JusticeChain: Using Blockchain To Protect Justice Data

Rafael André Pestana Belchior

Thesis to obtain the Master of Science Degree in

Information Systems and Software Engineering

Supervisor(s):	Prof. André Ferreira Ferrão Couto e Vasconcelos
	Prof. Miguel Nuno Dias Alves Pupo Correia

December 2019

Acknowledgments

This thesis was only possible to write due to the contributions and patience of several people. To all, I thank you:

- Professor André Vasconcelos and professor Miguel Correia, my advisors, for their continuous support, guidance, unlimited patience and encouragement along all this journey. For the patience, respect and trust towards my way of working.
- Carlos, my father, Cecília, my mother, Daniel, my brother and Ana, my sister, for all the support and love, throughout my entire life.
- Maria Pestana, Paulo Conceição, Tiago Conceição, Raquel Conceição, and Rita Conceição, for being my closer family.
- Ângelo Gonçalves, Tiago Araújo, Ricardo Ribeiro, Delfim Vieira and Gonçalo Magalhães, for all the support, comradeship and for having the courage for being my housemates for five years.
- Filipe Alves, Daniel Ramos, Jorge Pereira, Inês Sequeira, Ricardo Xavier, João Calisto, Pedro Gomes, Luís Aguiar, João Trindade, Pavlo Kovalchuk and Guilherme Quintino for being my friends and Técnico colleagues. Without you, it would be much more challenging to endure university.
- Pedro Grilo, Adilson Teixeira, Catarina Cachola, José Lavado, Daniel Covas, André Bucho, Pedro Canhoto, Ana Brava, Rúben Calado, João Teles, Ana Saraiva, Aurélio Martins, Carlos Pratas, David Brotas, Inês Rodrigues, João Carpinteiro, João Fernandes, João Madeira, Jorge Rodrigues, José Barroso, José Laço and Tiago Pires for your friendship and for making Elvas such a special city.
- Rui Henriques, whom I consider a mentor and an example to follow. His advice, patience, and empathy made all the difference in my academic and personal path.
- Rui Cruz, for introducing me to the teaching world.
- Nuno Fonseca and Carlos Brito, for introducing me the Portuguese justice and their projects. For all the support, guidance, suggestions and help.

To them, and the dear reader, I dedicate this thesis with respect and honesty.

Lisbon, October 2019

Rafael André Pestana Belchior

Resumo

A auditabilidade de sistemas de informação é fulcral na administração pública. Os acessos feitos a recursos geridos pelos sistemas de informação são guardados em ficheiros log, para mais tarde poderem ser analisados por auditores. No entanto, há dois problemas na forma em como os logs convencionais são geridos: i) os logs são vulneráveis a ataques, onde os adversários podem modificar os dados, sem serem detetados e ii) podem existir várias partes interessadas com diferentes papéis e níveis de confiança, com diferentes direitos de acesso à informação. Este é o caso no sistema judicial português, em que os utilizadores finais utilizam sistemas de informação geridos por terceiros. Este trabalho propõe utilizar a tecnologia blockchain para tornar o armazenamento de logs aplicacionais mais resiliente, e ao mesmo tempo suportar um caso com várias partes interessadas, em que diferentes entidades têm diferentes direitos de acesso aos dados, nomeadamente logs. Esta proposta é implementada no sistema judicial português através do JusticeChain. O JusticeChain é dividido na componente blockchain e na componente cliente da blockchain. A componente blockchain, baseada em Hyperledger Fabric, garante integridade dos logs, e aumenta a sua resiliência. O cliente da blockchain, JusticeChain Client, permite guardar logs aplicacionais, e é composto pelo JusticeChain Log Manager e o JusticeChain Audit Manager. O último permite auditorias mediadas pela blockchain. A avaliação do sistema mostrou que este consegue uma taxa de transferência de 37 transações por segundo, e uma latência menor que 1 minuto. O armazenamento necessário é na ordem dos terabytes por ano, por cada nó da blockchain. Propomos uma extensão do JusticeChain, JusticeChain v2.0, um sistema de controlo de accessos baseado em blockchain que permite distribuir mais confiança para os stakeholders. JusticeChain v2.0 permite distribuir o processo de autorização, enquanto providencia as mesmas vantages que o JusticeChain. A nossa avaliação mostra que o nosso sistema pode suportar cerca de 250 pedidos de acesso por segundo, com uma latência menor que 12 segundos. O armazenamento necessário é aproximadamente o mesmo que no JusticeChain.

Palavras-chave: blockchain, auditoria, logs de auditoria, administração pública

Abstract

The auditability of information systems plays an important role in public administration. Information system accesses to resources are saved in log files so auditors can later inspect them. However, there are two problems with managing conventional audit logs: i) audit logs are vulnerable to attacks where adversaries can tamper data, without being detected and ii) there can be distinct stakeholders with different roles and different levels of trust with different access rights to data. This scenario happens in the Portuguese judicial system, where stakeholders utilize an information system managed by a third-party. This document proposes using blockchain technology to make the storage of access logs more resilient while supporting such a multi-stakeholder scenario, in which different entities have different access rights to data. Towards that, we implemented this proposal in the Portuguese judicial system through JusticeChain. JusticeChain is divided into the blockchain components and blockchain client components. The blockchain components, implemented with Hyperledger Fabric, grant log integrity and improve its resiliency. The blockchain client component, JusticeChain Client, is responsible for saving logs on behalf of an information system and comprises the JusticeChain Log Manager and JusticeChain Audit Manager. The latter allows audits mediated by the blockchain. The evaluation results show that the system can obtain a throughput of 37 transactions per second, and latency lower than 1 minute. The required storage for each peer, for a year, is in the order of terabytes. As an extension of JusticeChain, which achieves even more trust distribution, we present a blockchain-based access control system, JusticeChain v2. JusticeChain v2 allows distributing the authorization process while providing the same advantages as JusticeChain. Our evaluation shows that our system can handle around 250 access control requests per second, with a latency lower than 12 seconds. The storage required is approximately the same as JusticeChain.

Keywords: blockchain, auditing, audit logs, public administration

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Organizations have the responsibility of protecting their sensitive data, a valuable resource that often guides business decisions. *Access control* mechanisms aim to identify subjects that require access to resources and allow or deny them the access, based on the context of the request [20, 39]. Digital traces are recorded in *log files*. Log files, or *audit logs*, enable *auditors* to audit the system's state, monitoring users and provide user accountability concerning their roles, performance and permissions. This way, one can verify if the involved parties are using the system for illegal purposes or to gain an unfair advantage. Audit logs are used by data-sensitive systems for logging activities on a database and can be used as a way to back up the system state. Since audit logs are typically saved on conventional databases, they reflect a centralized client-server model of communication, constituting a single point of failure.

At the Portuguese judicial system, there is an information system to manage judicial processes at courts that supports several stakeholders. The entity that maintains that the system faces different incentives from the stakeholders that use it, leading to a multi-stakeholder scenario with uncertain trust among them. In such a scenario, separate entities have different access rights to data. Threats to *log integrity*, like log file tampering, have to be minimized, as they can invalidate audits as tampered data cannot be trusted [12, 21]. If logs are not tamper-proof, attackers may delete their traces, which may result in permanent information loss. As a consequence, attackers can hide illegal activities from the auditors. Standard solutions to ensure log integrity, such as checksums [13], or database replication [55] do not tackle the need for *trust distribution* and *disintermediation.* Moreover, often there are users with root access to such databases, who can edit its entries directly, and thus there is the need for implementing a stronger security model, based on the immutability and distribution of audit logs.

Blockchain technology has emerged as a vehicle for decentralization, transparency and trust while conserving security, privacy and control, which can leverage auditability and trust distribution [60], both critical requirements for information systems at public administration. This research proposes the use of *blockchain* technology as a solution to overcome the problems concerning the integrity of logs and access by several stakeholders. The blockchain can replicate the audit logs across nodes which represent the various stakeholders. Hence, audit logs can be granted desirable properties such as immutability, tamper-proof, and integrity, allowing, us to build a transparent and collaborative network.

In particular, we introduce JusticeChain[1], a system to store, protect and decentralize applicational logs built on top of a permissioned, private, blockchain, *Hyperledger Fabric* [3], while distributing their access to the stakeholders. JusticeChain is a modular system with several components that takes advantage of Fabric to allow flexible management of the stakeholders. JusticeChain receives log entries from a set of *oracles*, processes them at the Log Manager component, and acts as a client to the underlying Hyperledger Fabric infrastructure. The Audit Log Manager component can be used by authorized auditors to read logs from the blockchain. We present an experimental evaluation of JusticeChain using *Hyperledger Caliper* [27], a project that enables blockchain load testing. After that, JusticeChain v2 is presented, as an incremental improvement over JusticeChain. JusticeChain v2 is a blockchain-based access control system, that regulates the authorization process, creates and maintains applicational logs.

1.1 Motivation

The *Instituto de Gestão Financeira e Equipamentos da Justiça* (*IGFEJ*) manages the financial, patrimonial and technological resources of the Portuguese Ministry of Justice. In particular, it provides the infrastructure and systems administration to enable several business processes, some of them pivotal for the proper functioning of the Portuguese courts. With a budget of around 450 million euros and a staff composed by 372 human resources (2019)[2], it manages several systems, like the *Citius* system, which supports judicial processes, and *SITAF*, which supports the administrative and fiscal courts. In 2018, users allegedly fraudulently used the system Citius, as could be seen on the news[3]. Although there are some guarantees concerning the integrity and availability of applicational logs that are audited, one could improve the current status.

Furthermore, there is a concentration of responsibilities by IGFEJ, as it controls the authentication, authorization and accountability processes of systems used by stakeholders with different incentives. Specific IGFEJ teams responsible for each information system manage the

[1]https://github.com/RafaelAPB/JusticeChain
[2]https://igfej.justica.gov.pt
[3]https://observador.pt/especiais/ como-a-toupeira-do-benfica-foi-apanhada-pela-justica

authorization process locally. This fact leads to two problems: i) there is space for interoperability issues, as teams from each information system are self-organized and independent from others and ii) conflict of interests, where several parties have different incentives. In particular, as the judicial power is independent of the political power, the users of the systems (judicial power) do not report to the administrators of such systems (political power). We propose JusticeChain to address the problems related to data integrity and to audit purposes and JusticeChain v2 to mitigate the problems i) and ii).

1.2 Objectives

Considering the existing data protection and access control solutions and the potential of the blockchain technology, based on its intrinsic properties, *the main objective of this work is to increase the resiliency of applicational logs.* Furthermore, access to protected applicational logs should be mediated, and the information produced by such accesses should also be protected. The solution should tackle several challenges:

- How to guarantee the integrity of the applicational logs of information systems which are administrated by a third-party? In particular, how to guarantee that applicational logs from information systems administered by IGFEJ are tamper-proof?
- How to distribute responsibility to the involved stakeholders?
- How to make sure that such logs are not accessed unduly?

We yield several contributions not only to the Portuguese but also for the blockchain ecosystem. In particular, this thesis proposes a systematic analysis of some security risks at the Portuguese justice, concerning data-tampering threats on applicational logs. Although research has been done on securing data integrity, the case at the Portuguese justice is more complicated: there is a multi-stakeholder scenario, in which each participant has limited trust in others. This fact turns the challenge of protecting applicational logs into the challenge of protecting such data in a distributed way. In other words, there is the need to assure all parts involved that the logs being stored are i) immutable, ii) can be accessed by authorized parties under exceptional circumstances and iii) facilitates trust between participants. As an additional contribution to JusticeChain, and to distribute trust on a larger scale, JusticeChain v2 tackles several challenges:

- Can we solve the root issues with access control, namely the local, centralised authorisation processes performed in different systems administrated by IGFEJ?

- Is there a possibility of exploring synergies between the access control systems and data integrity, namely log entries integrity?
- Ultimately, how to provide a secure, decentralised, scalable access control system, that is more robust and distributes trust concerning authorisation, while assuring logs' integrity and availability?

The specific needs under a specific context foster the usage of innovative technologies that can address them. To answer such threats and achieve disintermediation, we propose using blockchain technology to tackle such scenario: assuring data integrity, while distributing accountability, responsibility, and trust for that data.

1.3 Dissertation Outline

The remainder of the document is structured as follows: Chapter 2 presents background about the blockchain technology, audit logs, access control and blockchain access control. Chapter 3 analyses the context at IGFEJ, providing a list of requirements and a threat model. Next, it presents the system model and data model of JusticeChain. Lastly, it discusses JusticeChain's architecture and implementation details. Chapter 4 presents the system model, architecture and the implementation of JusticeChain v2. Chapter 5 presents an evaluation methodology to assess JusticeChain and JusticeChain v2. The results from the theoretical and experimental evaluation are presented and discussed. Next, we compare JusticeChain with JusticeChain v.2.0, providing a discussion on its similarities and differences. Finally, Chapter 6 concludes this dissertation, fostering future works.

Chapter 2

Background and Related Work

This chapter presents background about blockchain, audit logs, access control, and blockchain-based access control. First, as a viable solution that can address these problems, the blockchain concept is introduced, as well as its characteristics. Permissionless and permissioned blockchain infrastructures are be briefly discussed and compared, in order to choose the most appropriate to our use case. Next, *Hyperledger Fabric* are be discussed in detail, including its architecture and components. After that, we present related work considering the usage of blockchain technology to protect logs. After that, this chapter reviews access control models. Next, we review the state-of-the-art study of the current solutions that use blockchain for access control. Finally, we present the state-of-the-art at IGFEJ, such as the systems they administrate, the different participants in the ecosystem, their access control mechanisms, their main issues related to access control, and the system architecture of Citius, the information system that we focus on.

2.1 Blockchain

A *blockchain* is a transparent, persistent append-only distributed ledger, which contains records grouped into *blocks*. It is a distributed ledger, which allows trusted transactions amongst untrusted participants [47]. For the sake of simplicity, we make no distinction between blockchain and distributed ledger technology.

A core concept of a blockchain is the *transaction*. The term transaction refers to the first blockchain created, Bitcoin [1]. As Bitcoin was created to provide solutions on the financial scope, one can use the term *record* to refer to transactions on non-financial applications. A record is a collection of data. Records form blocks and blocks form a chain, hence the name blockchain. Each block has a timestamp and a hash of the previous block on the chain. For a block to

[1]http://bitcoin.org/bitcoin.pdf

be added to the blockchain, each record that belongs to it has to be validated. The validation process assures that records and thus, blocks are legal.

All entities that participate in distributed data storage form a network of nodes. Nodes are computing entities that communicate with other nodes, to complete transactions. In a network, nodes can also be called participants. Participants in the network agree on the state of the distributed ledger, and there may be malicious nodes. For instance, one node might try to append a block which is not valid to the ledger. The process of agreeing to a global state is called *consensus*. Participants of the network reach *consensus* using a consensus algorithm. In the context of blockchain technology, the first consensus mechanism is proof of work (*PoW*). PoW is the consensus mechanism for *Bitcoin*[2]. PoW consists of transaction validators (called *miners*) solving a resource-demanding cryptographic puzzle. In particular, miners need to calculate the hash of a random nonce which starts with a certain number of zeroes. As the process of maintaining consensus is expensive, the network monetarily incentivizes miners, rewarding the first miner that solves the puzzle. Thus, the first miner builds a block containing valid transactions and is awarded cryptocurrency (in this case, bitcoin). The fact that PoW is expensive, and that attackers are not rewarded bitcoin (as the network invalidates their transactions), makes the network resilient.

For different blockchains, one might see different consensus algorithms and different reward mechanisms, especially in *permissionless blockchains*, as they rely on the contribution of anonymous nodes to maintain the ecosystem. In general, the blockchain technology presents a set of properties that ensure integrity, authenticity, transparency, traceability, and auditability, that are attractive to solve a specific set of problems.

The blockchain technology is not only an alternative to centralized banking but also enables new architectures, as it can be used as a software connector [56]. One of its usages is as a communication service, as it is a decentralized alternative to the current centralized data storage, improving information traceability and transparency. Regarding integrity and authenticity, the system enables granular access control in conjunction with privacy and transparency.

Even though it might seem an appropriate fit for the problem tackled in this thesis, there are some inherent issues with this technology. It suffers from challenges such as security, scalability, privacy and governance issues, especially on *public blockchains*. Concerning Bitcoin, it requires an attacker a lot of computational resources to corrupt the digital ledger, due to the usage of hashes and private-public key pairs. Attacks such as the 51% Attack, the Brute Force Attack, Goldfinger, or Fork After Withholding are possible, although unlikely [14]. Risks associated

[2]https://bitcoin.org/bitcoin.pdf

with each blockchain platform are dependent on its implementation and, therefore, have to be addressed accordingly. For instance, in Hyperledger Fabric, a permissioned blockchain, the 51% Attack does not occur, as transactions are endorsed following an endorsement policy. Another problem, depending on the framework, is performance. For high scalability and high throughput requirements, the blockchain might not be the most suitable solution, as it is often lower than in traditional databases. Finally, even in a private blockchain, it is not apparent how to assure users' privacy, because, in theory, transactions are immutable.

2.1.1 Permissionless blockchains

Permissionless blockchains can be accessed and utilized by anyone with Internet access. Typically, in such networks, the participants are rewarded, self-sustainable, open-source and, therefore, have more support from the community. The first entirely decentralized digital currency, *Bitcoin*, is a permissionless peer-to-peer version of electronic cash, which would allow transactions to go directly from one party to the other, excluding the need for any intermediary. Bitcoin quickly gained popularity because it solved the famous *double-spending problem*, without the need of any central authority [57]. In the Bitcoin context, the records that miners append to the ledger are called transactions. The transactions occurring in the network are accessible to everyone. Miners, who constitute a specific class of participants, aggregate transaction requests from users, validate them and add them to the network. Given the decentralized nature of Bitcoin, where nodes can be malicious, there is the need to use a consensus algorithm. The problem of consensus often arises on distributed systems, where its agents may not agree on data needed for computation. To ensure proper behaviour, Bitcoin uses a probabilistic consensus algorithm called proof of work, in which miners have to compute a string which hash starts with a certain number of zeroes. As the consensus is probabilistic, it is possible for more than one block to be generated almost at the same time, competing for the same *blockheight h.* There may be situations in which different nodes accept different blocks for the same block height, resulting in a *fork.* Proof of work ends up being computationally very expensive, and has a poor performance: the expected block-mining rate is one block every 10 minutes, with around six transactions per second [56]. Effectively, permissionless blockchains have an expensive cost per transaction, as there is a trade-off between security and performance. When comparing Bitcoin and Ethereum[3] to VISA, we observe that the first can handle 3-20 transactions per second, while the second can handle on average 2000 transactions per second [56]. Besides this, Bitcoin was only meant to serve a particular purpose: facilitate digital transactions. Since Bitcoin is

[3]https://github.com/ethereum/wiki/wiki/White-Paper

a single-purpose blockchain, it motivated the appearance of another permissionless blockchain, *Ethereum.* Ethereum aims to be a decentralized peer-to-peer cryptocurrency and to support blockchain applications other than financial. The Ethereum blockchain is a decentralized platform which provides a runtime environment for smart contracts called Ethereum Virtual Machine (EVM), a distributed computer system that executes code. In Ethereum, groups of transactions are called blocks, like in Bitcoin. The transactions are signed with the owner's private key. Once they are signed, participants send them to *validators.* Validators are nodes that validate records and group them into blocks and are rewarded the transaction fees when a block is validated. Blocks are then propagated, flooding the network [56]. The consensus algorithm that Ethereum is based on is proof of work, although it is going to changed to *proof of stake (PoS).* In proof of stake, validators take turns proposing and voting on the next block to be added to the chain, and the weight of each validator's vote depends on the size of its stake (deposit). Significant advantages of PoS include security, reduced risk of centralization, and energy efficiency. The novelty introduced by Ethereum, the one that is particularly interesting to address our problem is the *smart contract* concept.

