/frontier-guide/content/installing_mac.html
[2] https://github.com/ethereum/go-ethereum

7.1.3 Experiment 1

The first experiment aims to assess Geth client RPC and WS default parameters in order to evaluate its security against the previously described attacks.

Default parameters

We find that **by default RPC and WS services are disabled**.

Default configuration for enabled RPC services

Further we activate the RPC services with the command `−rpc`. The default configuration is the following:

- `rpcaddr`: localhost
- `rpcport`: 8545
- `modules`: eth, net, web3
- `rpccorsdomain`: browser enforced

The default parameters for WS services are the following:

- `wsaddr`: localhost
- `wsport`: 8546
- `wsapi`: eth, net, web3

Conclusion

The experiment aims to evaluate the default Geth parameters with regard to the RPC and WS services. We remind that remote access is vulnerable to attackers when it is accessible from either a local network or public internet. We find that by default, the services are disabled, and that they are only accessible by localhost when enabled. The most sensitive modules are admin, miner and personal, which are all disabled by default when activating the RPC and WS services with default configuration. **We conclude that by default the client remote APIs are not vulnerable and that client remote APIs exposure emanates from an explicit will from the user**.

7.1.4 Experiment 2

The goal of this experiment is to analyze the frequency and the strategy of RPC API attacks by constructing a honeypot.

Honey pot set up

An experiment is conducted to evaluate the frequency and strategy of RPC API attacks on an Ethereum node. In order to count the number of attempts in a given time, a node is deployed with all RPC modules enabled with the generic address `0.0.0.0` and `rpccorsdomain` open to all ("*"). We expose our node to public internet by defining a static IP address and configuring DHCP.

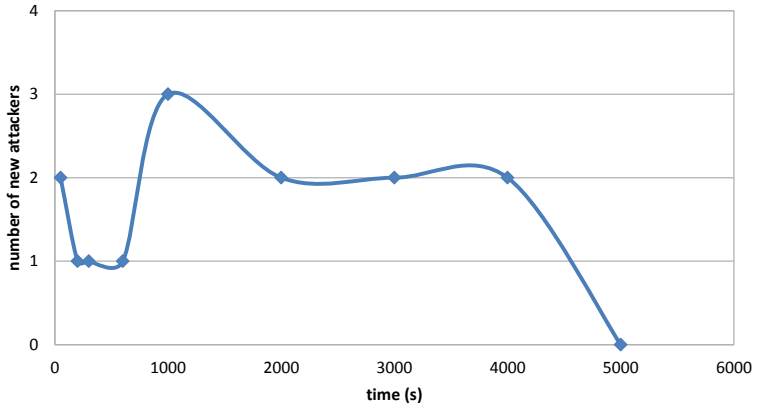

Figure 7.1: Number of new attackers arrivals in function the elapsed time (in seconds)

We host an Ethereum account protected by a passphrase on this node. The traffic is captured for one hour and a half with the packet sniffing software Wireshark (version 2.6.7), and filter with the parameter `http.content_type` $==$ `application/json`.

Analysis of the frequency

A frequency analysis is conducted based on the Wireshark capture. Within 5315 seconds (approximately 1h30), a total of 10849 JSON-RPC requests have been received. We can see that attackers are progressively discovering this node: Figure 7.1 shows the number of new attackers arrivals in function the elapsed time. After 4000 seconds, no new attacker is detected. In total, 13 different IPs got caught trying to steal funds from this honeypot.

As stated in subsection 5.1.5, by default, unencrypted keys are held in memory for 300 seconds once an account has been unlocked via remote client access. We aim to evaluate the frequency of attackers account related queries to measure the risk of an attacker accessing an unlocked account. A minority of attackers (30,8%) simply query the account balance, and forget the node address if the balance is null. However, 69,2% of the attackers continuously send packets at given intervals to detect if the account has been unlocked.
Figure 7.2 illustrates the time elapsed between the reception of two malicious queries aiming to steal funds. We see that the largest interval between two new attacks packet is 20 seconds, which is smaller than the default unlocking period of 300 seconds.

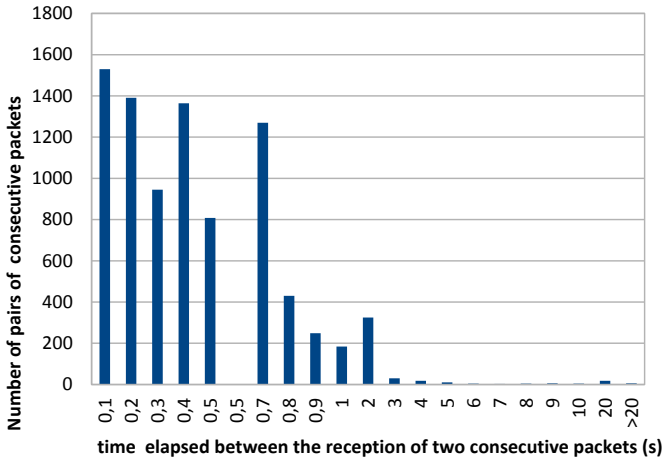

Figure 7.2: Time elapsed between the reception of two attackers account related packets

Analysis of the strategy

We further will to examine the attackers behavior. We use Wireshark filter to isolate the packets that emanates from the two most active attackers. We plot the time elapsed between two packets from the same sender as depicted in Figure 7.3, and observe two different patterns that we associate respectively to attacker 1 and attacker 2. When analyzing the graph, we notice that attacker 1 sends packets almost continuously whereas attacker 2 reproduces the same routine every 11 seconds. In both cases, the time elapsed between their attempts to access to the node account is smaller than the default unlocking period. The methodology is the following: the attacker first queries the address of the accounts hosted in the node, and then sends a request to transfer funds from this address to an external address. The packets captured are shown in Figure 7.4 and 7.5.