Smart contracts may be written in domain-specific languages such as Solidity. Clients upload smart contracts on the blockchain after being compiled into *bytecode*, a low-level programming language that is run by *Ethereum Virtual Machine (EVM).* In Ethereum, every contract has its *storage.* Only the corresponding contract can change the storage. Transactions run against a smart contract and can cause updates in the data storage. For smart contracts on blockchain systems such as Ethereum, access to real-world data is critical, as some applications need real-time information. It is not trivial to load external data in smart-contracts. Nonetheless, some efforts have been made in order to try to provide trustworthy data to smart contracts, through oracles [58].

Permissionless blockchains deliver a transparent, auditable network, although privacy is not assured, as they are opened for everyone to join and see the transactions. Drawbacks include:

- *Scalability.* The state of the blockchain should be kept in every node of the network. Bitcoin and Ethereum only can process 3-20 transactions per second, on average [56]. In the Bitcoin case, the latency for a transaction is around 10 minutes. The throughput is majored by its block size divided by the block interval [15]. As Bitcoin regulates itself dynamically, and the trend is the block size to increase, it arrises some challenges, as throughput tends to diminish. In the Ethereum network, requests' are executed in the order of consensus, which is sequential. Sequential execution translates into lower throughput, and higher latency, which translates in fewer transactions per second.

- *Privacy.* This concern is greater in public blockchains, where all data is accessible to everyone, permanently. Data privacy is important in any secured system since users' data is considered as an asset to both organizations and individuals.

- *Storage.* According to Kaspersky Lab, more than 200Gb of data has been generated by the Ethereum network in two years [4]. If the current growth rate stays stable, the blockchain's lifespan is limited due to storage capacity, as miners need to have the whole transaction history locally. A possible solution is the usage of thin clients that only download the headers of the blocks belonging to a blockchain.

2.1.2 Permissioned blockchains

Permissioned blockchains are blockchains where the participants are identified and have access rights to participate in it [53]. Inversely to public blockchains, in which the control and confidence are distributed to possibly unknown parties, private blockchains are the property of an organization or consortium of organizations. If more than one organization is included in the management of the blockchain, it is called a *community* or *federated blockchain.* A permissioned solution seems suitable for companies aiming for competitiveness that blockchain technology can offer, while protecting sensitive information. Write permissions are kept centralized to specific nodes within the organization. Therefore, a trade-off between control and decentralization exists. On a technological point of view, compared to public blockchains, there is an aggravated risk of subversion, as there are fewer participants in the network that validates transactions. Nevertheless, in practice, as the network only allows identified participants, it is easier to identify a culprit.

Taking advantage of programmable transactions, supported by multiple blockchain frameworks, the organization responsible for the blockchain administration can define the business logic and business rules that manage the blockchain. Some examples of permissioned blockchain systems include Hyperledger Fabric [3], R3 Corda [5], and Multichain [6]. Permissioned blockchains, in particular Fabric, is tackled in greater detail on Section 2.2.

2.1.3 Blockchain Infrastructures Comparison

Public blockchains allow any node to access the network and give its contribution. As every the node has access to the records, there is a privacy concern to the problem discussed. Although it is possible to have a public permissioned blockchain, it is not the most desirable case, because there

[4]https://www.kaspersky.com/blog/bitcoin-blockchain-issues/18019/
[5]http://www.r3.com/reports/the-corda-platform-an-introduction-whitepaper/
[6]https://www.multichain.com/white-paper/

might be sensitive information to be stored. Furthermore, they usually are poor performers, with a low rate of *transactions per second (tps)*, as the consensus mechanisms are expensive.

Permissioned blockchains guarantee that only authorized nodes can access and visualize the information on the network. For this reason, most of the compared elements are permissioned blockchain solutions. Regarding scalability, in a permissionless, public blockchain, the throughput is 3-20 transactions per second, whereas using a permissioned blockchain, it can be significantly higher [56]. The table 2.1 compares Fabric, Ethereum, *Quorum* and Corda. This comparison introduces some concepts, such as zero proof of knowledge (ZPK). ZPK is a method by which a party can prove to another their knowledge about a fact without disclosing any other information than that. Table 2.1 compares several blockchain infrastructures, with regard to metrics such as throughput, consensus, access control and data privacy features.

Metrics	Fabric (permissioned)	Ethereum (permissionless)	Quorum (permissioned)	R3 Corda (permissioned)
Throughput	More than 2000 tps	Around 200 tps	Around 100 tps	Around 170 tps
Consensus	Pluggable: - Trusted Solo - Crash fault tolerant Kafka - Raft	Proof of work	Pluggable: - Raft consensus - Istanbul BFT	Pluggable: - Trusted Solo - Raft
Database (DB)	LevelDB , CouchDB	levelDB	levelDB	H2 database
Access Control	Organization level access control on channels ABAC and RBAC in smart-contracts	ABAC and RBAC in smart-contracts	ABAC and RBAC in smart-contracts	Organization level access control
Data Privacy	Multiple private collections each with a different set of members. A transaction can span both private and public DBs.	No	Single private DB per node. A transaction cannot span both private and publicDB.	Data is private to the parties involved in the transaction
Tokens	Defined by smart contracts	Ether	Ether	No
Zero Knowledge Proof	Yes	No	Yes	No
Multi-Tenancy	Supported using channels	No	No	Isolated and multi-tenant by design
Transaction Privacy	Supported only across channels	No	Supported with private DB	All transactions are private and only seen by the authorized participants
Pruning of Blocks and State DB	No	Yes	Yes	Yes
Smart-Contract Language	Golang, Java, NodeJS	Solidity	Solidity	Kotlin, Java

Table 2.1: Comparison of Blockchain Platforms [18]

Ethereum has several drawbacks concerning our problem. Firstly, it is a permissionless blockchain. One would have to protect sensitive information from the rest of the network, for example, by encrypting data, which would constitute a considerable overhead. It also suffers from scalability issues, as the throughput is low, about 200 tps. The current consensus algorithm is proof of work, which is very demanding in terms of processing power. Furthermore, for each transaction, the initiator has to pay a fee (called *gas*). This fee is proportional to the computing power necessary to execute the transaction. *Quorum*[7] is a permissioned blockchain based on the official Go implementation of Ethereum. Quorum uses a raft-based consensus algorithm, which

[7]https://github.com/jpmorganchase/quorum/blob/master/docs/Quorum%20Whitepaper%20v0.2.pdf

allows for better performance than Ethereum. Quorum follows the order-execute paradigm, resulting in low throughputs. Although a better fit than Bitcoin and Ethereum concerning our problem, it still has a low throughput compared to other infrastructures, on the few hundreds of transactions per second. As a digital token, it uses Ether, which does not make sense in our case.

R3 Corda is a blockchain platform that enables management of legal contracts between mutually trusting organizations. The platform makes it possible for a diverse range of applications to interoperate on a single network, with a focus on financial applications. As a downside, it has a low throughput. It has no support for transaction endorsement and fine-grain access control.

MultiChain offers flexibility in designing blockchain applications, which are mainly directed to record assets. A significant limitation of MultiChain (version 1.0) is that it does not implement smart contracts. Despite not having support for smart contracts, Multichain (version 2.0) has smart filters. The blockchain embeds smart filters, which is a piece of code, similar to smart contracts. This code defines rules that regulate the validity of transactions or stream items. At a high level, those are similar to smart contracts. MultiChain 2.0 is now under development and is not suitable for production environments.

Quorum and Multichain use PBFT or similar protocols for atomic broadcast, and they follow the order-execute approach [3]. The order-execute paradigm translates into a sequential execution of transactions, resulting in higher latency and a lower throughput rate, with regard to permissionless solutions.

Hyperledger Fabric (Fabric) is a blockchain framework implementation, backed by the Linux Foundation and IBM [3]. Fabric can accommodate faulty assumptions, as clients might have Byzantine behaviour (provoking byzantine faults). A byzantine fault is a condition in distributed systems, where components of the network might fail, and there is imprecise information on whether a component failed. Fabric's architecture lets users choose an orderer service that implements a consensus algorithm that fits their application.

Fabric uses endorsement policies to define which peers, and how many of them are necessary to vouch for the correctness of a specific smart contracts (also called *chaincode*) execution[8]. Organizations group peers who form independent trust domains, i.e., peers from the same organization trust each other. Several organizations can form a *consortium* is a group of non-orderer organizations that form channels and own peer nodes. Fabric achieves a separation between the execution of smart contracts and the addition of new blocks to the blockchain, deviating from the order-execute paradigm, hence following the *execute-order-validate* paradigm [3].

[8]https://hyperledger-fabric.readthedocs.io/en/release-1.4

Programmers write smart contracts using a mainstream programming language, such as GoLang or Java. The consensus in Fabric is pluggable [52], i.e., using Kafka's Zookepeter or Raft, both crash fault tolerant consensus mechanisms. Plugability allows the developer to adapt the consensus to the requirements of the environment. The consensus algorithms currently implemented to achieve consensus deterministically. This way, forks as in the Bitcoin network are prevented. Fabric can utilize different consensus protocols that do not require a native cryptocurrency, which reduces attack vectors and performance issues due to expensive operations, for instance, the ones required by proof of work. Hyperledger Fabric added a novel way to deal with privacy and confidentiality, through its *channel* architecture and private data feature[9].

Fabric seems suitable for the public administration's use cases. High throughput is needed to handle access control requests, as well as to record the applicational logs. The multi-tenancy offered by Fabric allows us to isolate two environments: access control, and the applicational logs. Additionally, channels provide transaction privacy, while private collections allow for the privacy of sensitive information. In particular, access to smart-contract logic, transaction data, or the current state of the blockchain can be restricted.

2.1.4 Performance Evaluation

There are fundamental concepts associated with the evaluation of a blockchain solution. A typical configuration for evaluating a blockchain's performance comprises the *test harness*, which is the software and hardware necessary to perform the evaluation, and the *system under test*, a blockchain solution formed by several nodes (including the software, hardware, and networks). Key performance evaluation terms, blockchain terms and key metrics as highlighted by the Hyperledger Foundation[10]:

- Load-generating-client: submits transactions to the system under test on behalf of clients.
- Workload: defines how the system under stress is exercised. It should be representative of the production environment.
- Faultloads: are a set of issued faulty transactions to benchmark blockchain solutions, by simulating stressing conditions.

The following list defines common terms used to discuss the blockchain technology, with regard to the evaluation:

[9]https://hyperledger-fabric.readthedocs.io/en/release-1.4/private-data/private-data.html
[10]https://www.hyperledger.org/resources/publications/blockchain-performance-metrics

- Consensus: the fault-tolerant mechanism used to achieve an agreement on a single state of the network.
- Commit: the point where a transaction is committed to the distributed ledger, held by the network nodes.
- Finality: characteristic of blockchain technology. When a transaction is committed, it is final and cannot be reverted.
- Network Size: number of validating nodes participating in the consensus.
- Reads: a read is similar to a query, and behaves as a transaction where there is no state change.
- State: the state is the contents of the blockchain ledger at a specific point in time. Blockchain systems are state machines, where every new block of transactions is a state transition.
- Global State: a state which is shared across all nodes.
- Transaction: state transition that may change data, and follows specific rules.
- Endorsement policy: policy that defines which peers are required to endorse transactions, concerning specific chaincode. Therefore, one can restrict a group of nodes to execute and verify smart contracts.

2.2 Hyperledger Fabric

As referred in Section 2.1.3, Fabric allows for different kinds of participants in the network, which facilitates the *execute-order-validate* paradigm for distributed execution of chaincode [3]. The authors defend that execute-order-validate has advantages in relation to the usual *order-execute* paradigm seen in blockchains like Bitcoin and Ethereum. Endorsement peers (*endorsers*) execute (*endorse*) the smart contracts (*chaincode*) and return blockchain clients the validation output of submitted transactions, which contain the endorsement peers' signatures. This process allows parallel execution and addresses non-deterministic code [52].

Chaincode is a key element in a Fabric network, as it dictates the rules to be followed by member participants. It is run in Docker containers, and is, thereby, isolated from the shared ledger. There are two types of chaincode: application chaincode, that executes the application logic and communicates with the peers using *gRPC messages*, and system chaincode, ran on the *configuration channel*. Chaincode can be deployed dynamically, and it usually is running concurrently on the network. It runs directly on the peers' processes. The configuration channel stores the definition of MSPs, the network address of OSNs, configuration about the consensus, ordering service parameters, and rules on how the channel configuration is tweakable. Chaincode executes transaction proposals against world state data, as the world state provides direct access to the latest value of these keys. Given this, there is no need to traverse the entire transaction log and calculate its values. Each peer contains a ledger component, formed by the *block store*, which stores blocks containing transactions and the *peer transaction manager (PTM).* There is a different ledger for each channel, as channels enforce chaincode and data isolation.

Channels allow participants to establish a communication path between the subset of participants that have permissions to visualize a subset of transactions. For instance, in the same network, there can be a subset of peers that have access to only a certain kind of transactions. In addition to channels, Fabric supports *private data*, which allows a defined subset of organizations on a channel to isolate their data from others. In specific, organizations with permissions can endorse, commit, or query private data, which is logically separated from channel ledger data. In case of a dispute, private data can be shared and shown. For further privacy, hashes of private data go through the orderer, instead of the data itself. It is disseminated peer-to-peer rather than via blocks. When transaction data must be kept confidential from ordering service nodes, it is a solution to use private data collections, rather than channels.

Fabric introduces a hybrid replication model, combining active and passive replication (primary-backup-replication, ported to the untrusted environment). Concerning active replication or state machine replication, the ledger state only reflects the transactions after these are validated and the consensus is reached, concerning their ordering. Passive replication happens as endorsers send the result of the transaction processing to commit nodes [52]. Fabric comprises three main elements around data: *world state*, a versioned key-value store which corresponds to the distributed ledger; the *transaction log*, stores the history of all transactions (PTM); and a NoSQL database, such as CouchDB, stores the world state. It is possible to restrict users' access to view and edit specific fields and only authorising the read-only permissions. CouchDB supports complex data queries, comparatively to LevelDB, against the whole blockchain data, making it a suitable solution when it comes to data analysis and auditing. *LevelDB* is the other built-in option for storing the world state. It is a simple, fast key-value storage library that provides an ordered mapping from string keys to string values.

Although Fabric does not have a built-in cryptocurrency, it is possible to create an underlying token with chaincode, which can represent assets or rights to perform specific actions. Such assets can be exchanged between network participants, through transactions. Participants can hold one or more peer nodes on the network. Fabric defines on its model several kinds of peer nodes:

- *Committing peer.* Each peer maintains the current snapshot of the current state of the ledger, as a store of key-value. Such peers cannot invoke chaincode functions.

- *Endorser peer.* Endorser peers have chaincode installed. When they receive a transaction proposal, they simulate the transaction execution on isolated containers. Based on that simulation, such peers prepare a transaction proposal that is then sent to the orderer peer. The existence of endorser peers avoids sequential execution of transactions by all peers.

- *Orderer peer.* Orderers receive endorsed transactions and assemble them into blocks. After grouping transactions, orderers assure consensus, by propagating such blocks to committing peers, where they are validated and then committed to the shared ledger. Orderer peers record valid and invalid transactions, while other peers only contain valid transactions.

Additionally, Fabric defines *anchor peers* and *leader peers.* Anchor peers serve as an intermediary between peers from its organisation and peers from an external one. Leader peers take the responsibility of distributing the transactions from the orderer to committing peers. To achieve consensus, and given that there is an assumption of partial trust in a Hyperledger Fabric network, Fabric uses a permissioned voting-based scheme, which achieves low-latency. The *endorsement policy* defines the voting-based scheme to be used by peers and, consequently, the weight of each peer regarding the validity of a transaction. The transaction flow, which follows the execute-order-validate paradigm, is depicted in Figure 2.1, is as it follows:

- *Transaction proposal.* A blockchain client, which represents an organisation, creates a transaction proposal, and sends it to endorsement peers, as defined in the endorsement policy. The proposal contains information regarding the identity of the proposer, the transaction payload, a nonce, and a transaction identifier.

- *Execute (endorsement)*: the endorsement consists in the simulation of the transaction. The endorsers produce a write-set, containing the keys and their modified values, and a read-set. The endorsement peers execute the transactions, in an isolated environment. The endorsement is sent as the proposal response and contains the write-set, read-set, the transaction ID, endorser's ID, and the endorser's signature. When the client collects enough endorsements (which need to have the same execution result), it creates the transaction and sends it to the ordering service. The endorsement phase eliminates any eventual non-determinism.