Our account is protected by a passphrase, thus the JSON-RPC answer to attackers packets stipulates that the account is locked. The Wireshark capture present evidence of **dictionary attacks**. Indeed, two attackers attempted to guess the password by consecutively trying to unlock the account with a guessed password. For both attackers, the first password tried is 123456.

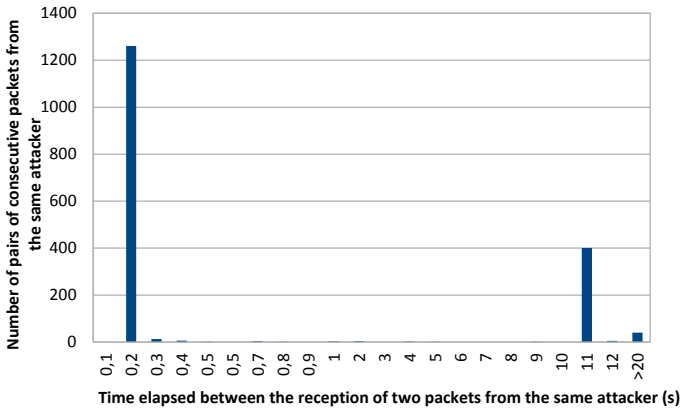

Figure 7.3: Time elapsed between the reception of two packets from the same attacker

```
    Timestamp echo reply: 981050804
▶ [SEQ/ACK analysis]
▶ [Timestamps]
  TCP payload (50 bytes)
  TCP segment data (50 bytes)
0000  88 1f a1 17 27 2e d4 60  e3 c5 46 78 08 00 45 00   ····'.·` ··Fx··E·
0010  00 66 52 60 40 00 32 06  e1 59 52 c4 00 5b c0 a8   ·fR`@·2· ·YR··[··
0020  01 11 e8 9e 21 61 69 c3  70 c9 50 43 80 e4 80 18   ····!ai· p·PC····
0030  05 a4 33 e8 00 00 01 01  08 0a 74 c1 c1 af 3a 79   ··3····· ··t···:y
0040  a5 b4 5b 7b 22 69 64 22  3a 30 2c 22 6a 73 6f 6e   ··[{"id" :0,"json
0050  72 70 63 22 3a 22 32 2e  30 22 2c 22 6d 65 74 68   rpc":"2. 0","meth
0060  6f 64 22 3a 22 65 74 68  5f 61 63 63 6f 75 6e 74   od":"eth _account
0070  73 22 7d 5d                                       s"}]
```

Figure 7.4: Account addresses query packet captured with Wireshark

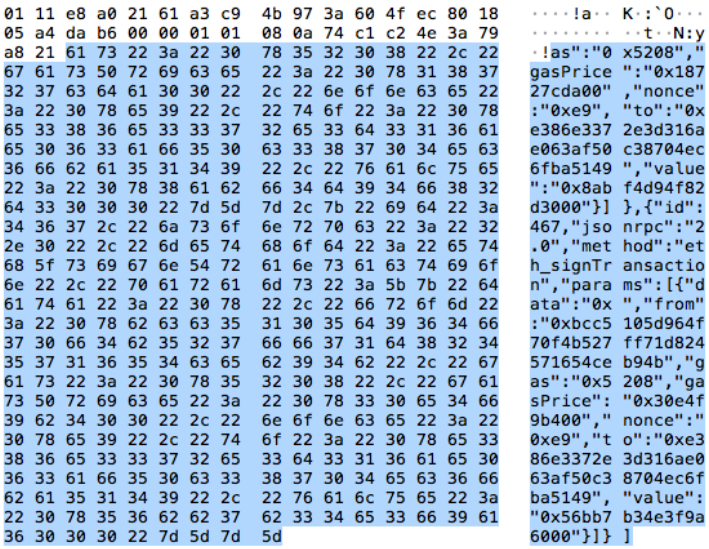

Figure 7.5: Captured transfer funds attack with Wireshark

Conclusion

The purpose of this experiment is to examine the RPC API attacks frequency and patterns. First, all collected packets aimed at attacking the account hosted on the node thus we can deduce that **the purpose of most attackers is gaining financial benefit**. Our analysis focuses on evaluating the elapsed time between two attempts to access the node account, in all cases, the time elapsed between the reception of two attack attempts was smaller than the default unlocking period. As a consequence, **if the account had been unlocked during this experiment, funds would have been stolen with a probability of 100%**. The analysis of the attack strategies reveals that the duration set between two attempts is always smaller than the default unlocking period. Finally, the experiment shows evidence of **dictionary attacks**.

7.1.5 Experiment 3

This experiment takes place in the continuity of Experiment 2. We intent to estimate the frequency of successful attacks, and the average overall profitability.

Methodology

In order to conduct this experiment, we extract from Experiment 2 an attacker account address. For that purpose, we analyze a packet which attempts to transfer funds from the most active attacker, and collect the `to` field of the `eth_signtransaction` method in the JSON-RPC request. We then extract the account's transactions report from the website `https://etherscan.io` and exploit it in with Excel.