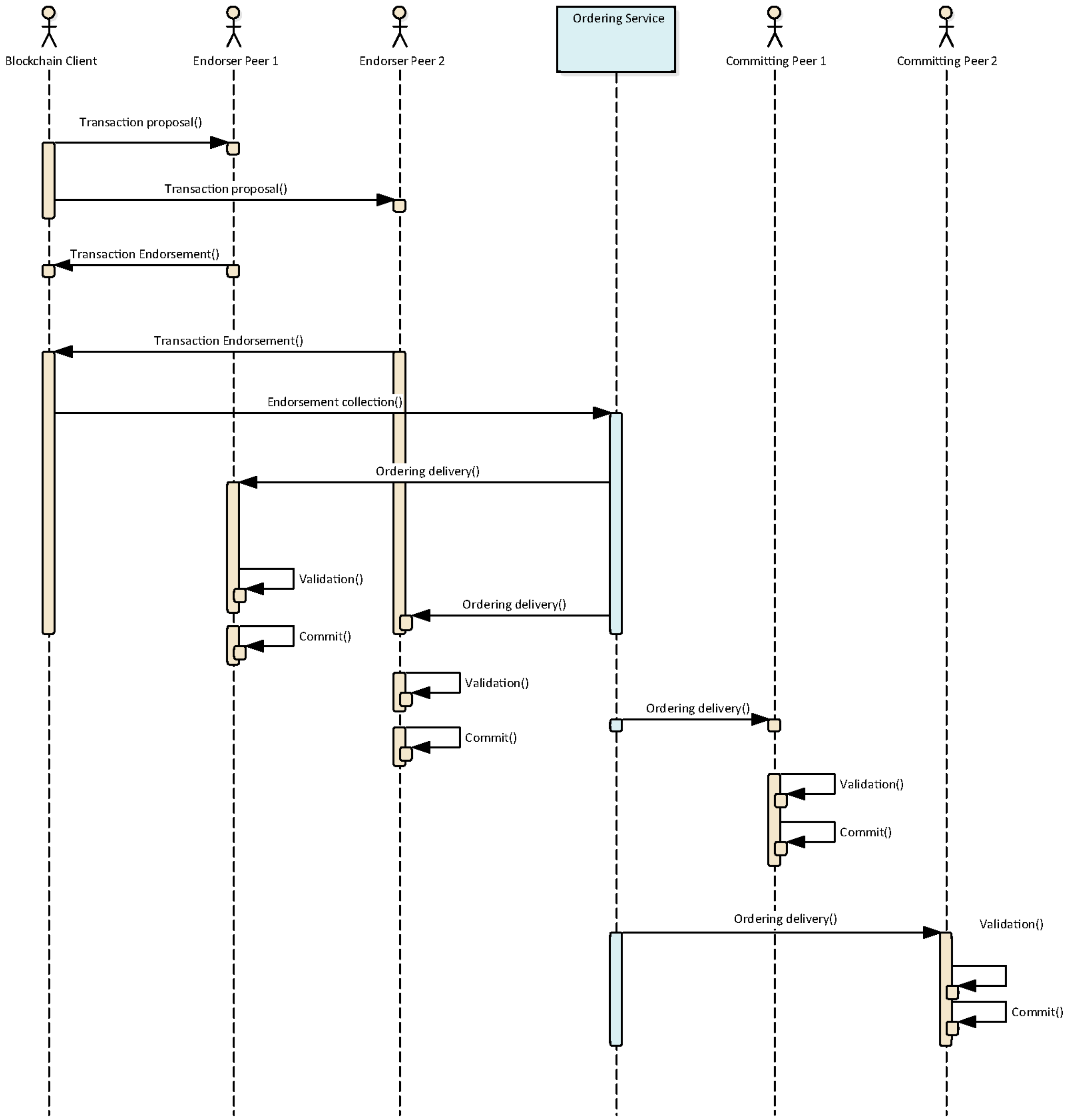


Figure 2.1: Fabric's Transaction Flow [3]

- *Order*: after the endorsement, there is the ordering phase, performed by orderers. The ordering service checks if the blockchain client that submitted the transaction proposal has appropriate permissions (broadcast and receiving permissions), on a given channel. Ordering produces blocks containing endorsed transactions, in an ordered sequence, per channel. The ordering allows the network to achieve consensus. The orderer broadcasts transaction's outputs to all the peers. For a correct ordering, there are some properties that the system must comply to, defined as [52]:

  Definition 1. *Hash chain integrity: For any two blocks B delivered with sequence number s, and B'delivered with s'at correct peers such that s = s', it holds B = B'.*

  Definition 2. *Hash chain integrity: If some correct peer delivers a block B with number s and another correct peer delivers block B'= ([tx 1 , . . . , tx k], h') with number s+1, then it holds h'= H (B), where H (·) denotes the cryptographic hash function.*

  Definition 3. *No skipping: f a correct peer p delivers a block with number s > 0 then for each i = 0, . . . , s - 1, peer p has already delivered a block with number i.*

  Definition 4. *No creation: When a correct peer delivers block B with number s, then for every tx ∈ B some client has already broadcast tx.*

  The ordering step assures all the above properties for each channel. The ordering service broadcasts blocks to the peers that maintain the state of the ledgers, via the ordering service or gossip protocol.

  Fabric comes with both a CFT (Crash Fault Tolerant) and a BFT (Byzantine Fault Tolerant) consensus implementation. The CFT ordering service is based on *Apache Kafka*, which is the reference implementation of the ordering service [26], whereas the BFT ordering service is based on SimpleBFT [11]. In Kafka, the leader orders transactions and sends the results to replicas. Although Kafka is crash fault-tolerant, it is not Byzantine tolerant, as faulty nodes can prevent the system from reaching an agreement. The consensus is not limited to the agreed-upon order of a batch of transactions between peers, but rather, it is also the result of the Execute-Order-Validate process that takes place during a transaction's flow from its proposal to its commitment on each peer's ledger.

  Definition 5. *Validity: If a correct client invokes broadcast(tx), then every correct peer eventually delivers a block B that includes tx, with some sequence number.*

  There is the need to note that the ordering service aims to achieve consensus and therefore prevents the network from forks. Consensus in Fabric encompasses the whole transaction flow, from the transaction proposal to the committing. It happens at the transaction level, where not all nodes need to engage in the consensus mechanism. Channels ensure that messages are delivered in the same logical order to committing peers. The ordering service achieves consensus in a deterministic way. As the ordering is deterministic, and not probabilistic, as in Bitcoin, forks do not occur. Permissioned blockchains which rely on the BFT replication protocols to achieve consensus can only support *f* out of *3f+1* faulty nodes. This assumption may not match the trust model idealised for a specific application. The endorsement policy establishes the trust model, which is decoupled from the consensus mechanism, allowing developers to reason about the trust model independently of the consensus algorithm [52].

- *Validate.* Firstly, each peer validates the received transactions by checking if a transaction follows the correspondent endorsement policy. After that, a read-write conflict check is run against all transactions in the block, sequentially. For each transaction, it compares the versions of the keys in the read-set with those currently on the ledger. In case they do not match, the peers discard the transaction. Finally, the ledger is updated, in which the ledger appends the created block to its head. The ledger appends the results of the validity checks, including the invalid transactions[3].

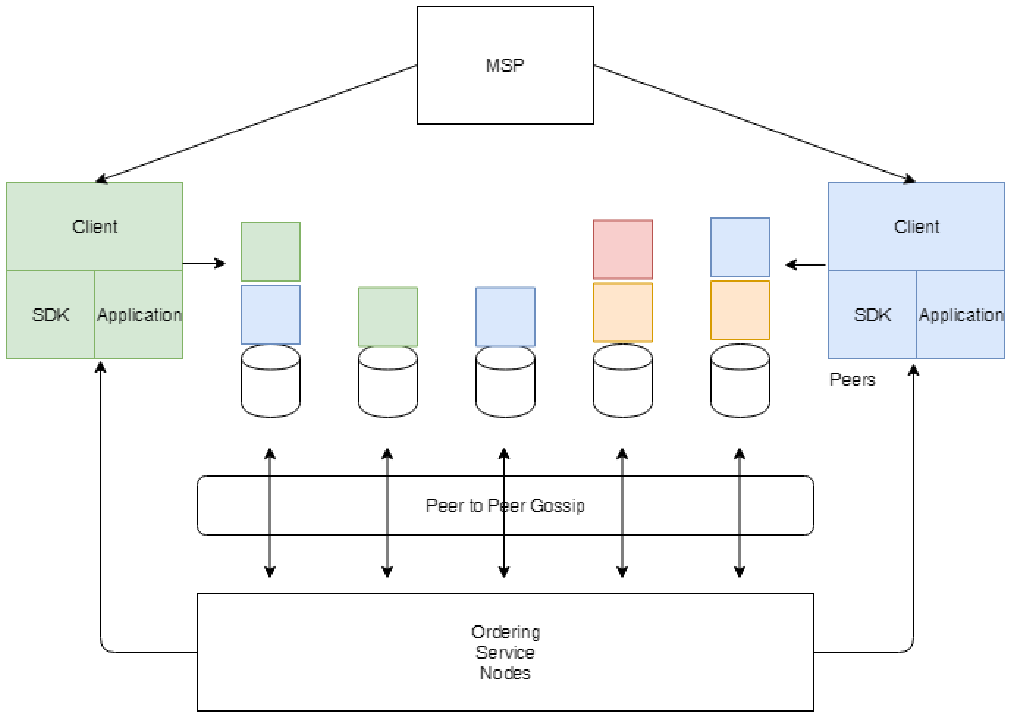


Figure 2.2: Example of a Fabric's network [3]

Fabric has the following architectural components that support the transaction lifecycle:

- *Membership Service Provider*: the goal of this service is to identify participants (authentication) in the network uniquely. Public key infrastructure (*PKI*) is used to generate certificates which are tied to members and organisations. Each organisation issues identities to its members and every peer recognise the members of the organisation. The identification is made through the association of a given peer to a cryptographic entity (i.e., digital certificate). Different identity management protocols can manage the identity of participants, such as LDAP and OpenID.

- *Ordering Service*: it manages multiple channels and comprises orderers and their processes. The batching process occurs in this phase, in which validated transactions are received and grouped into blocks. For each channel, after the transaction validation by the orderer, it broadcasts state updates to the ledger, in an atomic way, following a consensus algorithm. The ordering service can reconfigure channels and restrict broadcasting of transactions if needed.

- *Peer Gossip*: the peer gossip is responsible for broadcasting the results of the ordering phase, as well as *transfer state* for unsynched peers (recently joined, after a downtime, or a peer is slower at validating the blocks before committing). Gossip data dissemination helps to achieve consistency and data integrity across nodes. Since blocks are signed and numbered, after a peer receives them, it can reconstruct the blockchain and verify its integrity. Thus, the gossip protocol manages peer discovery and channel membership and disseminates ledger data across peers on a channel.

2.3 Audit Logs

This section focus on related work regarding generalities about log files, applicational logs, audit logs and blockchain audit logs.

Audit files generated by information systems not only allow administrators to monitor the activity of users, providing insights about their behaviour [51, 9], but are also used by auditors. Auditing processes utilise audit files and address *validation, attribution and evidence.* The validation phase asserts if a subject has performed as expected. The attribution comprises the identifies the subject that is not complying with the system. The evidence phase produces non-repudiable support that holds attribution [54]. In order auditors to be able to reconstruct the sequence of actions a user performed on a system, audit logs need to have integrity guarantees.

Conventional schemes for protecting data include the use of systems with weak security models, which work with the assumption that the logging site cannot be compromised. Attackers have exploited these weak security models [33] and raised awareness about threats to data integrity. Such threats and their realisation, as in log files tampering, is of paramount relevance, as it may condition audits, as the data cannot be trusted. Furthermore, it allows the attacker, insider or not, to delete his or her traces. Not only application logs data integrity is crucial, but also leveraging mechanisms that distribute trust to that integrity.

The checksum is a string used to verify the integrity of information, in a computer system [13]. Data is given as input to a checksum algorithm, and its result is later compared with the digest of the transmitted data. If the checksum is not long enough, the probability of a collision is higher (when another chunk of data yields the same checksum result), resulting in false-positives. Checksums are simple, yet innefective ways of checking data's integrity.

In [4] Bellare et al. propose a method to assure the integrity of data while assuring that data before a certain point cannot be altered, in an undetectable way. A write-only logger creates log entries to provide integrity guarantees. More advanced solutions use a third-party notary service to prevent data-tampering, along with cryptographic hashing, and partial result authentication codes [45]. Such solutions, although efficient, have a single point of failure, where the centralised authority that grants integrity can collude with attackers [28]. Several solutions support forward security but depend at least partially on a third-party. Such solutions, although suitable, does not tackle the need for a trust distribution.

Schneier et al. [44] describe a method for making all entries before logging difficult to the attacker to read, modify or delete undetectably. When audit logs are generated, its audit authentication key is hashed. A log's encryption key is derived from the authentication key, in order to be possible to decrypt and read entries. Each log entry contains an element in a hash chain that enables the verification of the previous log's values. This system requires logs to be generated before the attack and, foremost, the system does not provide a way to stop the attacker from tampering entries.

Snodgrass et al. propose using a third-party notary service to prevent data-tampering on RDBMS audit logs, along with cryptographic hashing, and partial result authentication codes [45, 43]. A check field is associated with each tuple. Each time a tuple is modified, the RDBMS generates a timestamp and associates it with the modified tuple. The tuples are hashed and sent to the notary service. The service returns a unique ID, which is associated with the tuple. In case an attacker changes the data, the ID returned by the notary service will be different, and in that way, administrators can detect attacks. This solution implies that a third-party can be trusted to delegate the integrity checks. This service can collude with an attacker and hence is not for an ecosystem composed of participants with different incentives.

Ma et al. propose an approach that supports forward security and compact aggregation of message authentication codes (MACs), which depend partly on a third-party or with public key infrastructure (PKI) signatures [28]. Log entries are combined sequentially using forward-secure append-only signatures. This solution, although suitable, does not tackle the need for trust distribution.

Ray et al. [40] propose a framework that delegates audit log management to cloud computing services. The secure cloud-based log management service is based on cryptography, that ensures confidentiality, integrity while performing operations on audit logs. There are solutions in which a middleman is used to verify the integrity of data, and then send the report to the dedicated parts [5]. Bharathi et al. propose this approach, introducing a third-party auditor (TPA). As a drawback, this service requires that all stakeholders trust the third party. A distributed ledger can help alleviate such problems derived from trust assumptions [62]. Combining existing, settled methods with new approaches might lead to a suitable solution. In particular, distributed ledger technologies can be leveraged to close the gap that exists concerning trust distribution.

Blockchain has emerged as a technology that promises advances in not only the integrity of the data but also its availability and non-repudiability, all desirable characteristics to applicational logs. Using such technology for such a goal might seem appealing, but one has to consider its limitations - namely the low throughput, high latency, and storage problems inherent to many blockchain infrastructures [21].

Some authors are exploring the the blockchain technology to create tamper-proof audit logs. Sutton and Samavi propose a mechanism for log integrity and authenticity verification that auditors can utilise [46], using Bitcoin. The proposed system logs proof integrity proofs on the blockchain. There are three main entities on the system proposed by, the logger, audit log and auditor. The logger creates the logs and stores them on the audit log. Auditors query the audit log, which contains the logs created by the logger.

As events need to be non-repudiable, the logger signs the transactions, it submits to prove accountability.

Furthermore, the logs need to have integrity assurances, so integrity proof digests of the log events (i.e., cryptographic hash) are generated and stored on the *integrity preserver* (i.e., blockchain). Those records can be retrieved to participate in the process of compliance checking, with log integrity verification. The solution has low throughput, as the Bitcoin blockchain can only hold 3-7 transactions per second.

Cucurull et al. present a similar approach to Ma et al. since it combines MACs and DAs (data auditors). The Bitcoin blockchain is used, as it provides distributed immutability [16].

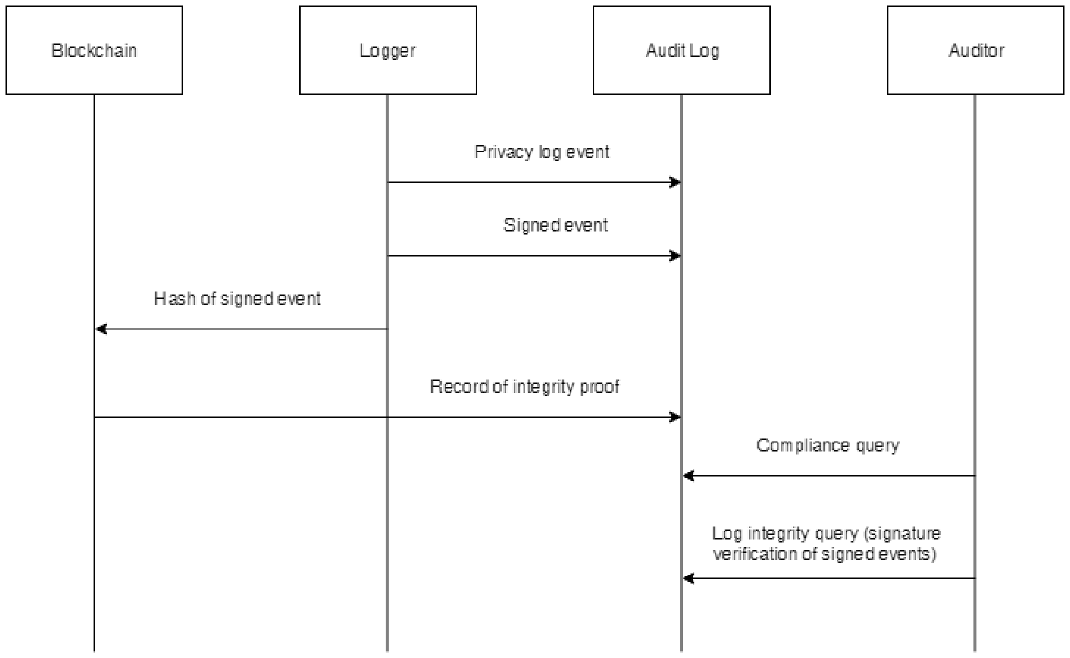


Figure 2.3: Privacy audit log generation process [46]

By using the blockchain, the logs are chained in the same order as they were generated. In their proposal, the logger has a pair of signing keys and signs each log entry. Log entries can be regular ones or checkpoints. Each log entry *LogInfo*$_i$ is chained with the previous one, using a MAC cryptographic function. The logger concatenates the log entry with the hash of the previous log entry, creating a proof of integrity h_{i-1}. This process constitutes a chain of hashes. A different random session key K_J can be used to prevent modification, deletion or addition of intermediary entries.

$$L_i = (LogInfo_i, h_i), \text{where } h_i = HMAC(K_j, (h_{i-1}|LogInfo_i)) \tag{2.1}$$

Log entries constitute the base of the integrity mechanism proposed. Checkpoint entries are used to verify the authenticity and non-repudiation of the last block (j) of entries. In each checkpoint, the MAC session key used to chain the last block is disclosed so that other parties can verify its authenticity. A new session key is generated P_{enc} from time to time. A digital signature of the entry is also created with the signing key (S_{sig}). This way, log tampering attempts are detected. One can assure the authenticity of the integrity proofs by cryptographically signing the issued transactions.

$$Chk_j = L_i = (LogInfo_i h_i, K_{j-1}, E(P_{enc}, K_j), Sig_j, h_{i-1}, h_i) \tag{2.2}$$

$$h_i = HMAC(K_b, (h_{i-1}|K_{j-1}|LogInfo_i)) \tag{2.3}$$

$$Sig_j = S(S_{sig}, (h_{i-1}|K_{j-1}|E(P_{enc}, K_j)|h_i|LogInfo_i)) \tag{2.4}$$

This mechanism, shown in equations 2.2, 2.3, and 2.4, can detect the location of modification attempts, which is an advantage compared to digitally signed logs. Distributed ledger technologies can help replicate secure logs while keeping the properties described by the author. However, this solution requires an infrastructure of servers (i.e., an infrastructure comprising blockchain nodes), a mechanism to save the logs (i.e., blockchain client which communicates with the logger) and a consensus protocol (i.e., enforcement policy). The blockchain then receives integrity proofs, which are hashes of the checkpoints (Chk_j). The integrity proofs can be later compared with the actual log files, stored elsewhere, in order to check for manipulations. The validation phase consists in retrieving the checkpoint hashes from the blockchain, recompute them using the log files and compare their match in content and order.