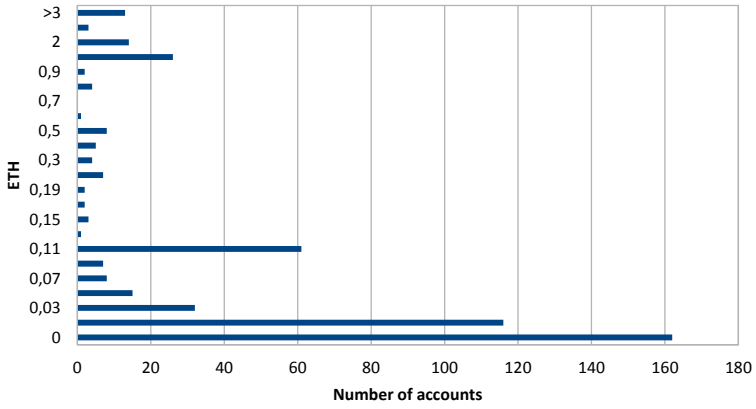

Figure 7.6: Stolen ETH amount from attacked accounts

Results

Within the last 545 days, this attacker received 871 transactions from 497 accounts for a total of 594.74 ETH. Figure 7.6 plots the distribution of stolen amount from accounts, highlighting that often small amounts are stolen. A hypothesis is that those small amounts come from mining rewards, however, we note that none of the attackers tried to change mining reward destination account on the honeypot node during Experiment 2.

Conclusion

A total of 597.74 Ether has been collected from the most active attacker of Experiment 2 within a period a 545 days. This represents a **consequent gain** which explains the large number of attackers and justify the setup of automatized attack attempts. This amount comes from 497 different accounts, which pushes the belief that a **large number of nodes present an insecure RPC services configuration**. As mostly small amounts are being collected by attackers, a hypothesis would be that the **funds are stolen from node mining rewards**.

7.2 Quorum

7.2.1 List of Conducted Experiments

We aim to conduct the following experiments:

- (1) Setting up a permissioned network in which nodes have different permission files
- (2) Dynamically add a node to every permission files
- (3) Dynamically delete a node from all permission files
- For different TLS configurations, we need to evaluate the following behaviors:
  - (4) Dynamically modify whitelist to add a node
  - (5) Dynamically delete a node from the whitelist
  - (6) Refresh a node certificate to test the acceptance of the network after the operation when using TOFU & CA
  - (7) Dynamically change root CA
  - (8) Dynamically revoke a node using TOFU mode
- (9) Evaluate the default WS and RPC parameters and if possible demonstrate possible attacks
- (10) Build a honeypot with default RPC parameters to evaluate the frequency of attacks

7.2.2 Test Environment Setup

We choose to follow the official GitHub Getting Started guidelines[3] to locally set up Quorum due to limited computational resources required by VirtualBox and Vagrant. The experiments are based on 7nodes setup example[4] which is advertised on Quorum official GitHub page and appears to be the most popular among the user community, this choice relies on the Hypothesis 2 stated in section 4.5. This sample script starts up a fully functioning Quorum (version 2.2.1) environment consisting of 7 independent nodes.

7.2.3 Experiment 1

The behavior of a system in which nodes have different permission file is assessed through Experiment 1.

Methodology

To conduct the experiment, we provide each node with a different permission file at network initialization and then start the network.

Results and conclusion

The initialization of the network occurs without error. However, all attempts to send transaction fail with the same error always being raised. As a consequence, when deploying a Quorum network, the initialization is not sufficient to ensure the proper functioning of the network and the existence of different permission files in the network causes the network to be non-functional.

[3] https://github.com/jpmorganchase/quorum-examples
[4] https://github.com/jpmorganchase/quorum-examples/tree/master/examples/7nodes

7.2.4 Experiments 2 and 3

This experiment is conducted in two parts to evaluate the system dynamic node modification capabilities via the permission file.

Methodology

First, we use the 7node example script with disabled TLS, and try to add the same node on every node permission file once the network is up (Experiment 2). We then re-initialize the network parameters and similarly delete the same node on every node permission file once the network is up (Experiment 3).

Results and conclusion

The experiment appears to be successful to demonstrate that the network permissioning system is dynamic as the newly added node is dynamically added to the system and a deleted node is dynamically revoked by the system. When a node tries to connect, the file permissioned-nodes.json is checked and if the new node is listed, the connection occurs. However, this implies that every node on the network needs to manually add or remove the new node to the `permissioned − nodes.json` file.

7.2.5 Experiments 4 and 5

This experiment is conducted in two parts to determine the whitelist dynamic node modification capabilities.

Methodology

We first evaluate the dynamic node addition capabilities (Experiment 3) and second the dynamic node deletion capabilities (Experiment 4). For that purpose, we deploy and launch the 7nodes sample script with TLS whitelist option enabled. Once fully functional, we try to add a new node in each node whitelist and assess its ability reach other nodes in the network. Second, after re-initializing the deployment conditions, we try to delete the full whitelist file at each node and evaluate resulting functioning of the network.

Results and conclusion

The experiment shows that **TLS whitelist does not support dynamic modifications**. Indeed, the first part of the experiment shows that the added node cannot reach the other network nodes, and the second part of the experiment demonstrates that the full deletion of whitelist file at each node has no impact on the network current and posterior operations.

7.2.6 Experiment 6

When using the TLS TOFU & CA mode in Quorum, the node permission is granted by verifying the validity and the chain of trust of its certificate. However, no precision is given for the procedure to follow after a certificate refresh. It appears logical that certificate refresh does not impact the system when using the CA mode on its own. However, the situation becomes more complex when it

is associated with TOFU. An investigation is conducted to determine the detail scheme of TOFU mode, and how it handles refreshed certificates.

Methodology

We generate node certificates via OpenSSL[5]. Once the system fully functional, we first refresh a node certificate using the same keys. Second, we refresh the certificate of this node with new keys to confirm the expected behavior.

Results and conclusion

First, Quorum source code relies on folders `KnownClient` and `KnownServer` which store a finger print of nodes' public keys and its hosts. As long as the user submit the same key with its CSR, it will be recognized as the previously registered host. Indeed, TOFU ensures that the keys associated with a given host do not change after the first connection. **This system thus relies on the assumption that no key rotation scheme is used. This represents a potential vulnerability as it prevents user from refreshing keys.** Even if changing keys when refreshing a certificate is not mandatory, if there is a suspicion for key leakage, the potential cost and difficulty of re-initializing all nodes would be a discouragement. **Finally, the experiment demonstrates that node is authenticated by TLS TOFU mode when his certificate has been refreshed with the same keys, but sees his access denied when using new keys as expected**.

7.2.7 Experiment 7

Similarly to experiment 6, the official documentation lacks of information regarding the revocation of a Root CA. In order to assess the system behavior when a Root CA is dynamically replaced in each node's TrustStore, the network behavior is evaluated in the following cases:

1. The Root CA has just been changed in every node configuration
2. A node loses its connection to the network (by killing its process)
3. The system is redeployed

Results

The results are presented in reference to the previously listed ordering:

1. This does not modify the ongoing or posterior network behavior
2. When relaunching Tessera node process, all nodes are still allowed to communication with each other and to submit transactions as before
3. When restarting the network, the initialization fails

[5]https://www.openssl.org/

```
ARGS="--nodiscover --istanbul.blockperiod 5 --networkid $NETWORK_ID --syncmode full
--mine --minerthreads 1 --rpc --rpcaddr 0.0.0.0
--rpcapi admin,db,eth,debug,miner,net,shh,txpool,personal,web3,quorum,istanbul"
PRIVATE_CONFIG=qdata/c1/tm.ipc nohup geth --datadir qdata/dd1
$ARGS --rpcport 22000 --port 21000 --unlock 0 --password
passwords.txt 2>>qdata/logs/1.log &
```

Figure 7.7: Node 1 default RPC parameters in 7nodes example script

Conclusion

Experiment 7 proves that Quorum Root CA revocation is not dynamic. As a consequence, all nodes in the network need to be re-initialized to revoke the current Root CA.

7.2.8 Experiment 8

This experiment aims to determine the dynamicity of the revocation process when TOFU mode is used.

Methodology

When using TLS TOFU mode, each node maintains a list which contains the public keys and the IP address of already encountered nodes. To conduct this experiment, we delete the public key of the node we want to revoke in every node TOFU listing.

Results and conclusion

This experiment shows that TLS TOFU does not support dynamic revocation. Indeed, after removing the public key of the node that we want to revoke from the TOFU listing, the node is still able to communicate with the network.

7.2.9 Experiment 9

Similarly to Ethereum Experiment 1, this experiment evaluates the default RPC and WS parameters of the nodes of 7nodes example script. The goal is to demonstrate possible attacks on weak parameters.

Default parameters

The chosen official sample script enables by default all RPC modules and force the node to listen to any interface $-$`rpcaddr`$'$`0.0.0.0`$'$. Further, all accounts are permanently unlocked by default `unlock0`. The command line which defines those parameters for node 1 as an example is depicted in Figure 7.7. We notice that WS services are not enabled by default.

Attack demonstration

The RPC vulnerabilities being identified, Curl[6] command line tool is used to remotely access node 1 and demonstrate the possible attack. First, the node

[6]https://curl.haxx.se/

is exposed to public internet by defining a static IP address and configuring DHCP as a regular node would be in industry (it will need to be accessible from other organizations outside its LAN as Quorum aims to develop intra-enterprise collaborations). The following command is used:

```
curl -H "Content-Type: application/json" -X POST --data
    {\"jsonrpc\":\"2.0\",\"method\":\"NAME OF THE
    METHOD\",\"params\":[],\"id\":23} ip_destination:22000
```

In the command, we replace name of the method by RPC APIs methods such as:

- `eth_getTransaction` with the parameter `latest` which provides us the latest transaction received by node 1.
- `eth_sendTransaction` which submit a transaction to the other nodes in the networks.
- `eth_getLogs` which returns an array of all logs from a given address of a given topic.

All attacks were successful, and the previously states methods do not present an exhaustive list of possible remote call. The full list can be found at the following address: `https://github.com/ethereum/wiki/wiki/JSON-RPC`.

Conclusion

The official Quorum example script presents a vulnerable configuration with regard to RPC API, WS API is disabled by default. Indeed, RPC is enabled by default on all nodes, with no restrictions on which connection to accept and the initialization forces the unlocking of the accounts. Using Curl via Mac Terminal, **piracy attacks and attacks which aim to access confidential information are demonstrated**. As all accounts are unlocked and all modules enabled by default, no password sniffing attacks or module-enabling attacks have been conducted.