Anderson and Smith propose AuditChain, a blockchain-based full-stack system to secure, standardise and simplify health record audit logs [2]. In this work, identities are issued through Hyperledger Fabric certificate authority and linked to a local user profile. Those identities are linked to an organisation within the blockchain, having read permissions. Participants can invoke chaincode query audit log entries. Organisations, such as a public hospital, can invoke chaincode to create audit log entries. On top of the blockchain, the authors built a user interface, to enable users to inspect the logs. A limitation of this system is a lack of a process that links the authentication credentials from electronic health records systems that are used by health professionals within an organisation to the cryptographic identities hold on the blockchain network.

In [38], Pourmajidi and Miranskyy propose Logchain, a blockchain-assisted log storage system. Logchain tries to decentralise trust on stakeholders that use a third party service. In particular, Logchain goes towards granting the user protection against the tamper-motivation cloud providers might have (in case of dispute, e.g. accruing from improper service provisioning). Cloud participants have access to logs but, unlike JusticeChain, fine-grain permissions related to audit are lacking. Privacy and performance problems with LogChain go *pari passu*, as the system entails the usage of a permissionless public blockchain. Blockaudit [1] takes a holistic approach to the problem of protecting audit logs from adversaries, by protecting both physical access attack and the remote vulnerability attack, using a custom blockchain infrastructure.

Author	**IC**	**IA**	**SPF**	**D**	**DP**	**FGP**
Cohen, 1987 [13]	✓	x	✓	x	-	-
Bellare et al., 1997 [4]	✓	x	✓	x	-	-
Schneier et al., 1998 [43]	✓	✓	✓	x	-	-
Snodgrass et al., 2004 [45]	✓	✓	✓	x	-	-
Ma and Tsudik, 2009 [28]	✓	✓	✓	x	-	-
Ray et al., 2013 [40]	✓	✓	x	x	-	-
Bharathi and Rajashree, 2014 [5]	✓	✓	x	✓	-	-
Cucurull and Puiggal, 2016 [16]	✓	✓	x	✓	x	x
Sutton and Samavi, 2017 [46]	✓	✓	x	✓	x	x
Pourmajidi and Miranskyy, 2018 [38]	✓	✓	x	✓	x	x
Anderson and Smith, 2018 [2]	✓	✓	x	✓	✓	x
Ahmad & Saad & Mohaisen, 2019 [1]	✓	✓	x	✓	✓	x

Table 2.2: Features from contributions to audit logs.

Blockaudit leverages the PBFT consensus mechanism to protect audit logs. Blockaudit focuses on securing logs and not in distributing trust towards that security, something that JusticeChain aims to do. Furthermore, JusticeChain tackles privacy issues, by allowing only auditors to access the audit logs they are allowed to audit.

Table 2.2 summarizes the contributions of different authors to audit logs.*IC* stands for integrity checking, and *IA* stands for integrity assurance. *SPF* represents if the solution has a single point of failure, and *D* is used if the solution can decentralize information for different stakeholders. *DP* means data privacy and *FGP* represents if there is the possibility to have fine-grain permissions over accessing the stored data, validated by the blockchain.

2.4 Access Control

Access control is the selective access restriction to a set of resources. For that, one can introduce the concepts of *authentication* and *authorization.* In Authentication, in which the requester provides proof of its identity. After authentication, the authorization phase takes place, where the access control system grants or denies access to a particular resource. Access control systems grant or deny an authenticated subject the execution of a particular action concerning a resource, as outlined in the corresponding access control policies. Access control policies are sets of rules which define when a specific user should have access to a particular resource. At a high level, access control systems contain three main elements: subjects, resources, and policies.

According to Zuquete [61], the defence against non-authorized activities has its difficulty increased when the attackers belong to the organization that owns the information system. That is due to the abuse of privileges that the users have, in order to obtain what they need. Access control systems have an essential role in preventing, discourage or avoiding data-tampering, such as editing logs or sensitive information, by providing control access to information, managing the users, to blocking unauthorized accesses to resources, such as information systems, files, and applications. We should take into account that certain operations, such as accessing sensitive information, should be recorded in order to leverage audits [11].

2.4.1 Access control on the computer

The most common access control methods in the computer occur at the operating system level. It starts with the *identification* of a subject and the *authentication* [42]. When users authenticate, they are providing the system proof of their identities. Authentication in the operating system level can be done through passwords.

As referred, *Access control mechanisms* aim to provide a sound system that allows or denies a subject to perform a specific action to a specific object, as outlined in the *access control policy.* The access policies also define access control rules. Ultimately, we want to guarantee the confidentiality and integrity of resources. The access control mechanisms that are going to be described follow the *reference monitor concept* [35]. The reference monitor sets requirements for a validation mechanism that enforces an access control policy on subjects that request access to a particular object, on a system. They are complete mediators, tamper-proof, not by-passable and verifiable [24]. Some of the most common access control models are:

- *Mandatory access control (MAC)*[22]: the system administrator specifies which subjects can access specific data objects, by assigning them security levels. In this model, all data objects have a security label attached, the classification and category. The category limits access across security levels.

- *Discretionary access control (DAC)*[22]: in this model, the owner of the object specifies who can access the object and its permissions. A popular implementation of this model is access control lists.

- *Role Based Access Control (RBAC)*[19]: this model incorporates characteristics from MAC and DAC. The system binds privileges with groups of subjects with similar responsibilities, called roles. Using roles allow the administration to manage a high number of subjects, by assigning each one with a role. Users are not able to delegate or assign permissions to a role they are allowed to perform to other users.

[11]https://www.sans.org/media/score/checklists/ISO-17799-2005.pdf

- *Attribute-based Access Control (ABAC)*[12]: Attributes are name-value pairs that are associated with different identities. ABAC grants permission for a user to access a resource based on his or her attributes, attributes associated with the application granting access and environment variables. ABAC has a set of rules that are combined, following a certain combining algorithm. Those rules have to be satisfied in order to grant access. ABAC is indicated for fine-grain access controls, as it checks an arbitrarily complex set of attributes against an arbitrarily complex set of rules. This model allows expressing rich, complex access control policies. *XACML* is an implementation of the ABAC model, defined by the OASIS consortium. It is an XML based language to describe attribute-based access control policies. It also defines an architecture and a processing model to address access requests, according to certain control access policies.

- *Identity-based Access Control (IBAC) [25]*: access is granted on a user-by-user basis. IBAC allows fine-grain access control over who is allowed to use specific services and can access resources.

- *Entity-Based Access Control (EBAC)* [8]: supports the comparison of attribute values, and relationship traversing. This access control method leads to expressive access control policies.

There are some implementations of these models, such as [37]:

- *Access Control Directory*: the objects (e.g., files, programs, network resources) have an owner, that sets the access rights for other subjects, concerning a particular object. The owner can revoke access to them.

- *Access Control List*: permissions over object are mapped into a list. It holds all subjects and their permissions for a given object.

- *Capability*: a form of access control that has to be managed by the user. Usually, it is a token that is unforgeable and gives its owner a set of rights over individual objects. A key point in capabilities is propagation. A subject can send a copy of its capability to another subject if it has permissions to do so. Frequently, capabilities are backed up by more general structures, such as access control lists or access control matrix.

2.4.2 Authentication, authorization, and accounting

Network authentication, authorization, and accounting (AAA) is a control access methodology. Authentication aims to provide an answer to the question, “Who is the client?” while authoriza-

[12]https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cd-03-en.html

tion provides an answer to, “What is the client allowed to do?”. The *Accounting*, on its turn, answers to the question, ”What did the client do?”. These fundamental security building blocks enable networks to have effective and dynamic security. According to XACML, the components of an AAA system are:

- *Client*: a device that requests access to a resource, on behalf of a user.
- *Policy Enforcement Point (PEP)*: a device that requires a set of conditions to the client to access the resource, by intercepting access requests from the subject.
- *Policy Information Point (PIP)*: holds information about the environment, the clients and their permissions.
- *Policy Decision Point (PDP)*: decides to allow or to deny the client access to the resource.
- *Policy Retrieval Point (PRP)*: used by the PDP to retrieve access control policies, for evaluation against a request.
- *Policy Administration Point (PAP)*: is the component responsible for managing access control policies.
- *Accounting*: involves tracking usage during the lifetime of the connection, such as accesses that the PDP grants or denies to the client.

A widely-used AAA protocols that mediate the communication between the PEP and PDP is the *Remote Authentication Dial-In User Service (RADIUS)* protocol [7]. Figure 2.4 represents the logical components of AAA.

The client is the device that needs authentication to access the network, on behalf of a user. The authentication credentials, submitted by the client to the PDP (via PEP) are normally one of the following types:

- *Shared-key*: A username is associated with a password.
- *One-time password*: a token is used to generate an access code valid for one access only. This process has to be synced with a token server (PIP).
- *Digital certificate*: emitted and signed by an authority. This certificate contains information about the entity it refers. It contains a private-public key pair, used for encrypting and decrypting the same message.
- *Biometric credential*: based on what the client is, such as a fingerprint.

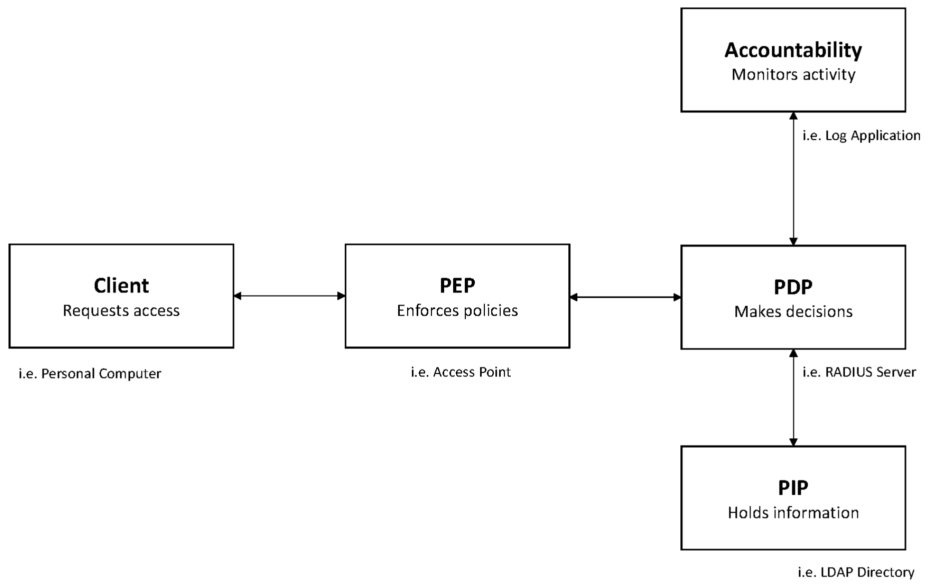


Figure 2.4: AAA Logical Components [7]

When a client requests access to a network, the PEP sends a challenge. The PEP sends the request information to the PDP. One of the most used protocols to make the connection between the PEP and PDP is the *RADIUS protocol* [7]. The PDP queries the PIP for information about the client requiring access. The communication can be achieved via a protocol called *Lightweight Directory Access Protocol (LDAP).* The structure that saves users' information is also called LDAP. Another commonly used PDP-PIP communication protocol is *Active Directory (AD)*, based on LDAP, RADIUS, and *Kerberos.*

The PIP, which serves as a credential directory, holds an *X.500* directory (also called LDAP directory), accessed by either the LDAP protocol or RADIUS. The LDAP directory is held in a *Directory Service Agent (DSA)* and is accessible by LDAP and RADIUS clients. The most common DSAs are *OpenLDAP* and *Microsoft Active Directory (MAD, or simply AD).* As long as there are several authentication servers, it allows scalability and redundancy on the authentication. The PIP then validates or not the client's credentials sent by the PDP, and returns this information to the latter. PIP can also send additional information about the client to the PDP. The granularity of this authorization is as good as the sophistication of the PDP.

The PDP now compares the information it has about the client against its configured policies. It may also take into account the environment in which the decision-making process takes place (i.e., time of the day). The PDP announces the authorization result to the PEP and writes that result in the accounting system. The PEP applies the authorization granted by PDP, notifying the client. If the authorization is successful, the client can now access the resource through the PEP. Those categories are logical containers of functionality and, therefore, do not need necessarily to be on different physical devices. For instance, one can concentrate PDP and PIP in one logical container, as well as joining PDP and Accounting.

The underlying idea to include the *blockchain* technology into an access control system is using it to serve as the PDP, PIP, PAP and accountability system. A comprehensive introduction to the blockchain technology is done, to us to understand if it can be a viable solution.

2.5 Blockchain Access Control

This section reviews the state of the art blockchain-based access control methodologies. New distributed software architectures leverage the blockchain, as it is a technology that enables to decentralize data and code execution, across untrusted participants. Updating or removing blocks on a blockchain is not possible - this is desirable when we want to achieve integrity, for instance, an access control log. Since all blocks are connected, traceability is also achieved, making it possible to reconstruct the sequence of operations a user performed in an information system.

Zyskind et al. conceptualize the blockchain technology as an access control moderator, complemented by an off-blockchain storage solution [62]. Blockchain clients representing users that provides its data to a service provider are the owners of their data. Based on that premise, this solution is meant to empower users, so they have the information about which data is collected about them by third parties and how their data is used. For achieving that goal, each data owner can issue transactions, used to change the set of permissions granted to a service or entity. Each transaction is recorded on the blockchain, allowing for auditability and traceability.

Laurent et al. propose a fine-grain blockchain-based access control solution, complementing the approach referenced before [34]. In this solution, there are several entities: the blockchain infrastructure, the *data owner (DO)*, *data retriever (DR)*, and the *data storage provider (DSP)*. The DSP governs host application services that hold DO's data. The DO is the entity responsible for attributing access rights for its resources, stored on a DSP. The DO defines those accesses by whitelisting DRs, the entities that require access, with a specific access control list in a smart contract, stored on the blockchain. The smart contracts containing the whitelists are final, but hold variables that are referring to the entities that have permission. Therefore, it is possible to a DO to dynamically assign or revoke permissions to a certain DR. The DRs are blockchain clients that can access content stored in remote servers depending on the permissions recorded on the blockchain by DOs. The idea is the blockchain to participate in the authentication of each DR in respect to DSP, before granting access. Blockchain properties leverage identification, authentication, auditability, and data integrity, as it prevents the control list to be altered. Figure 2.5 illustrates the white list creation process.

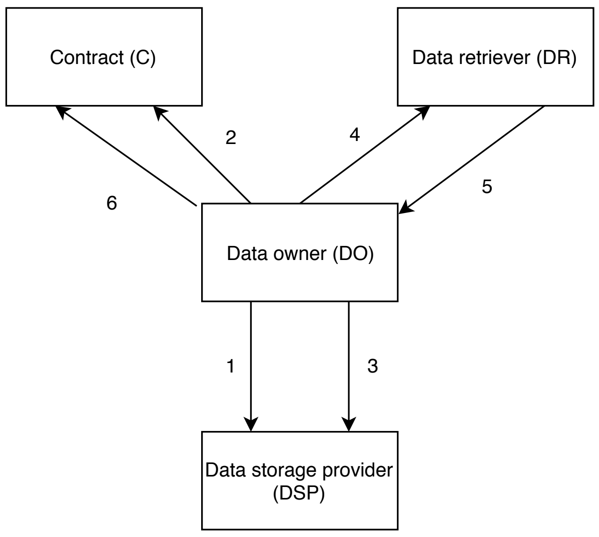


Figure 2.5: Whitelist creation process [34]

1) Firstly, the DO contacts the DSP, expressing which resources are there to protect. 2) After that, a smart contract is created on the blockchain, containing a whitelist with the authorized DRs and fine-grain access control rules (access control lists). The DO can alter the whitelist, and it is empty when it is instantiated. 3) The DO shares the address of this contract with the DSP, along with the files to protect. 4) After that, it contacts each DR and sends the contract address. 5) The DO collects DR addresses that are meant to be granted permission to specific resources and updates the whitelist with their addresses. 6) The whitelist is updated. The blockchain records all changes. In practice, we can see which DO whitelist each DR, for each resource associated with a specific DSP, with specific permissions. This process allows for fine-grain access control. When it comes to a DR to access data, it sends an access request to the corresponding DSP. The DSP sends the key to the DR) and starts listening to the blockchain for transactions signed with that key. The DR sends a transaction to the smart contract containing the key, resulting in leaving a trace on the blockchain. When the DSP detects a transaction with a given nonce, it matches the issuers' address with the ones on the whitelist specified by the contract. In case it matches, the DR is identified and granted access (or not) to the required resource. In this solution, the blockchain identifies DRs and DOs, by their address.

This solution misses some key points that are essential to address a suitable solution to the problem proposed. This solution assumes that a data owner-centric model, in which for each outsourced data resource the DO creates a smart contract. Besides that, every authorization has to be declared explicitly (judge X can access resource Y), which can add considerable overhead to the implementation process. Furthermore, this solution is implemented with Ethereum, a public blockchain that comports overheads, such as the payment of gas. Zhang et al. propose a solution directed to Internet of Things (IoT) blockchain-based access control [59]. The authors introduce the concepts of *Judge Contract* (JC), *Register Contract* (RC) and *Access Control Contracts* (ACC). Access control contracts store access control policies for a subject-object pair. In this system, both the JC and RC are essential pieces when it comes to achieving distributed and reliable access control. The JC receives misbehaviour reports and applies penalties according to them. The RC stores the misbehaviour information from the JC and manages misbehaviour judging method. Moreover, it stores information such as name, subject, object, and smart contract, for access control. In this solution, all entities controlled are peers in an IoT network. Nonetheless, the concepts of JC and RC are useful in the context of this study, in particular, to leverage effective auditing techniques. Maesa et al. propose a blockchain access control solution based on an XML-based Language which expresses attribute-based access control policies [29]. Policies define access rights on *resource R*, defined by the *resource owner P*. After that, policies might be updated or revoked by the resource owner. Each policy identifies the affected subject unequivocally, and the conditions (attributes, environment) for the access to be granted. Although the authors implement this solution with an ABAC model in mind, it can be extended to cover other access control models. XACML defines a reference architecture for the access control framework that is going to be introduced. A PAP is responsible for translating XACML policies to smart contracts (smart policies). These smart policies, written in the *Solidity* programming language, are stored on the policy repository, the blockchain, with a transaction. In this solution, the policies and the rights are visible on the blockchain, and this connector allows for access rights to be transferred amongst subjects. This process allows for distributed auditability, as any participant can know at any time the policy associated with a set of resources and subjects. Subjects might transfer permissions through a transaction. Attributes from subjects, resources and the environment are stored in smart contracts. Smart contracts are collected by the PIP when a request is made.

In this solution, permission transfers are possible and are always a subset of the current permissions over an object, which means that the previous conditions cannot be violated. The new conditions are more restrictive than the original ones. The subject trying to access a resource authenticates with the PEP. The subject might be required to sign a challenge with the private key corresponding to the identity it used to get the access rights in the smart contract. The information is passed to the *Context Handler (CH)*, that interacts with other components. The CH sends the request to the PAP. The latter retrieves the smart contracts related to the request (first policy and its updates) and returns it to the CH. The CH queries relevant attributes from the PIP and redirects them to the PDP. The PDP evaluates the decision and returns it to the CH. The CH forwards the decision to PEP, which enforces it, resulting in the request being executed or not. Transparency is achieved by saving the smart contracts that encode access control policies on the blockchain and tracking updates to the policies.

Although this solution might seem to address the problem given, there are some significant drawbacks. Subjects have the right to freely exchange action rights between themselves without interacting with the policy issuer (can be the resource owner). The XACML format is massive, even compressed. A lighter object notation, like JSON, could be a better fit. Additionally, non-explicit access control rights transfer can be dangerous if not well controlled, as that implies that the policy issuer does not know in advance who will be accessing resources from a given policy. This model constitutes an issue because users should not be able to exchange rights without permission, on sensitive scenarios. There are also privacy implications with this approach, as every node on the network knows which policy is associated with a specific user. Privacy concerns can also arise, as all participants can access the defined access control policies.

Maesa et al. complement the solution mentioned above by introducing some novelties [31]. The general idea is the same, to implement an access control service on top of the Ethereum blockchain. The blockchain is used to store smart contracts that represent access control policies and to perform the decision process. Such smart contracts are called *smart policies (SPs).* Thus, SPs are responsible for the policy evaluation process, embedding a PDP for a specific access control policy. Each time an access request needs to be evaluated to make an access decision, they are executed, by the blockchain, in a distributed way. The decision is made based on information concerning the users. For this purpose, the concept of *Attribute Manager (AM)* is introduced. AMs are the components that manage the attributes of the entities involved in the process, such as subjects, resources, and environmental context. AMs can update and retrieve its values and are created by an entity, the *attribute provider (AP).*

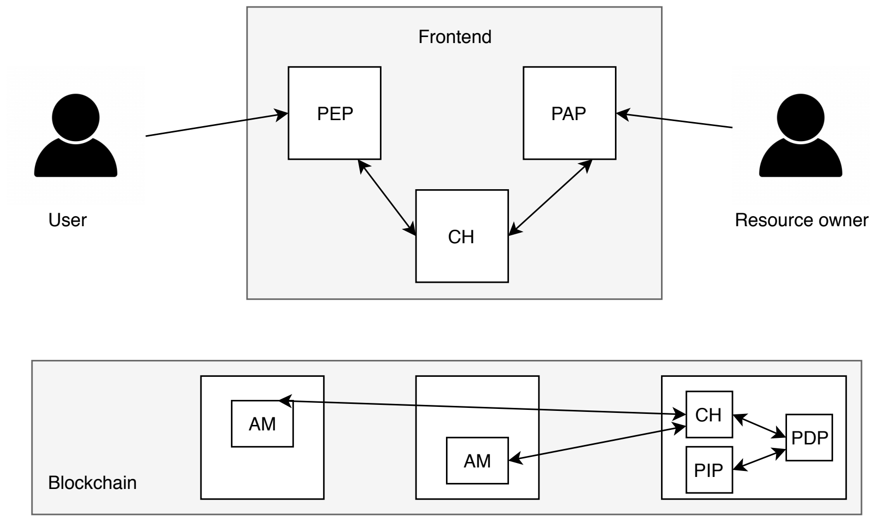


Figure 2.6: Blockchain-based access control service architecture [32]

Furthermore, the part of the smart policy that invokes the smart AMs is considered a PIP, because it is the interface that allows interacting with the AM, retrieving its attributes. In a high-level view, the decision-making (PDP) is achieved through the SP execution. Smart function calls on the SP realize the PIP function, as they retrieve necessary attributes. SPs can be revoked and, therefore, not callable anymore, allowing for dynamic access control policy management. The SP is still accessible and seen in the blockchain, as the ledger is append-only.

This method allows for an access control mechanism that relies on the blockchain technology. The concepts of PEP and CH are the same as previously referred. The CH is divided between a *blockchain component* (CH_B) and *off-chain component* (CH_O). The CH_O is responsible for interacting with the blockchain (with the different SPs), on behalf of the PEP and PAP. The SPs perform the role of the CH_B, as communication in the blockchain is achieved through smart contract calls and event firing. For simplifying, we refer the CH_O as the CH.

Regarding the workflow, the first step is to create XACML policies. Then an administrator uploads policies to the PAP, which translates them into SPs. SPs are deployed to the blockchain by the CH. The PAP keeps a reference to the deployed policy. When the PEP intercepts an XACML access request, it redirects it to the CH. It then parses the request, obtaining the resource ID and other attribute values necessary for a decision. It then queries the PAP for retrieving the address of the smart policy paired with that resource. The CH sends a transaction containing a call to the evaluate method of the corresponding SP. The SP retrieves the necessary information from the AMs, and the PDP evaluates the request. After the SP evaluates the request against the policy, it returns the decision to the CH. The CH redirects the decision to the PEP that on its turn enforces it.

In this solution, both the PAP administrator and the subjects need to have an Ethereum wallet, which makes the setting cumbersome. Moreover, the PAP would have to pay for each access control policy deployed, whereas the subjects, would have to pay gas for each access request.

Uchibeke et al. developed a *Blockchain Role-Based Access Control Business Network* (*BR-BAC BN*), based on Hyperledger Fabric, which is used as an access control layer to cloud computing environments and big data services [50]. The *BRBAC BN* is used to verify that an entity has access to a dataset represented by an ID. The ID can represent an asset, a query that retrieves data or a function that pulls data from an external repository. This solution aims for peer-to-peer authorization grants, i.e., a user can grant or revoke permissions required by another user concerning a specific asset. The data model of this solution comprises a *person* participant, a *data* asset and a *request*, *grant*, *revoke*, *verify* and *view* transactions. Participants

Author	**D**	**FGP**	**S**	**PB**	**MS**
Zyskind et al., 2015 [62]	✓	✓	x	x	x
Laurent et al., 2018 [34]	✓	✓	x	x	x
Zhang et al., 2019 [59]	✓	✓	x	x	x
Uchibeke et al., 2018 [50]	✓	✓	x	✓	x
Maesa et al., 2019 [32]	✓	✓	✓	x	x

Table 2.3: Features from contributions to blockchain access control.

who join the network can issue one of the transactions defined in the business network. The asset owner can grant or revoke the request to an asset. The ledger is updated when there is an accepted authorization. The user that views the ledger can query the blockchain and have access to the access control decision. This solution does not tackle the use case of a decentralized access control framework which distributes trust and responsibility between organizations, which are stakeholders of an information system.

Table 2.3 summarizes the contributions of different authors to blockchain-based access control. *D* stands for decentralization, and *FGP* stands for fine-grain permissions. *S* represents the scalability of the system (where there is no need to explicitly map a user to have the rights to act on an object, on a specific context). *PB* represents the usage of a private blockchain. *MS* represents support to multi-stakeholder scenarios, in which there might be several administrators of the network.

2.6 IGFEJ

This section presents insights about processes at IGFEJ, the systems they administrate, and the functional and non-functional requirements for JusticeChain. IGFEJ administrates several relevant systems for the Portuguese Republic, in which a blockchain-based solution could improve their logs' resiliency:

- *CITIUS – Magistrados Judiciais, CITIUS – Ministério Público and CITIUS – Entrega de Peças Processuais e Documentos por Via Electrónica.* Those are more commonly just referred as Citius. Citius serves the necessities of judicial magistrates and aims to expedite judicial courts processes.

- *Sistema de Informação dos Tribunais Administrativos e Fiscais (SITAF).* It aims to streamline processes from fiscal and administrative courts.

- Other systems less relevant to the context of our problem, such Sistema Informativo de Custas Judiciais (SICJ).

All the systems above depend on IGFEJ. Figure 2.7 represents one of the primary use cases of the Citius system [13].

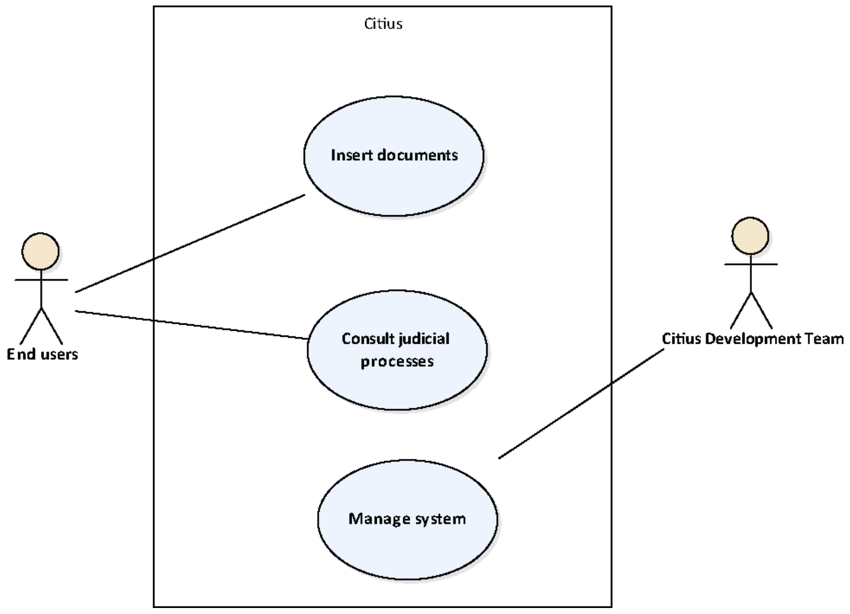


Figure 2.7: Use cases of Citius addressed in this thesis.

Citius' stakeholders are IGFEJ, *Inspeção-Geral dos Serviços de Justiça (IGSJ)*, *Conselho dos Oficiais de Justiça (COJ)*, *Procuradoria Geral da República (PGR)*, and *Conselho Superior da Magistratura (CSM).* The stakeholders with respect to SITAF are IGFEJ, PGR, COJ, *Conselho Superior dos Tribunais Administrativos e Fiscais (CSTAF)* and IGSJ. IGFEJ develops and maintains Citius, while COJ, PGR, and CSM represent Citius' end users: judges (magistrados); court clerk (funcionários de justiça); and probation officers (oficiais de justiça. IGSJ and CSTAF are entities which are likely to be involved in the auditing of Citius. Citius has approximately ten thousand users. Citius is managed by IGFEJ, by a development and operations' team. For the sake of simplicity, we refer to the teams that develop, operate, and administrate the system as *Citius development team.*

The authentication at Citius is made through the LDAP protocol with a Microsoft Active Directory (AD) server stored on the cloud. The AD is used to authenticate users and manage the domain's policies. Authorization concerning posterior requests is done locally, with a resource to a component embedded on the application. The application which implements the interface to access the protected resource embeds the PEP. Regarding authentication, judges have two-factor authentication, a username-password pair, and a smart card. Probation officers have a username-password keypair. Both groups are vulnerable to social engineering, as peers can easily pick their credentials and act on behalf of them. The sequence diagram that represents a

[13]https://www.citius.mj.pt/portal/article.aspx?ArticleId=0

login is depicted in Figure 2.8.

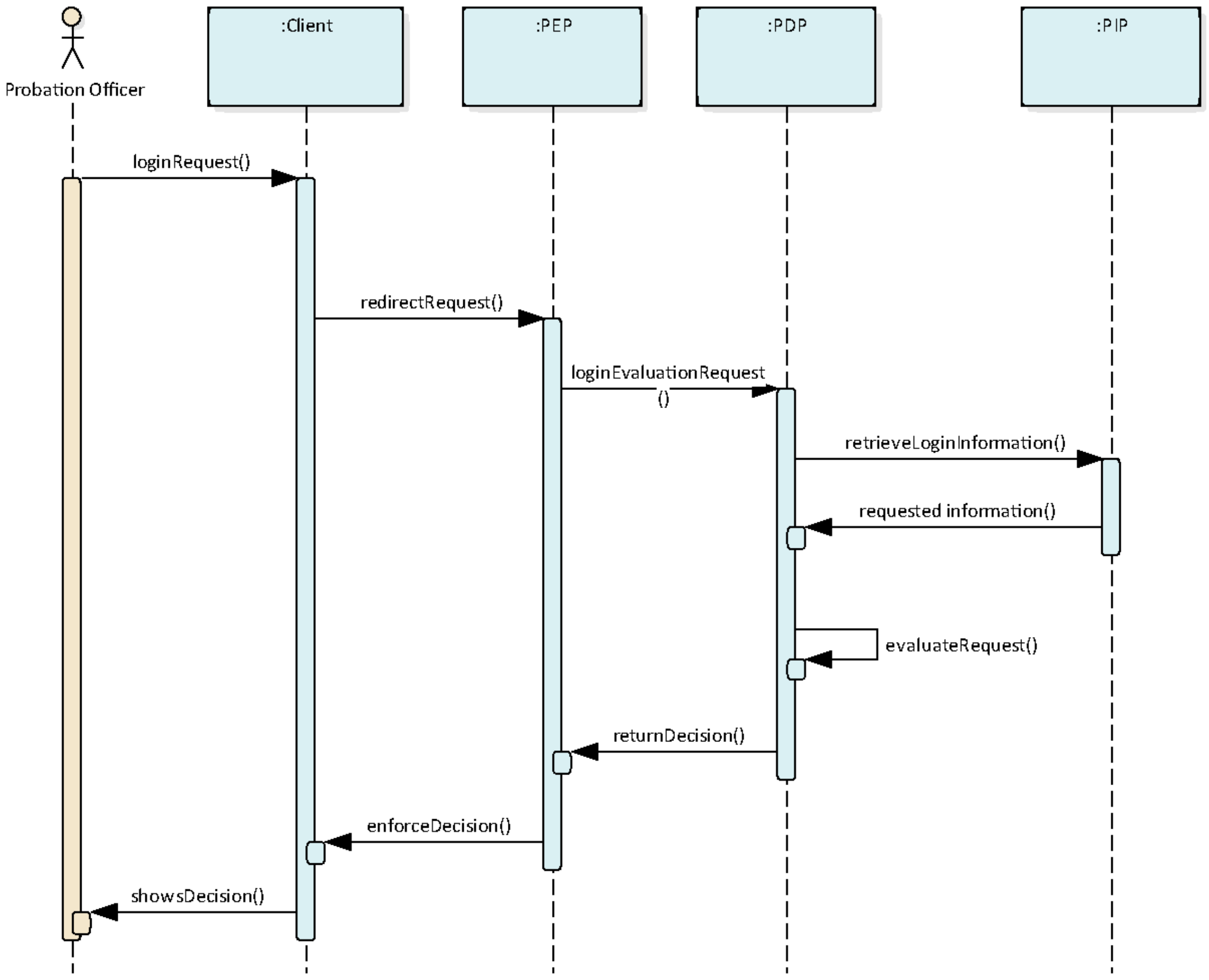


Figure 2.8: Citius' login request flow

For the sake of simplicity, we omit the user interface from this flow. The user inserts the credentials on the client component. The component sends it to the PEP, which redirects them to the PDP. The PDP contacts the PIP and retrieves the necessary attributes to proceed to the authentication. In this case, it compares the username-password provided with the one stored on the AD. The PDP decides and communicates that decision to the PEP, which redirects it to the client. The login is recorded on the AD logs. The flow for an access request (authorization) is similar. The difference lies in the PIP. In Citius, as the authorization is made locally, the PIP is a local database. The accountability is achieved through writing the authorizations provided on text files (via database triggers).These text files hold information about the user's activity, including access requests towards different resources, user activity and user's personal information. Such data is stored on the same local database. Twenty-

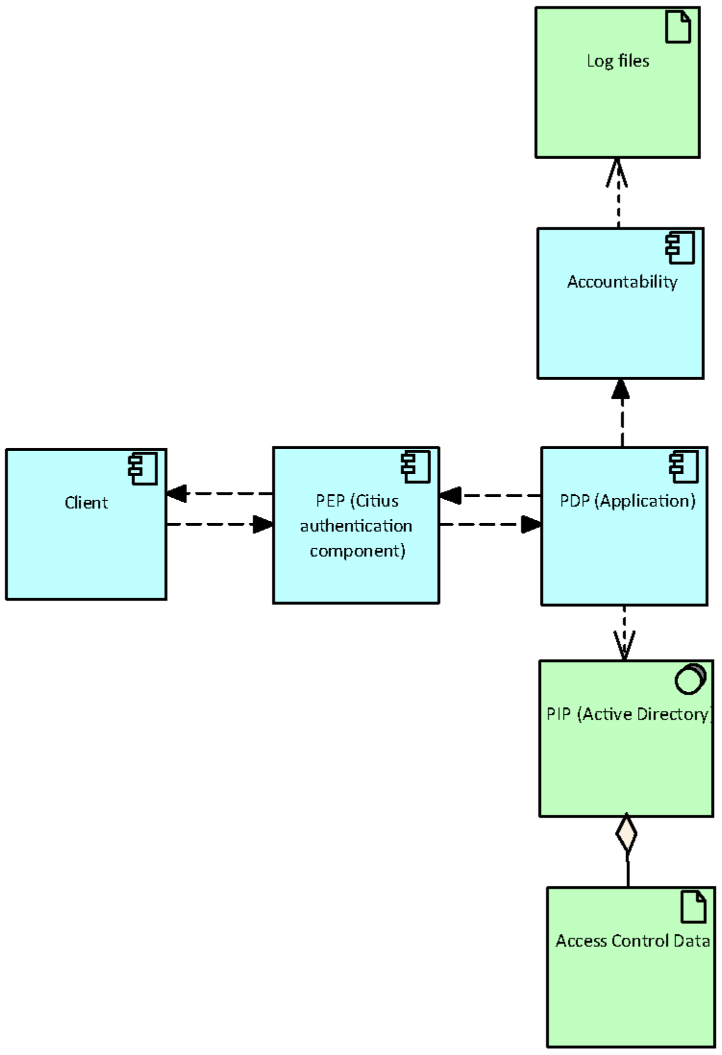


Figure 2.9: AAA architecture of Citius

three databases serve the system, being each one a PIP, one for each legal division of the courts (*comarca*). Each *comarca* has its logs, kept locally, without availability or redundancy guarantees. Those constitutes a point of attack, where attackers can several attack vectors to compromise the integrity of data and, thus, the audits performed over that data. Section 3.7 provides details on the implementation of a possible solution to this problem.