7.2.10 Experiment 10

Experiment 10 demonstrates the possible remote attacks due to vulnerable nodes' RPC parameters. This experiment tends to determine if such attacks occur, and if so their frequency. We believe that this kind of attack would be conducted for industrial espionage purposes.

Assumption

This experiment relies on the assumption that Quorum has been designed to handle private financial transactions across enterprises. As companies' nodes need to be reachable from other companies, an acceptable assumption is to believe that each node will be exposed on a public IP. Indeed, even if defining a private network with all nodes would be possible in theory, the cost of such a network (infrastructure, labor and maintenance costs) appears to be discouraging. Further, the correct compartmentalization of this network from each enterprise network would present a challenge and open the door to other vulnerabilities.

Honeypot setup

Similarly to Ethereum Experiment 2 (7.1.4), we build a honeypot. We deploy our 7nodes network with default parameters and expose our node to public internet by defining a static IP address and configuring DHCP. The traffic is captured during twenty-four hours with the packet sniffing software Wireshark version (version 2.6.7), and filtered with the parameter `http.content_type` ==′ `application/json`′.

Results and conclusion

No packet has been captured during the experiment. **We carefully conclude that this attack is not frequent**.

7.3 Hyperledger Fabric

7.3.1 List of Conducted Experiments

1. Evaluate the official sample script default parameters for TLS authentication
2. Evaluate the Fabric CA default TLS parameters
3. Demonstrate that a majority of organizations can maliciously modify the network or channel MSP without another organization noticing it

7.3.2 Test environment Setup

We choose to deploy Hyperledger Fabric v1.3.0. following the popular official tutorial called Deploy your first network[7]. Following the instruction, we install Curl[8] v7.64.0, Docker for Mac[9] v2.0.0.0-mac81 2018-12-07, Go[10] v1.12, Node.js v10.15.2. The samples, binaries and docker images are installed via the official script. The sample script provides a setup with two organizations, each maintaining two peer nodes, and an ordering service.

7.3.3 Experiment 1

This experiment aims to find the default TLS parameters of Hyperledger Fabric official sample script.

Methodology

We inspect the TLS parameters of the deployed sample script. We look for the following parameters: `peer.tls.enabled` and `peer.tls.clientAuthRequired` for both peers and orderers.

[7] Building Your First Network
[8] https://curl.haxx.se/download.html
[9] https://www.docker.com/get-started
[10] https://golang.org/dl/

Results and conclusion

For both peers and orderers, TLS server authentication is enabled but client authentication is not required in the default parameters.

7.3.4 Experiment 2

This experiment aims to find the default TLS parameters of Fabric CA module.

Methodology

We inspect the TLS parameters of the downloaded GitHub official code[11].

Results and conclusion

We find out that by default, for both server and client modules, TLS is disabled. Indeed, server TLS is configured in `noclientcert` mode and client has TLS disabled.

7.3.5 Experiment 3

The goal of this experiment is to demonstrate a vulnerability which consists in modifying the channel MSP using a malicious majority of organizations without another organization noticing it.

Methodology

The deployed network consists in two organizations. We first add two other peers belonging to Org3 in the `docker − compose − base.yaml` file. Then, we define Org3 in the `configtx.yaml`, and for simplicity purpose we copy Org2 definition and adapt it. We then deploy our network. Once functional, we perform the following actions:

1. Channel MSP admin which is node 1 retrieve the channel configuration
2. The channel configuration is decoded using configtxlator
3. The organization is removed from the channel configuration by editing the configuration section and a Root CA is changed
4. Both the original and modified configurations are encoded using configtxlator
5. The encoding is wrapped and included in a config transaction
6. The transaction is sent by the MSP channel admin
7. Actions 1 to 7 are repeated with action 3 modified to re-add the originally removed organization

[11] https://github.com/hyperledger/fabric-ca

Results and Conclusion

During the configuration modification, Org3 peers loose peer connectivity, when readded, they automatically adopt the new MSP config without noticing that they have been removed. In conclusion, **we demonstrate that a majority of organization can maliciously modify the channel MSP config in order for another peer not to notice it. Thus, the channel does not authenticate itself to the peer**.

7.4 Corda

7.4.1 List of Conducted Experiments

The security analysis of Corda platform highlights two possible deployment schemes: either using the available Corda Business Network or building an independently managed network. However, both options require a large amount of time to be experimented as the first one requires to code a http server which hosts a list of authorized X.500 names and the second requires to code a http network map server and to provide a doorman server. In the framework of this project and relying on our Hypothesis 2, we decide to only assess the security of the Corda Dapp official sample script. We conduct the following experiments:

1. Evaluate the official sample script default TLS parameters
2. Try to participate to the Corda Business Network certificate

7.4.2 Test Environment Setup

We deploy the Corda Example Dapp v3.3 following the official tutorial[12]. Following the instructions, we install Java 8 JVM[13], IntelliJ IDEA v2018.1[14] with Kotlin plugin version 1.2.71 and Gradle 4.10[15]. When the download the sample script code from GitHub[16] (as of the 12th of December) and use Gradle to download the dependencies. We obtain a network of 3 client nodes and a notary node.

7.4.3 Experiment 1

This experiment aims to evaluate Corda official Dapp sample script TLS default parameters.

Methodology

We analyze the deployed code to find the TLS configuration.