Furthermore, the administration of accesses is kept to local teams. As a consequence, we obtain weak access control mechanisms concerning the authorization. Similarly to the authentication, the authorization should be uniformized, allowing for scalability and decentralized control. Audit logs created by IGFEJ's systems are fragmented across several different applications, and have different formats, making cumbersome crossing logs in case of several systems being attacked. Chapter 4 provides an additional contribution that secures logs and at the same time tackles the authorization issues at Citius. Citius' architecture is depicted on Figure 2.9. In this chapter, we presented central concepts on blockchain technology, in particular on permissionless and permissioned blockchains. Next, we introduced Hyperledger Fabric, as the chosen infrastructure to solve the proposed problem. We reviewed the state of the art blockchain-based access control. Finally, we introduced IGFEJ, the organisation that provides this thesis with a real case. Next, we propose a solution that enables Citius' stakeholders and end-users to benefit from a more secure system, which facilitates (through the tamper-proof logs) the identification of a culprit, in a breach scenario. IGFEJ and the stakeholders of Citius benefit from the transparency offered by a blockchain solution, allowing the government to distribute trust, responsibility and streamline the audit process.

Chapter 3

JusticeChain's Design and Implementation

To explore and evaluate the ways that permissioned blockchain technology can improve audit logs creation and security, we propose *JusticeChain*, a full-stack application, composed by the blockchain client components and the blockchain components. JusticeChain protects audit log data by loading it on a permissioned blockchain, which is composed of several peer nodes, belonging to different organisations. In particular, a *consortium* between the different stakeholders is formed to audit and manage audit logs.

Hyperledger Fabric provides the underlying permissioned blockchain functionality, which provides *data integrity*, *stream integrity* (correct order of data), *forward integrity* (the attacker can erase log entries, but cannot otherwise modify an existing entry or create new entries with logging time preceding his break-in time) [4] and *non-repudiation* (an entity cannot dispute its authorship toward the creation of a log). Chaincode manages audit log creation and access by enforcing both legal requirements and professional recommendations for audit log records. As chaincode handles creation and retrieval of audit log data, it also imposes a standard structure on this data, achieving interoperability for participating provider organisations. As the ledger is append-only, audit log data cannot be altered once committed to the ledger. Moreover, chaincode allow us to implement automatized auditing techniques if needed. The solution's user interface presents the information stored on the ledger in a readable and useful format to users. Users have their local authentication linked to the authentication credentials from Fabric's on-chain cryptographic identity management service.

JusticeChain allows to evaluate the central question of this thesis: how can the properties of a private, permissioned blockchain, such as Hyperledger Fabric, improve the resiliency of logs at the Portuguese government? In this chapter, we start by introducing the system requirements.

Secondly, we present the implementation and design decisions of JusticeChain.

3.1 Requirements

Systems administrated by IGFEJ (e.g., Citius, SITAF) are expected to maintain audit logs that ensure trust in the application data and application state and provide tamper-resistant evidence of the modification of such state. The non-functional requirements of JusticeChain are:

1. *Tamper-Resistant.* The system should ensure that audit logs are tamper-resistant. In case a node suffers a remote attack, or even an internal access attack, JusticeChain can stop the attack, as other nodes hold the global ledger. We also expect JusticeChain to ensure tamper-resistant audit logs in the presence of byzantine nodes (faulty or dishonest nodes).

2. *Adaptability.* As the reality of IGFEJ is evolving, there is a need for an adaptable solution. The solution should be flexible enough to be adapted to similar use cases (e.g., protecting applicational logs other from Citius).

3. *Low latency and high throughput.* As audit logs are generated in real-time and persisted in the system database, JusticeChain needs to achieve low latency while maintaining the capacity of processing a large number of transactions per second (high throughput).

4. *Auditability.* The system should make available a set of audit logs to auditors, in function of their credentials and the context of the audit. Access to audit logs should also be recorded and thus, auditable.

5. *Availability*: the system should be functioning in proper conditions, with minimal downtime.

6. *Privacy*: the solution should assure privacy concerning the contents to be protected. It should allow for the possibility of choosing which network nodes can access applicational logs. Following the principle of the least privilege, only auditors related to a particular system can access its logs.

7. *Scalability*: The solution should offer the possibility of adding more nodes to the network, while not compromising significantly the performance (i.e., adding one more node should not reduce the throughput significantly). The solution should support at least two nodes.

8. *Testability*: it should be possible to test the solution in a non-production environment (i.e., emulating higher workloads).

9. *General Data Protection Regulation (GDPR) Compliance*: the system should anonymise personal data before the blockchain records it.

JusticeChain must operate in a distributed way with smart contracts running on multiple hosts, in order to distribute trust. Although IGFEJ should not be able to insert, modify or delete data, it should have the privileges to govern the network. Concerning functional requirements, the solution should be able to manage logs securely. Those include:

- Store applicational logs
- Retrieve applicational logs
- Manage the audit authorizations
- Verify logs' integrity
- Verify auditors' compliance

The Citius system should be able to securely store logs through our solution, and provide data integrity guarantees. Auditors, both from IGFEJ and an external independent organisation should be able to retrieve applicational logs, only when needed. External auditors should also be able to verify that no auditors have access to audit data when its not needed. Moreover, auditors should be able to verify logs' integrity. The use cases that the solution should tackle are expressed in Figure 3.1.

3.2 Threats to Data Integrity

The more unlikely an attacker is to change the data, the stronger its integrity. Gaetani et al. propose that data integrity is a quantitative concept, rather than a simple binary property [21]. In order to quantify how well our solution addresses threats, we evaluate the vulnerabilities with the current systems administrated at IGFEJ (i.e., Citius), and we present its threat model. We propose the security model objectives in order to enhance the system's resiliency to different attacks. Based on the threat model and security model, one can aim to a fine-grain evaluation of the solution, concerning data integrity requirements.

JusticeChain improves the log resilience in the two following ways: it records applicational logs from information systems with different stakeholders and secures them on the blockchain; it decentralises the storage of such logs, resulting in higher redundancy and availability. Therefore, it allows authorised auditors to analyse the usage of the system with integrity guarantees. The auditing process is decentralised and transparent for all participants on the network, due to

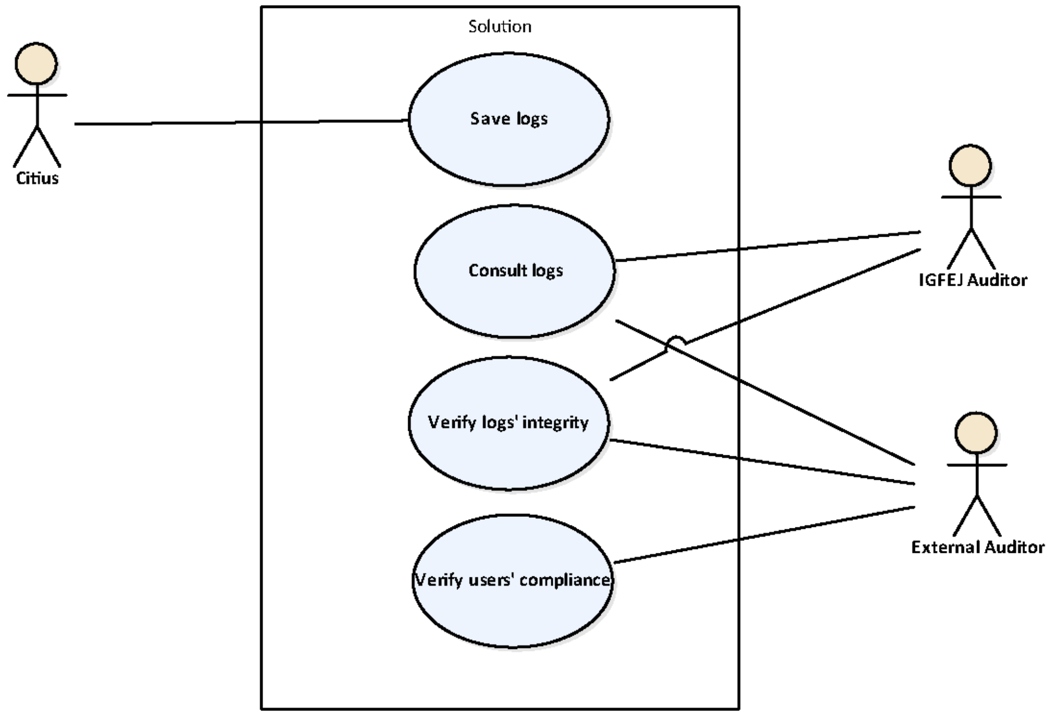


Figure 3.1: Use cases concerning applicational logs

smart contracts that inspect logs. This section presents the threats to which logs are exposed. The threat model will focus on three factors: i) internal access attacks ii) remote attacks and iii) Hyperledger Fabric blockchain attacks.

We assume that there are two types of adversaries. A third-party adversary, *Adversary* A_t, can access the trusted computing base and remotely penetrate the system, by exploiting security flaws or by hijacking root user credentials. Stemming from this, A_t might tamper audit data to compromise auditing and forensic procedures or tamper system data to cause damage to the organisation. On the other hand, an adversary from within the organisation, *Adversary* A_o can be an employee with root privileges. Such a person can access the databases at will and can be motivated to tamper data for personal gains. Such adversaries can tamper data to hide their traces or tamper data to aid third-parties illegally.

There can be two main types of attacks to data integrity: the *physical access attack* and the *remote vulnerability attack* [1]. In the physical access attack, the adversary *Adversary* A_t or *Adversary* A_o have access to the critical system components. The attacker generates transactions to change the current values of the objects held in a database. As objects are being updated, the database generates an audit log, tracking the changes made by the attacker. The attacker then focuses on deleting the evidence by deleting the generated audit logs or changing its content.

The attacker can manipulate the auditing process by modifying the history maintained by the audit log.

As a consequence, the obfuscation of illegal activities and impersonating someone's actions can occur. In remote vulnerability attacks, the attacker exploits default vulnerabilities on systems, such as software malfunctions and security vulnerabilities (e.g., SQL-injection). Although such attacks are common [6], corporate organisations have their systems and databases secure against conventional attacks. We assume that IGFEJ is protected against remote vulnerability attacks, and focus on the most damaging type, the physical access attacks. Five threats compose the threat model:

Threat 1. *Log tampering from an external element: An attacker violates the integrity of the applicational logs, by editing them.*

Threat 1 (T1) is an external adversary gaining access to the logs. Having access to the logs, attackers can edit the logs at their will, i.e., deleting arbitrary entries. This attack has a higher severity on information systems which do not replicate logs as the attacker can permanently delete information.

Threat 2. *Database tampering from an internal adversary: An attacker from one of the stakeholders violates the integrity of the applicational logs, by editing them directly.*

T2 is similar to T1, with a higher degree of severity. If adversaries are insiders, they have direct access to the protected resource. An internal adversary might know peculiar ways to obfuscate such activities.

Threat 3. *Log tampering by the system administrator: The administrator of the system, with the highest permissions, violates the integrity of the applicational logs, by editing them. There is the possibility of obfuscating activity traces by deleting evidence on other systems (e.g., RADIUS logs).*

Threat 3 (T3) is similar to T2, with higher severity. Administrators have access to all resources and can, theoretically, delete all traces. The usage of a blockchain can prevent the threats as mentioned earlier. Nonetheless, such a method has its threats to data integrity.

Threat 4. *A participant edits logs that are protected by the blockchain.*

T4 is not severe, if using a permissioned, private blockchain, like Hyperledger Fabric (Fabric), because transactions have to be endorsed before committed. Even if any participant on the network tries to modify the applicational logs maliciously on its ledger, they cannot change other peers' ledger state, as honest endorsers would not endorse such transactions.

Threat 5. *The majority of participants conspire and modify the logs.*

T5 evolves from T4, where the minority of nodes try to tamper logs. Members from the network can collude to alter the logs' integrity, to trick an external auditor. If all participants on a network want to change its state, it is theoretically possible. The participants can follow the protocol and rewrite the world state, submitting new transactions from the point on in which they want to change the state.

Threat 6. *Real decentralization is at stake with one administrator*

T6 is related to the fact that the distribution of trust seems incompatible if there is only one entity responsible for the systems. Although one could initially consider only one administrating organisation, IGFEJ, this does not achieve real decentralisation. It is crucial to assure that every organisation has an active role in the process, being IGFEJ the main but not the only administrator of the network. Although IGFEJ could be the only administrator, this does not grant that organisation the ability to corrupt the state of the digital ledgers, assuming that participants do not collude and that the system is configured correctly, as discussed in Section 3.7. Including nodes from other organisations lowers collusion risk, and thus increases the robustness of the solution.

The fact that Fabric allows the creation of a permissioned blockchain (updates to the configurations of the system and deployments of smart contracts are recorded), where often participants are vetted, allows reducing the risk of collusion. This process enables the straightforward identification of the subject that initiated specific actions, being a demotivating factor for adversaries. Although Fabric is not tamper-resistant, in practice, under a strict endorsement policy, it can provide a high level of trust. Performance can be enhanced and threats minimised by tuning the definition of the structure, including but not limited to the endorsement policies, number of peers, nature of peers, peers' permissions and external backup guarantees [48].

3.3 Preliminaries on JusticeChain

So far, we have discussed the advantages of audit logs, their vulnerabilities and the existing solutions that address them. Furthermore, we have studied the suitability of blockchain technology to enhance audit logs. We presented a threat model to outline adversarial conditions and possible actions. In this section, we leverage this knowledge to meet the requirements of a blockchain-based audit log solution that tackles the needs of Portuguese justice.

The use case addressed in this thesis presents four characteristics: *i)* the participants are willing to cooperate but have limited trust in each other, *ii)* the trust and responsibility of managing the logs belong to all stakeholders, *iii)* one or more organisations should be able to administer the network, in accordance with the governance model, and *iv)* different participants (end users) have different access rights to data. Regarding the first and second characteristic, consensus mechanisms ensure that no single entity controls the blockchain. As far as the third point is concerned, Fabric allows decentralised management of the ecosystem, allowing for multiple administrators. If IGFEJ is the only administrator, the representation of IGFEJ could be divided into several nodes that represent different teams with different interests and different administrations within IGFEJ. Concerning the fourth characteristic, specific permissioned blockchains, such as Hyperledger Fabric allow the delegation of a different level of control to specific participants [3]

Besides the mentioned characteristics, there is the possibility of not all entities to be faithfully collaborating, by executing the consensus protocol, which can lead to undesired states of the network [45]. This risk is much higher in public blockchains, leading to attacks as the 51% Attack [57]. Such risks are diminished by utilising a permissioned, private blockchain as Fabric. Moreover, Fabric utilises an endorsement policy to validate transactions, meaning that tolerance towards Byzantine nodes is a function of consensus defined on the endorsement policy to be applied. The stricter the endorsement policy, the more resilient the network is. This fact leads to the decoupling between the consensus algorithm and security model.

A blockchain participant represents a stakeholder which participates in the blockchain. We assume that all participants have limited trust (i.e. have different political or economic incentives) in each other and participate in the same channel. Member participants control peer nodes which maintain the ledger and may endorse transactions. This research assumes that there is at least one member participant that owns an information system, and has a Logger that records the logs of the first on the blockchain. We assume that, in case of an attack, one or more nodes, composing the minority, are colluding to change applicational logs. Additionally, communication within the blockchain ecosystem is done using secure channels (i.e., using SSL/TLS). We assume the machines running nodes are physically separated, so one organisation could not subvert other's peer nodes.There are three actors (participants) who take part in the ecosystem:

- *Logger*. Loggers receive log entries from an oracle connected to an information system. Database transactions are capptured by an *object-relational mapper*, and sent to the oracle. Such transactions are generated in Citius' databases when an user performs access control requests. After retrieval, a pre-processing stage might occur, to normalise log entries. After that, the logger sends HTTPS requests to a *Representational State Transfer (REST)*

Application Programming Interface (API) to store the audit log entry. Loggers act as blockchain clients and use oracles.

- *Auditor*. Auditors audit secured applicational logs on behalf of an organisation. Auditors have a set of permissions, allowing for fine-grain permissions for auditing purposes.

- *Network Administrator* (admin). An admin manages the blockchain configurations. Responsible for creating and managing participants within the blockchain, such as Loggers and Auditors. Admins should belong to the organisation that maintains the system that generates logs. In case there is more than one admin, a quorum should make decisions.

The authentication of Loggers (PU_L,PK_L), Network Admins (PU_N,PK_N) and Auditors (PU_A,PK_A) rely on public-key cryptography (*PKI*) [17] and public-private key pairs (PU, PK), leveraging Fabric's certificate authority (CA). We assume that only the owner of a key pair knows the private key (PK), whereas all participants know the public key (PU).