[12]https://docs.corda.net/tutorial-cordapp.html
[13]https://www.java.com/
[14]https://www.jetbrains.com/idea/
[15]https://gradle.org/releases/
[16]https://github.com/corda/samples

Results and conclusion

By default in Corda, **mutual TLS authentication is required between nodes**. However, when looking for **TLS RPC settings, we find that it is disabled by default on the sample script version**.

7.4.4 Experiment 2

This experiment aims at deploying a node on Corda Business network.

Methodology and conclusion

First, we send an email to doorman@r3.com to ask for the trust root certificate. However, we answer highlighting the expected payment to receive this trust root was received even though it was not explicitly stated on the documentation that this action required a payment of 2500$. We cannot explore commercial blockchain platform in the scope of this thesis and thus do not deploy any node on Corda Business Network.

Chapter 8

Recommendations

This chapter presents hardening guidelines for each platform in response to the vulnerabilities identified in chapter 6. These recommendations allows to mitigate most vulnerabilities, however, some exposure cannot be prevented as they would require a change either in the platform design or implementation.

8.1 Ethereum

- The mechanisms for user remote access (RPC and WS) shall be disabled if not used. If used, an HTTP basic authentication along with a reverse proxy and a firewall shall be configured to restrict the access only to authorized users. Further, the account unlock duration shall be disabled.
- Accounts passphrase should be set with a proper security policy requirements.

No action can be taken to prevent key compromise except enforcing physical security as the choice of the elliptical curve secp256k1 is imposed by the platform.

8.2 Quorum

- TLS shall be enforced (by turning the parameter to `STRICT`) with mutual client authentication. Best configuration associate CA with a permissioned enforced network (using the $--$ `permissioned` command).
- Remote user access (RPC and WS) shall be disabled if not used. If used, a HTTP basic authentication along with a reverse proxy and a firewall shall be configured to restrict the access only to authorized users. Further, the default account unlock duration shall be disabled (set to 0).
- Default passwords for keystores, truststores and accounts shall be set with a proper password policy which enforces password complexity.
- Different keys shall be used for node identity keys and TLS keys (cryptographic material should not be reused across protocols).

8.3 Hyperledger Fabric

- A different CA shall be used for MSP identities and TLS certificates (cryptographic material should not be reused across protocols).
- A channel must contain several orderers to prevent it from being a single point of failure.
- TLS mutual authentication between nodes shall be enforced to prevent DDoS and DoS attacks.
- A peer shall not join a channel in which he trusts fewer than 50% of the participants to prevent an attack in which other participants maliciously modify the channel MSP.

8.3.1 Fabric CA

- TLS mutual authentication shall be enforced between Fabric CA client and server.
- Fabric CA TLS certificate shall be generated from a dedicated CA that has not been used to generate certificates for any organizations in any channels.
- The Fabric CA variable $--$ `maxenrollements` shall be given a particular attention, and be set to 0 if the new peer is not supposed to be able to enroll anyone. -1 should be avoided as it allows to enroll an infinite number of peers.

8.4 Corda

- Notaries shall always be deployed in clusters to prevent them from being single points of failure.
- TLS mutual authentication between nodes shall be enforced.
- The parameter `crlCheckSoftFail`shall be set to false, forcing the system to fail if the required CRL list is unavailable.
- Only Dapp for which the developer is trust shall be used to prevent an elevation of privilege attack.
- Remote user access (RPC) shall be disabled on every node if unused. If used, a proper password policy which enforces password complexity shall be set and TLS server authentication shall be enforced.
- When building a network map sever, HTTPS shall be enforced.
- Node identity default key generation scheme shall be changed to the curve secp256r1 when using Corda Utility.

Chapter 9

Discussion

We began this research by making the assumption that users tend to adapt their blockchain from existing sample scripts rather than building it from scratch. We would have liked to prove this hypothesis by conducting a large-scale scan of the public internet to look for blockchain clients listening to the same ports as defined in the official sample scripts. That would have validated or disprove our hypothesis, and security test on those ports would have helped us assess the default security mechanisms deployed in industry. However, we could not conduct such experiments for legal reasons.

Authentication and authorization schemes described correspond to the latest release of each chosen blockchain platform. However, as the studied platforms are recent, their design and implementation are often subject to change. This observation implies that the schemes depicted in this thesis will probably not remain relevant for future versions.

The choice of the experiment environment setups relies on our first hypothesis which states that the most deployed platform versions are the most likely to be targeted by attackers. As a consequence, we chose an official sample script for each platform and deployed it in order to conduct our experiment. However, we could not find exact data about the popularity of a given example script which induced a subjective choice. Further, the interest for a given demo might vary over time, and there is no guarantee that these demos will not have been replaced in the future.

The vulnerability analysis rests on the current state-of-the-art. Some cryptographic techniques that are known to be secure today might be broken in the future. Additionally, we consider as vulnerable any scheme that has successfully been broken, but some attacks are still complex to perform and might require specific access, and thus could not be demonstrated in the scope of this thesis. Most platforms allow users to either import their own keys or generate it directly, as a consequence, we assess the platform key generation schemes but could not be exhaustive about the security of imported keys.

Finally, we tried to provide the most generic threat model possible with regard to blockchain authentication and authorization schemes. However, this thesis shows that each platform has a singular architecture and thus this model might not perfectly cover the potential threats of platforms that are out of this thesis scope.