3.4 Data Model

JusticeChain has a data model that addresses the business concerns about managing applicational logs. In particular, applicational logs can only be created and accessed by identified, authorized parties. As JusticeChain needs different participants with different responsibilities, a set of participants were defined, along with a permission set. Participants defined in Section 3.3 interact with the data in the following ways:

- *Network Administrator (Admin)*. An Admin has the following permissions: can see the whole ledger, the whole transaction history and update participants. Network admins cannot create, update or delete applicational logs.

- *Auditor*. An Auditor member (external from the organizations directly involved in the network) participates in the network, monitoring the flow. If the adversaries try to change their attributes, the auditor node will know, as there will be state inconsistencies across nodes. Permissions: Auditors cannot create, update or delete applicational logs. Auditors can only see part of the ledger - logs associated with the auditor's organization. Auditors can only see the transaction history that affects the network configuration and the logs associated with their organization.

- *Logger*. A Logger can be active (records logs) or inactive (does not record logs). Loggers have a *level*, which allow to tune the level of detail of the logs. Levels follow the standard

RFC 5424 [23]. For instance, the INFO level designates informational messages that expose the flow of the application at coarse-grained level. The lower the logging level, the higher the granularity. Permissions: Loggers cannot update or delete applicational logs. Loggers can create logs associated with one information system. For instance, a Logger associated with System A can create an asset type *Log-A*, but cannot create an asset of type *Log-B*. Loggers can see their transactions and the transaction history that refers to the state of the ledger.

Furthermore, the ecosystem has *Assets*. An asset represents anything of value, including physical and non-physical assets. The asset we aim to protect is the log.

- *Log Entry*: A log entry (or simply log) has an unique identifier, making it possible to unambiguously identify each log entry. A log entry has a *blockchain timestamp*, corresponding to the timestamp of the transaction on the blockchain, and a *entry creation timestamp*. The entry creation timestamp refers to when the entry log is generated in an external system from the blockchain, e.g., Citius. Moreover, the log entry contains an associated Logger and case-specific attributes. More log types can be defined, depending on the amount of information systems participate in the network. Logs have been implemented taking into account the *Conselho de Ministros*'s technical recommendations[1] for the security of personal data on public adminstration's information systems.

The UML class diagram for JusticeChain's data model is present on Figure 3.2. The class *Log Entry*, or simply *Log*, allows the system to scale to different business needs. *CitiusLog* and *CitiusLog*$_{30}$ contain 20 and 30 attributes, respectively. Administrators have permissions to manage Auditors and Loggers. Auditors can access Logs, while Loggers can create them. Loggers create Logs recurring to transactions, that issue the execution of chaincode that creates logs.

Transactions are requests to the blockchain to execute specific chaincode. Transactions can affect participants and assets. Chaincode written in *Javascript*[2] creates logs that are recorded on the immutable ledger, via a transaction issued by a blockchain client. The types of transactions are:

- *Create Log*: This transaction creates a Log type. An event *New log* is triggered. Only Loggers can issue this kind of transaction. More transactions can be defined, to allow for saving other types of logs, allowing for adaptability and easier maintenance.

[1]https://dre.pt/home/-/dre/114937034/details/maximized
[2]https://developer.mozilla.org/pt-PT/docs/Web/JavaScript

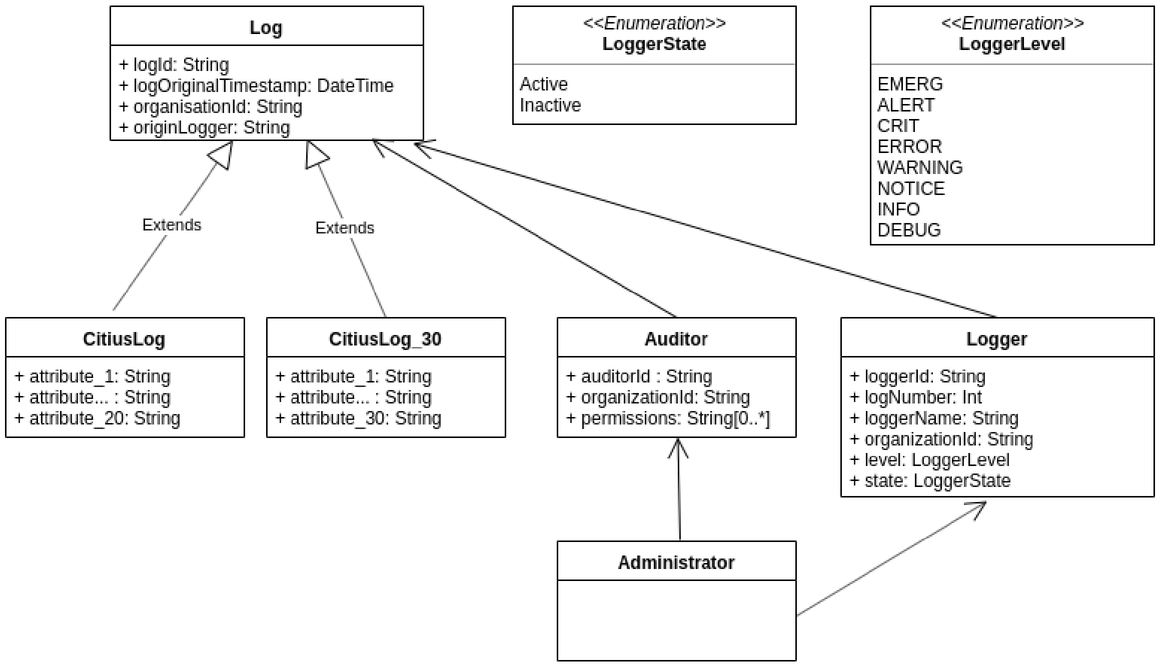


Figure 3.2: JusticeChain's UML class diagram

- *Update Logger level*: this transaction updates the level associated with a given Logger. Only Admins can issue this transaction.
- *Update Logger state*: this transaction updates the state associated with a given Logger. Only Admins can issue this transaction.
- *Change permissions*: this transaction updates an Auditor's permissions to audit information systems. Only Admins can issue this transaction.

The issuing of transactions can lead to chaincode execution which can produce *Events*. Events are notifications emitted when a specific condition is met, which applications may listen and react to. JusticeChain emits an event when: *i)* a new applicational log is recorded on the ledger, *ii)* the state or level of a Logger is changed, and *iii)* an Auditor accesses applicational logs. The types of events are:

- *New log*: this event is emitted when a *Log* is created.
- *Logger level changed*: Fabric emits this event when the level from a logger is changed.
- *Logger state changed*: Fabric emits this event when the state from a logger is changed.

3.5 Technologies

This section introduces the decisions about the tecnologies used to implement JusticeChain. One should not discard alternatives to a blockchain, including but not limited to traditional and distributed database.

Traditional databases, like MySQL are, due to its nature, mutable when a predefined set of users can insert or update data. Administrative roles can alter the contents of the stored information, independently of their decentralisation. The traceability, verifiability, transparency, security, privacy and notarization of transactions are normally more fragile in distributed databases than of a permissioned, private blockchain [10].

A distributed database, such as Amazon DynamoDB supports multiple non-trusting writers and are generally cheaper. Nonetheless, distributed databases cannot directly achieve the distribution of trust, i.e., blockchains provide improved transparency and auditability across the involved parties. Furthermore, one may need transaction interaction and a set of rules to be running across nodes to implement the desired functionality, something that distributed databases can not grant per se. Our use case presents three vicissitudes: (i) its participants have limited trust in each other, but limited trust, (ii) the trust and responsibility of managing the logs should belong to every organisation and (iii) IGFEJ should be able to administer the network. In a multi-organization scenario, entities are not willing to delegate full control of applicational logs, as they carry essential information, and its manipulations have consequences (such as hiding actions). Consequently, each entity should be able to monitor, at least partially, the shared applicational logs. These requirements match the characteristics of a blockchain. Regarding the first and second characteristic, consensus mechanisms ensure that no single entity controls the blockchain. Concerning the third characteristic, specific permissioned blockchains, such as Hyperledger Fabric allow the delegation of a different level of control to specific participants. These features make a blockchain solution suitable to address the given problem.

Given a system that can verify the log's integrity, one can cross this information with other information provided by IGFEJ, raising the probabilities of identification of the intruder, in case of manipulation. Additionally, network nodes are distributed. If some of the nodes are offline, when those nodes rejoin the network, they will automatically re-sync all the blocks. Let us consider the trade-offs between a permissionless and a permissioned blockchain.

In permissionless blockchains, there are privacy concerns due to the information leakage that can occur. More specifically, applicational logs contain sensitive information, such as user behaviour, that should be protected (i.e., hidden) from not only the public but also from within the system stakeholders. Even though it is possible to limit access to sensitive information, there is the possibility of information leakage. This concern is enforced and gains a more significant magnitude when taking into account the general data regulation protection. This system is used mostly for providing an access control solution and leveraging efficient and trustable auditing. It only concerns the administration of the organisms that are responsible for such systems. Public permissionless blockchains, such as Ethereum and Bitcoin, were built with a specific design based on a cryptocurrency. Those blockchains are maintained by miners, which receive economic incentives paid by peers who send transactions, at the expense of considerable computational power.

Using such infrastructure would require IGFEJ to pay transaction fees, which is not adequate for our scenario. Additionally, such infrastructures have limitations, including the sequential execution of smart contracts (after consensus), non-differentiation between smart contract execution nodes and validator nodes, and hard-coded consensus protocols [52]. Public blockchains are not suitable to address the proposed problem.

Taking into account the requirements listed on Section 3.1, a permissioned blockchain is suitable to implement our solution. At a permissioned blockchain, the risk of any participant intentionally introducing malicious code through a smart contract is lower than in permissionless blockchains. First, the authentication component identifies all the participants. Second, the blockchain records the transactions initiated by blockchain clients. Transactions follow an endorsement policy that is established by the consortium for the network and relevant transaction type. The blockchain records all transactions, even the invalid ones (those that are not added to the ledger). This fact allows for the competent authorities to identify not only offenders, but potential offenders, and in terms of the governance model, they can handle the incident.

To implement a blockchain solution with Hyperledger Fabric, one can use *Hyperledger Composer* [3]. Hyperledger Composer (or simply Composer) is an abstraction that aims to facilitate the process of building blockchain applications and business networks. It has a set of tools and scripts which simplify the creation of Hyperledger Fabric networks. The modelling language allows the definition of super-types, leveraging inheritance's properties.

Composer has a built-in modelling language, *Hyperledger Composer Modeling Language*, defined by a namespace where all the necessary resource declarations take place. Composer can generate a REST API server, enabling data integration capabilities automatically. It provides an interface adaptable to the supported languages to interact with Hyperledger Fabric blockchain. It generates a *business network archive (BNA)*[4] for the network. Composer broadly covers these components:

- Business Network Archive (BNA)
- Composer Playground

[3]https://hyperledger.github.io/composer/latest/
[4]https://hyperledger.github.io/composer/latest/introduction/introduction.html

- Composer REST Server

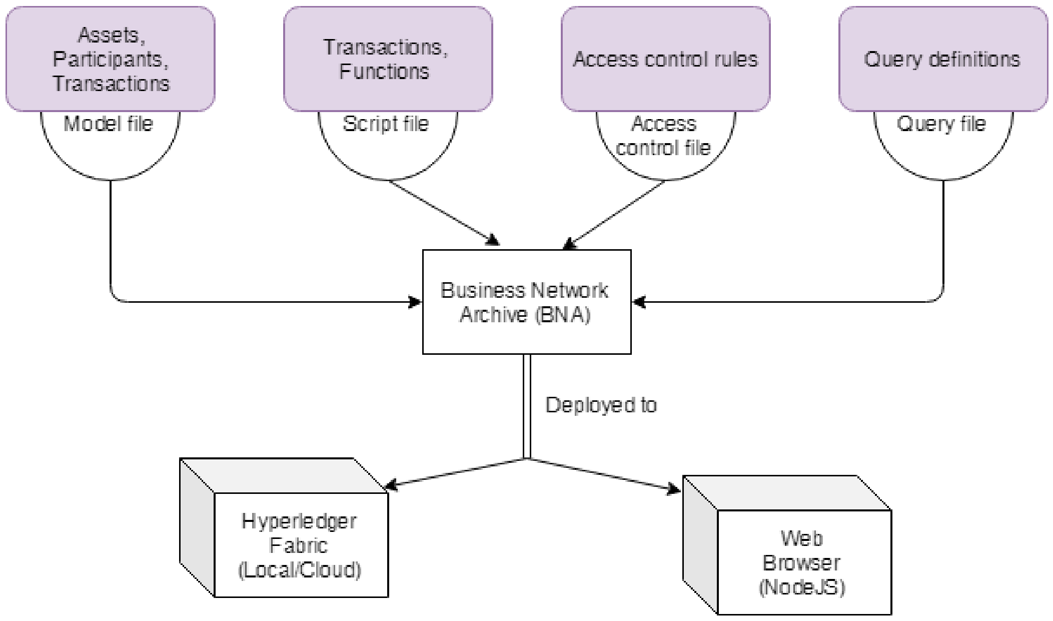


Figure 3.3: Business Network Archive

Figure 3.3 represents the elements of a BNA. A BNA is a set of files containing the blockchain's logic, which can be deployed directly to Fabric. Several files compose it:

- *Network Model*: a definition of the entities that participate in the blockchain, including *participants, assets, concepts, transactions* and *events*. Participants are the ecosystem's nodes, which submit transactions to the blockchain. Assets are entities to be protected and managed by the blockchain. Concepts are abstract classes that are not assets, participants or transactions. Transactions and events are defined by a transactionId or eventId, respectively, and a timestamp.

- *Business Logic*: logic for the transaction functions (i.e., chaincode written in NodeJs). Chaincode defines how participants interact with assets, using defined transactions.

- *Access Control Logic (ACL)*: access control rules are defined using Hyperledger Composer access control language. Rules define the rights of different participants in the network, using *CRUD access control* (create, read, update or delete). The default user System has permissions to perform all operations. Participants can access their permitted resources and transactions. Access to a specific asset can be mediated through transactions. Each rule is identified with a name and description. An action, which is restricted to a participant, under certain conditions is then ALLOW, allowing the participant to perform the specified operations or DENY, being the issuer blocked from operating a specific resource.

It is possible to assure that the modifications made to an asset are made through a specific transaction, allowing for fine-grain access control under certain conditions.

- *Query File*: Composer supports both *named queries* and *dynamic queries*. Named queries are exposed through the GET methods from the generated REST server. Dynamic queries can be built at runtime by the client.

Composer Playground is a web-based user interface used to model and test business networks, which allows to visualize operations done on the *Composer REST Server*. Composer REST Server generates a REST API server, based on the business network definition. This API can be used by client applications and allows us to integrate non-blockchain applications in the network. The REST API server can be used to connect the off-chain component to the blockchain.

The implementation of the blockchain system is encapsulated in the BNA, while the integrations use the Composer REST server. Figure 3.4 represents a typical full-stack blockchain solution. The idea is the business logic to be run by the blockchain, while REST APIs expose the blockchain logic to front-end clients, which can be used by blockchain participants to interact with it.

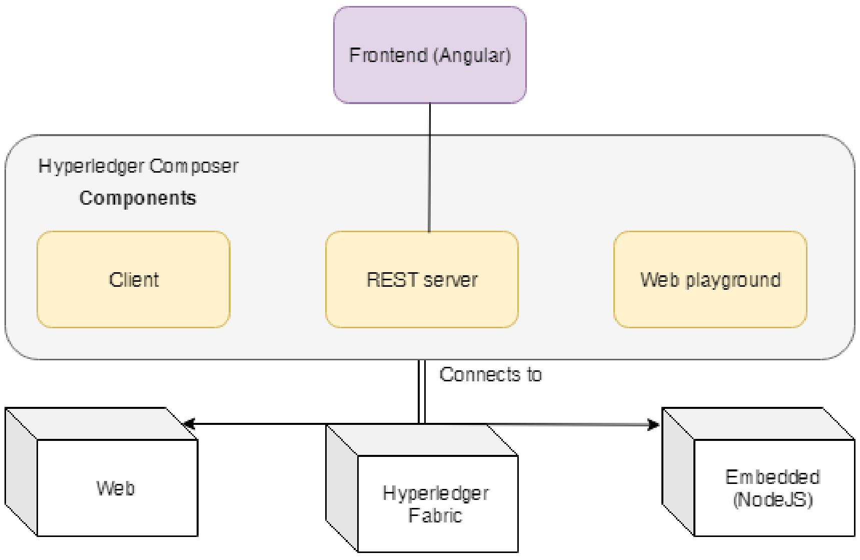


Figure 3.4: Typical Hyperledger Composer solution architecture

3.6 JusticeChain Architecture

In this section, the architecture of JusticeChain is introduced. The assets to be protected are applicational logs generated by an information system related to the judicial system. The proposed solution is scalable when it comes to supporting different organizations, different types of logs and different auditors. This modular design allows the decomposition of the JusticeChain's

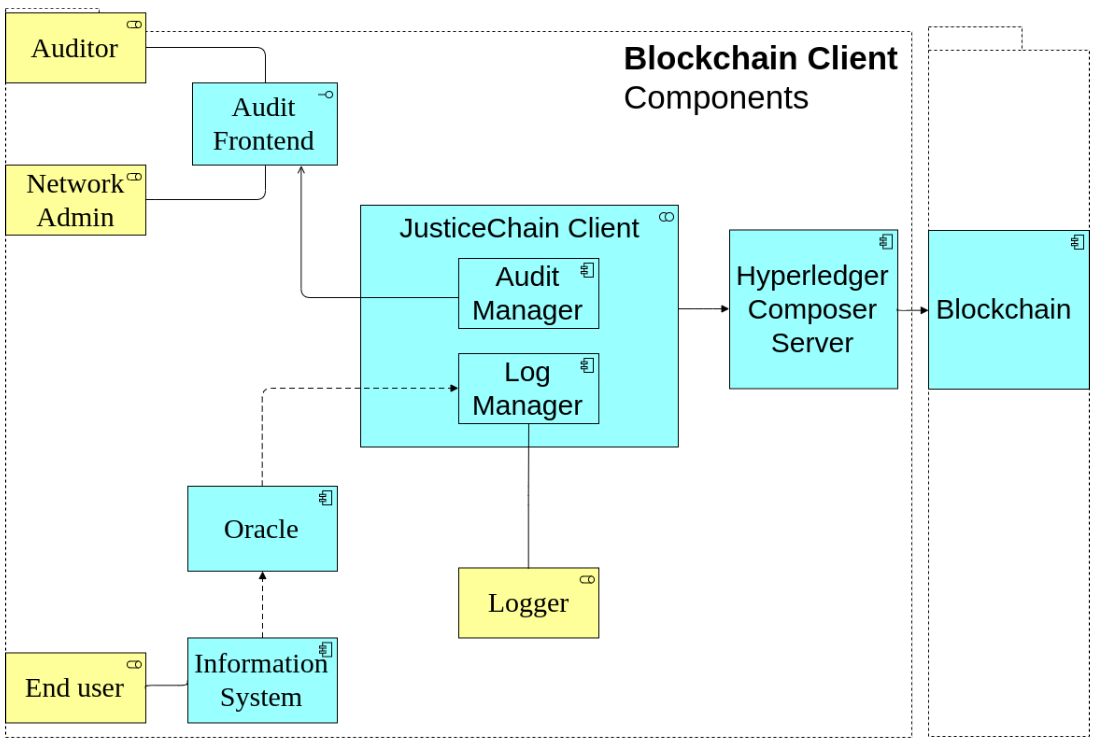


Figure 3.5: JusticeChain Architecture

architecture on what is the blockchain component architecture (ledger, permission management, transaction validation and definition, data model) and blockchain client component architecture (off-chain data, off-chain control, audit components) [56]. The architecture is represented in Figure 3.5 using Archimate language [49].