Chapter 10

Conclusion

We conduct a study of both the design and implementation of the authentication and authorization mechanisms existing in four blockchain platforms widely used in industry: Ethereum, Quorum, Hyperledger Fabric and Corda. Despite the complexity and sparseness of available documentation, the results of several experiments that we ran provide us with the missing information required to provide a detailed description and demonstrate several vulnerabilities.
Even though the authentication and authorization mechanisms studied appear to be platform-specific, our security analysis shows recurrent vulnerabilities among platforms such as: the usage of both weak keys and weak or non-existing remote user authentication schemes, the absence of password policy to enforce the definition of strong passwords and the lack of sandboxing of smart contract execution which often enables malicious smart contracts to conduct elevation of privilege attacks.
Further, we find that communication encryption via TLS is optional in all permissioned blockchain studied thus often causing potential exposure to man-in-the-middle, session hijacking, DDos and Dos attacks.
The execution of official sample scripts for experimental purpose shows us that weak default parameters are frequently preferred to ease software adoption and functionality demonstrations irrespective of the consequences on security. With regard to our hypothesis that users tend to adapt existing sample scripts to deploy their own systems, we believe that this lack of security in official sample scripts induces the deployment of vulnerable systems in industry.
Despite the fact that blockchain platforms evolve rapidly and therefore described schemes and vulnerabilities in this thesis might become obsolete in the near future, our threat model provides a comprehensive baseline of potential vulnerabilities with regard to blockchain authentication and authorization mechanisms that will ease both updates of this material and similar security assessments.

Bibliography

[1] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* [Online; accessed 25-June-2018]. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[2] Stallings, William. *Cryptography and Network Security: Principles and Practice.* 1990.

[3] D. Richard Kuhn, Vincent C. Hu, W. Timothy Polk, Shu-Jen Chang. *Introduction to Public Key Technology and the Federal PKI Infrastructure, NIST.* [Online; accessed 14-February-2019]. 2001. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-32.pdf.

[4] Eliza, Paul. *What is Digital Signature- How it works, Benefits, Objectives, Concept.* [Online; accessed 14-February-2019]. 2017. URL: http://www.emptrust.com/blog/benefits-of-using-digital-signatures.

[5] Diffie, W., and Hellman, *M. New directions in cryptography. IEEE Trans. on Inform. IT-22, 644-654.* 1976.

[6] M. E. Kabay. *Identification, Authentication and Authorization on the World Wide Web.* [Online; accessed 14-February-2019]. 1997. URL: http://www.mekabay.com/infosecmgmt/iaawww.pdf.

[7] D.Antoniou, K.Socha. *CERT-EU Security Whitepaper 16-003 Authentication Methods.* [Online; accessed 14-February-2019]. 2016. URL: http://cert.europa.eu/static/WhitePapers/CERT-EU%5C%20SWP-16_003_Authentication%5C%20Methods.pdf.

[8] Farnaz Towhidi, Azizah Abdul Manaf, Salwani Mohd Daud, Arash Habibi Lashkari, Advanced Informatics School, Universiti Technologi Malaysia (UTM), Kuala Lumpur, Kuala Lumpur, Malaysia. *The Knowledge Based Authentication Attacks.* 2011. URL: https://s3.amazonaws.com/academia.edu.documents/38218671/SAM8123.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1550672317&Signature=WGr47Yg%5C%2BM95mnHf%5C%2FAy0mmnRZMuE%5C%3D&response-content-disposition=inline%5C%3B%5C%20filename%5C%3DThe_Knowledge_Based_Authentication_Attac.pdf.

[9] Jesudoss A., Research Scholar, Faculty of Computer Science and Engineering, Sathyabama University, Chennai, Tamil Nadu, India. *A survey on authentication attacks and contermeasures in a distributed environment.* 2004. URL: http://www.ijcse.com/docs/INDJCSE14-05-02-061.pdf.

[10] Maria Nickolova, Eugene Nickolov. *Threat model for user security in e-learning systems, International Journal Information Technologies and Knowledge Vol.1.* [Online; accessed 14-February-2019]. 2007. URL: https://pdfs.semanticscholar.org/35f8/407804ff45ede9305982956dde98a559e6f7.pdf?%5C_ga=2.187336296.1345382057.1551434913-1643474519.1550584578.

[11] Dobromir Todorov. *Mechanics of User Identification and Authentication, Auerbach Publications.* 2007.

[12] Morrie Gasser, Andy Goldstein, Charlie Kaufman, Butler Lampson Digital Equipment Corp. *The Digital distributed system security architecture.* [Online; accessed 14-February-2019]. 1989. URL: http://www.cs.cornell.edu/people/egs/cornellonly/syslunch/fall04/ddss.pdff.

[13] Andrew D. Birrell, Butler W. Lampson, Roger M. Needham, and Michael D. Schroeder. *A Global Authentication Service without Global Trust, Proceedings of the 1986 IEEE Symposium on Security and Privacy, IEEE Computer Society,* 1986.

[14] Halima Akhter, Md. Ansarul Haque. *Comparative analysis of authentication and authorization security in distributed system.* [Online; accessed 14-February-2019]. 2014. URL: https://ijret.org/volumes/2014v03/i11/IJRET20140311067.pdf.

[15] Shafi Goldwasser, Silvio Michali, Ronald L. Rivest. *A Digital Signature Scheme Secure Against Adaptative Chosen-Message Attacks.* [Online; accessed 14-February-2019]. 1995. URL: %5Curl%7Bhttps://people.csail.mit.edu/rivest/GoldwasserMicaliRivest-ADigitalSignatureSchemeSecureAgainstAdaptiveChosenMessageAttacks.pdf%7D.

[16] David Pointcheval. *Practical Security in Public-Key Cryptography, Proceedings of the 4th International Conference on Information Security and Cryptology'01 (7 December 2001, Seoul, South Korea).* [Online; accessed 14-February-2019]. 2001. URL: https://www.di.ens.fr/david.pointcheval/Documents/Papers/2001_icisc.pdf.

[17] Bryan Ford. *So They're Selling You a Blockchain.* [Online; accessed 11-February-2019]. 2018. URL: http://bford.info/2018/09/11/blockchain/.

[18] Hartwig Mayer, CoinFabric. *ECDSA Security in Bitcoin and Ethereum: a Research Survey.* [Online; accessed 14-February-2019]. 2016. URL: https://pdfs.semanticscholar.org/1785/6bad4335c8ca7419aab2c715ea25ce5e0621.pdf.

[19] Xu Wang, Xuan Zha, Guangsheng Yu, Wei Ni, Senior Member, IEEE, Ren Ping Liu, Senior Member, IEEE, Y. Jay Guo, Fellow, IEEE, Xinxin Niu, and Kangfeng Zheng, School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, China, Global Big Data Technologies Centre, University of Technology Sydney, Australia, Data, CSIRO, Sydney, Australia, State Key Laboratory of Public Big Data, Guizhou, China. *Attack and Defence of Ethereum Remote APIs*. [Online; accessed 14-February-2019]. 2014. URL: https://www.researchgate.net/profile/Xu_Wang166/publication/330351717_Attack_and_Defence_of_Ethereum_Remote_APIs/links/5c3b4939a6fdccd6b5a9e381/Attack-and-Defence-of-Ethereum-Remote-APIs.pdf.

[20] Graham Shaw. *Security Assessment Management Report*. [Online; accessed 14-February-2019]. 2017. URL: https://www.hyperledger.org/blog/2018/02/07/hyperledger-fabric-1-0-release-process.

[21] Christian Cachin. *Architecture of the Hyperledger Blockchain Fabric*. [Online; accessed 14-February-2019]. 2016. URL: https://pdfs.semanticscholar.org/f852/c5f3fe649f8a17ded391df0796677a59927f.pdf.

[22] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich (IBM), Srinivasan Muralidharan (State Corps), Chet Murthy, Binh Nguyen (State Corps), Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolifffdfffd, Sharon Weed Cocco, Jason Yellick (IBM). *Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains*. [Online; accessed 14-February-2019]. 2018. URL: http://delivery.acm.org/10.1145/3200000/3190538/a30-androulaki.pdf?ip=194.2.202.93&id=3190538&acc=TRUSTED&key=4D4702B0C3E38B35%5C%2E4D4702B0C3E38B35%5C%2E4D4702B0C3E38B35%5C%2EE47D41B086F0CDA3&__acm__=1550680662_65369776cf8e8c1322b6c03fb724c03c.

[23] Dr Garrick Hileman & Michel Rauchs. *Global Blockchain Benchmarking Study, Cambridge Centre for Alternative Finance, EY*. [Online; accessed 13-February-2019]. 2017. URL: https://www.ey.com/Publication/vwLUAssets/ey-global-blockchain-benchmarking-study-2017/%5C$FILE/ey-global-blockchain-benchmarking-study-2017.pdf.

[24] Dr. Gavin Wood. *Ethereum: a secure decentralised generalised transaction ledger Byzantium version*. [Online; accessed 14-February-2019]. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

[25] *Quorum Whitepaper, JPMorgan*. [Online; accessed 14-February-2019]. 2016. URL: https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum%5C%20Whitepaper%5C%20v0.1.pdf.

[26] Neriman Gamze Orhon, Ayham Taleb.
Parallel Pollard's Rho Attack for Elliptic Curve Cryptography.
[Online; accessed 14-February-2019]. 2014.
URL: `https://www.academia.edu/29788543/Parallel_Pollards_Rho_Attack_for_Elliptic_Curve_Cryptography`.

[27] Yuval Marcus, Ethan Heilman, Sharon Goldberg.
Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network.
[Online; accessed 14-February-2019]. 2018.
URL: `https://www.cs.bu.edu/~goldbe/projects/eclipseEth.pdf`.

Acronyms

ABAC Attribute-Based Access Control. 38, 43

ACL Access Control List. 31, 35, 38, 50

API Application Programming Interface. 26, 27, 30–32, 38, 39, 75

CA Certificate Authority. 8, 15, 31, 39–43, 46–49

CRL Certificate Revocation List. 15, 42, 43, 49, 73

CSR Certificate Signing Request. 42, 49, 83

DDoS Distributed Denial of Service. 54, 56–59, 61–70, 72, 91

DoS Denial of Service. 54, 56–59, 61–70, 72, 91

ECDSA Elliptic Curve Digital Signature Algorithm. 28, 29, 32, 34, 39, 48

MITM Man-in-the-middle. 56, 57, 61, 62, 65, 69–71

MSP Membership Service Provider. 42

OCD Orange Cyberdefense. 10

PKI Public Key Infrastructure. 14, 15

RPC Remote Control Protocol. 21, 28, 30, 52, 73–75

TLS Transport Layer Security. 8, 31, 33, 34, 39, 41–43, 46, 48–50, 52, 61, 65–72, 87

WS Web Services. 21, 74, 75