The blockchain component stores applicational logs and enforces blockchain configurations concerning the different participants on the network. The blockchain client component ensures that participants can access the applicational logs via submitted transactions, and can audit logs, under certain circumstances. A more detailed description of the blockchain component architecture and blockchain client component architecture are available in Section 3.6.1 and in Section 3.6.2, respectively.

3.6.1 Blockchain Components

In Fabric, there is the possibility to assign different trust levels to individual peers. Each role has different permissions on the operations that can be applied by participants to the ledger. Through the definition of endorsement policies, one can put more or less trust in a specific set of endorser peers, making the trust system independent from the consensus algorithm to be used.

For implementing such a blockchain infrastructure, one needs different types of nodes: orderer peers (orderers), endorser peers, and committing peers (peers). Orderers from the ordering service provide delivery guarantees of blocks composed of transactions while assuring atomic communication (consensus). Orderers are bootstrapped with a configuration file. Such configurations can be deployed by network administrators. Peer nodes can be endorser nodes, which hosts and execute instances of the chaincode or committing peers, which host instances of the ledger [3]. Endorser peers can be committing peers at the same time. Anchor peers make sure peers in different organisations know about each other, being a vehicle for inter-organisation communication, within the blockchain. For the sake of simplicity, we omit the presence of anchor peers, which are not essential to define for our use case. This research use case needs two different chaincodes: chaincode that creates instances of applicational logs (*S1*) and chaincode that accesses the ledger (*S2*), regains logs and retrieves them to the end-user. Endorsing peers have S1 and S2 installed, while committing peers have S2 chaincode installed.

A custom certificate authority can be used to issue identities for each participant on the network, like the Portuguese government certificate authority. Keys can then be distributed using smart cards. Issued certificates are used by participants to sign transactions, assuring non-repudiation of transactions. The Portuguese government can also use *Fabric's CA*, the built-in CA from Hyperledger Fabric, to issue CA certificates.

As there is limited trust between participants, and there is a responsible administrator, there is no need for more than one orderer. Nonetheless, for assuring decentralisation, one can deploy multiple orderers belonging to different organisations (having multiple orderers is unlikely to be a performance bottleneck [3]).

Each organisation that participates in the network, and thus interested in auditing a specific information system should maintain at least a peer node that holds an instance of the ledger (committing peer). This allows one organisation to: i) assure log entries are not tampered, and ii) retrieve and access log entries when authorised (e.g., audit). The member organisation responsible for the information system that needs its log entries protected should have an endorser peer. Endorser peers can creates logs, acting as Loggers. As applicational logs should not be shared amongst different member organisations, one has to define access control rules that manage that flow (e.g., only Admins and Auditors from PGR can see applicational logs from Citius). If Auditors need specific permissions (or constraints), those can be tuned for each Auditor by a Network Administrator, via a transaction.

Privacy concerns can arise when there are multiple information systems and multiple organisations involved. Each organisation should be only able to see a subset of the logs of the information system that it is concerned with. For instance, we could define two consortia which are stakeholders from Citius and SITAF, respectively: IGFEJ, IGSJ, COJ, PGR, CSM and IGFEJ, IGSJ, COJ, PGR and CSTAF. In order to keep SITAF data private from CSM, and to keep Citius data private from CSTAF, one has some solutions such as *i)* create a channel for each consortium *ii)* use Fabric's feature, private data, and *iii)* tune access control rules via Composer.

A different channel has a different shared ledger, therefore ensuring data privacy. However, creating separate channels does not allow for all channels participants to see a transaction while keeping part of the data private (fine-grain data privacy is difficult to achieve), and adds performance overhead [48]. Private data can be used in this case, which allows a subset of organisations the ability to endorse, commit, and query private data. The last option consists of managing data access through chaincode through Composer's access control rules, see Section 3.7.1). We chose the last option for its excellent performance and ease of implementation.

Figure 3.6 depicts the network participants for one possible consortium (IGFEJ, IGSJ, COJ, PGR, and CSM), which forms a consortium (Portuguese Justice Consortium), the network configurations applicable to the orderer (N), the communication channel (Logs' channel) and the certificate authority (IGFEJ-CA). Each colour indicates the organisation that owns that resource. Each organisation might have blockchain clients, which are connected to Fabric through Composer and JusticeChain Client.

3.6.2 Blockchain Client Components

JusticeChain is a full-stack application that leverages Fabric to secure audit logs while providing audit support. The Composer REST Server is used to interact with the underlying Fabric blockchain; hence, it is a blockchain client component.

As presented in Figure 3.5, the blockchain client comprises two fundamental entities: the *Audit Manager* and the *Log Manager*. These two components collaborate to serve as the blockchain client, allowing two types of participants to access the blockchain for different purposes. Both the Audit Manager and Log Manager expose *application programming interfaces* (APIs), which allow the *Audit Frontend* and *JusticeChain Oracle* to access JusticeChain functionalities. JusticeChain, on its turn, communicates with the blockchain via the Hyperledger Composer API. Composer, on its turn, communicates with Hyperledger Fabric via Fabric's API. The JusticeChain blockchain client is composed by:

- JusticeChain Client: is a collaboration between two components - Log Manager and Audit Manager. The JusticeChain Client communicates with the blockchain network via a set of REST APIs provided by Hyperledger Composer.

- JusticeChain Oracle (Oracle): An oracle in the context of our problem is a component that retrieves applicational logs from an outward log repository. The oracle can create

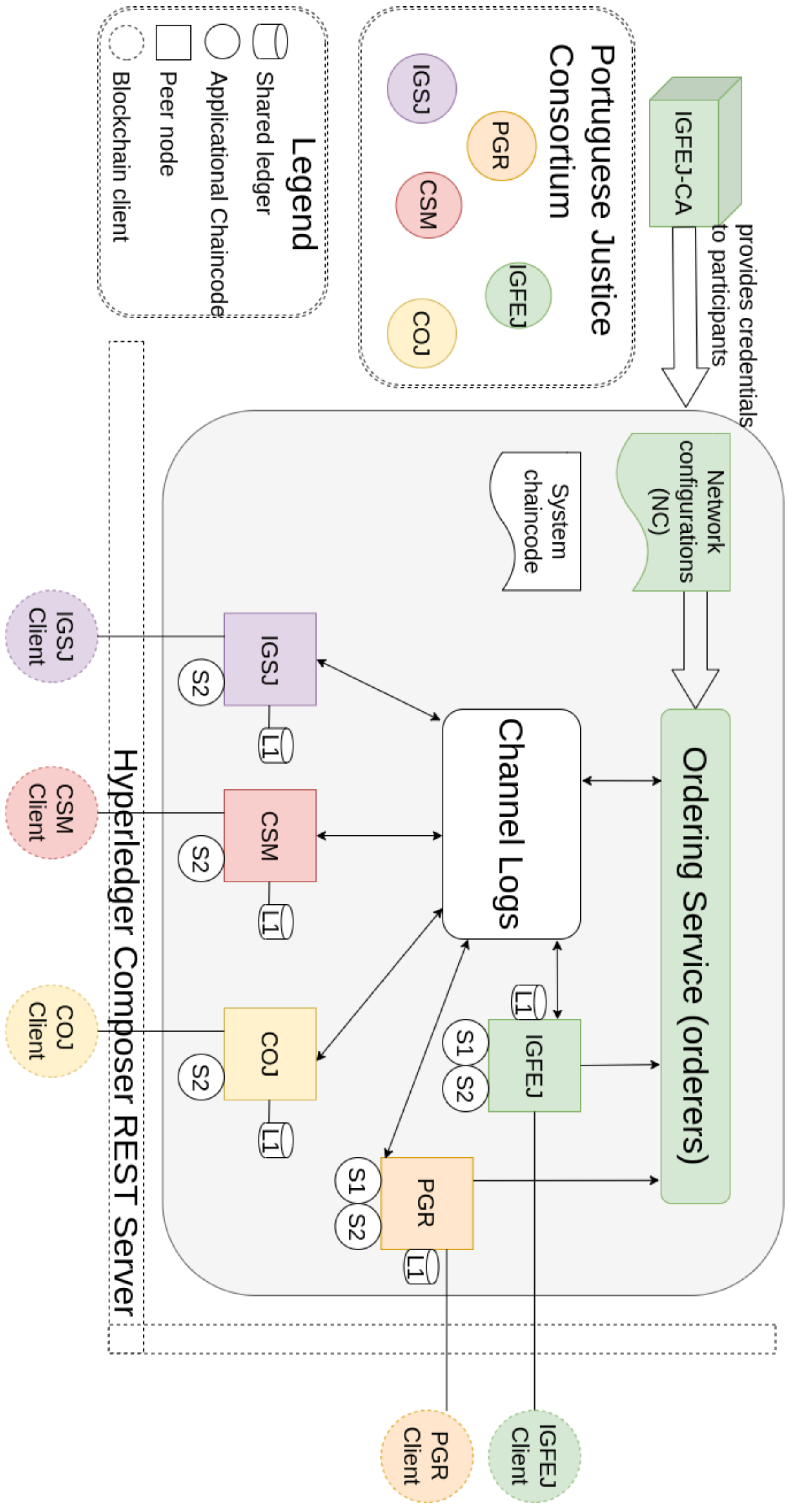


Figure 3.6: JusticeChain Blockchain Architecture - Portuguese Justice Consortium. This consortium is formed by 5 organizations (IGFEJ, IGSJ, COJ, PGR, and CSM), which hold 5 peers, 1 orderer (O), running network configurations NC, 1 certificate authority (IGFEJ-CA), operating on channel Logs.

a buffer of logs and send them to the Log Manager from time to time or send them in real-time. The optimal scenario happens when logs are sent in real-time, as it diminishes the load from the blockchain and information can be exposed to auditors in real-time.

- JusticeChain Log Manager (Log Manager): is connected to one or more oracles. When the Log Manager receives a log, preprocessing is applied, as anonymisation or standardisation. After that, the Log Manager submits a transaction to the blockchain, on the correspondent Logger's behalf.

- JusticeChain Audit Manager (Audit Manager): sends transactions to the blockchain on behalf of the corresponding Auditor or Admin who needs to audit logs. The transactions are, in fact, queries on the blockchain ledger, allowing the JusticeChain client to retrieve the desired applicational logs. The Audit Manager is listening to events emitted by the blockchain, which correspond to suspicious activity alerts.

- Audit Frontend: is a user interface that allows the stakeholders (Auditors or Network Admins) to retrieve the applicational logs, via the Audit Log Manager. Allows obtaining the logs in a certain time frame, allowing auditability.

- Log Repository: corresponds to the repository that stores logs (i.e. database).

- Hyperledger Composer (REST) Server: is generated from a business network archive (see Section 3.7), and exposes an API that the JusticeChain client can use to access the blockchain. Hyperledger Composer Server access Fabric using its API.

In addition to acting as a proxy between frontend applications and the blockchain, the JusticeChain Client, allows end-user authentication to the blockchain network. A local database stores local end user's credentials. This way, one can map local authentication credentials and the user's cryptographic identity on the blockchain network, providing traceability. For instance, an IGFEJ employee can use their government credentials to authenticate in JusticeChain, linking his profile with the blockchain identity. We provide a user interface that handles authentication, queries the blockchain and supports new user enrollment and registration.

3.7 JusticeChain Implementation

We use Hyperledger Composer to implement the blockchain components. The implementation comprised the definition of the Business Network Archive, a file containing the data model, access control rules, transactions (chaincode) and queries associated with the blockchain. To implement the blockchain client, we used Composer's API[5], NodeJS and Angular[6].

3.7.1 Blockchain Components

The chaincode responsible for storing logs was implemented using NodeJS v8.9.0 and using the Hyperledger Composer API version 0.20.8. The implemented data model follows the model described in Section 3.4.

Composer uses Certificate Authorities (CA) that are used to generate the key pairs necessary to enrol different participants on the network. The CA server configuration file contains a Certificate Signing Request (CSR) section that can be configured. The CSR was customized to generate X.509 certificates and keys for ECDSA signatures with curve *prime256v1* and signature algorithm *ecdsa-with-SHA256*.

We implemented the solution in a way that allows new participants to join the network, as well as supporting different types of logs. Nodes have specific permissions concerning which information can be seen and accessed (access control rules). The nodes from the administrating organization have higher permissions on the system and may validate and execute smart contracts and transactions, whereas other nodes can validate transactions and access the ledger. These configurations allow for privacy and confidentiality of data by enforcing the specification of rules for access control applied to each participant on the network.

3.7.2 Blockchain Client

The blockchain client comprises a client written in NodeJS, and a component generated by Hyperledger Composer: the Composer REST API. Composer REST API exposes JusticeChain's logic, namely from the Audit Manager and Log Manager. It communicates with the blockchain, providing an entry point to access chaincode. Chaincode then, based on the invoker's permissions, accesses the distributed ledger and executes the desired actions: returns logs to the client to be audited or inserts new log entries.

The Composer's built-in user interfaces were used as the JusticeChain's user interface, which allows the visualization of the available routes on Composer REST API and the inspection of Logs. In order to emulate the JusticeChain Oracle, we created a script to generate individual log entries.

[5]https://hyperledger.github.io/composer/v0.19/api/api-doc-index
[6]https://angularjs.org/

Chapter 4

Extending JusticeChain with Blockchain Access Control

In Chapter 5, we evaluated JusticeChain against the proposed goals, concluding that such a solution is suitable to improve the *status quo* at IGFEJ. JusticeChain presents an effective way to protect justice data, where different entities and users can access it with different permission levels. Nonetheless, there are some limitations: firstly, the possibility of centralisation, as the network participants must trust the Loggers and the JusticeChain components (Log Manager, Audit Manager). The existence of several Loggers controlled by different organisation alleviates this problem. Secondly, there is a decoupling between the system's logic and the log content, as the programmers must specify which activities are recorded into a log. As a consequence, relevant user behaviour can be lost, compromising audits.

The first limitation seems relatively straighforward to solve, as discussed in Section 5.2.4. Regarding the second limitation, we can relate this limitation to access control. In fact, one can see the blockchain access control problem as a generalisation of the log storing problem: logs are the product of users' access to system resources. If such access to all resources is mediated through the blockchain, there is no need of safely securing logs: all access requests history will be recorded in the shared ledger, when a transaction is initiated. This implies that peer nodes are able to see who has accessed or tried to access a specific resource. A blockchain access control system, therefore, constitutes a way to automatically log all user activity, creating secure data that auditors can use. Furthermore, it allows distributing not only confidence towards the log storage but also towards the authorisation process.

In multi-stakeholder scenarios, where the authentication and authorisation of end-users concern several organisations, one needs a reliable mechanism to provide secure access control [50].

As an additional contribution that improves JusticeChain, we propose a secure, scalable

Feature	**JusticeChain**	**JusticeChain v2**
Store applicational logs	✓	✓
Retrieve applicational logs	✓	✓
Manage the audit authorizations	✓	✓
Verify logs' integrity	✓	✓
Verify auditors' compliance	✓	✓
Log access requests to resources	x	✓
Log changes to access control policies	x	✓
Store access control policies	x	✓
Evaluate access control policies	x	✓

Table 4.1: JusticeChain and JusticeChain v2 features

blockchain access control solution, which includes the responsibility of protecting and managing access to the logs. Such a system decentralizes trust at a larger scale. In particular, we propose JusticeChain v2, an ABAC system based on Hyperledger Fabric, based on previous work [41, 31], which is extended to the domain of the Portuguese justice. Access control components such as the Attribute Manager (AM), the Policy Decision Point (PDP), and the Policy Administration Point (PAP) are embedded in smart contracts, executed by Hyperledger Fabric. We perform experiments on the access control application using Hyperledger Caliper, based on multiple configurations (including, but not limited to different world state databases and consensus methods).

4.1 System Model

A centralised access control system suffers from some issues [41]: i) risk of privacy leakage, and ii) risk of a single point of failure. A blockchain-based access control system could verify access control requests in a decentralized way. Smart contracts allow to monitor and enforce complex access control decisions.

The decentralised nature of the blockchain makes the technology suitable to address the risks mentioned above. The fact that there is no single point of failure alleviates the risk of privacy leakage. A shared ledger common to several parties eliminates centralization, increasing dependability.

The use cases that JusticeChain v2 can address, and are applicable to IGFEJ are expressed in Table 4.1.

This solution empowers both the owner and subjects, which is desirable in multi-stakeholder scenarios, where the end-users are utilising a third-party information system. The subjects have a mean for verifying which policy has been enforced when they performed an access request which has been denied. In practice, there is no way to the resource owner to deny access to a resource by a rightful requester. On the other hand, the resource owner has a mean to leverage audits, while being assured that no user had subverted the system.

The access control channel holds one the Portuguese Justice Consortium nodes (IGFEJ, IGSJ, COJ, PGR, and CSM). These five nodes would be peer nodes, that commit transactions and maintain the state of the ledger. IGFEJ, in particular is also an endorser, which can execute smart policies (access control policies embbeded in smart contracts). Clients are the systems that use this system, such as Citius. As IGFEJ adminstrates Citius, it should be the entity which is responsible for access policies evaluation. For that, each PEP from the information system to be integrated has to be refactored.

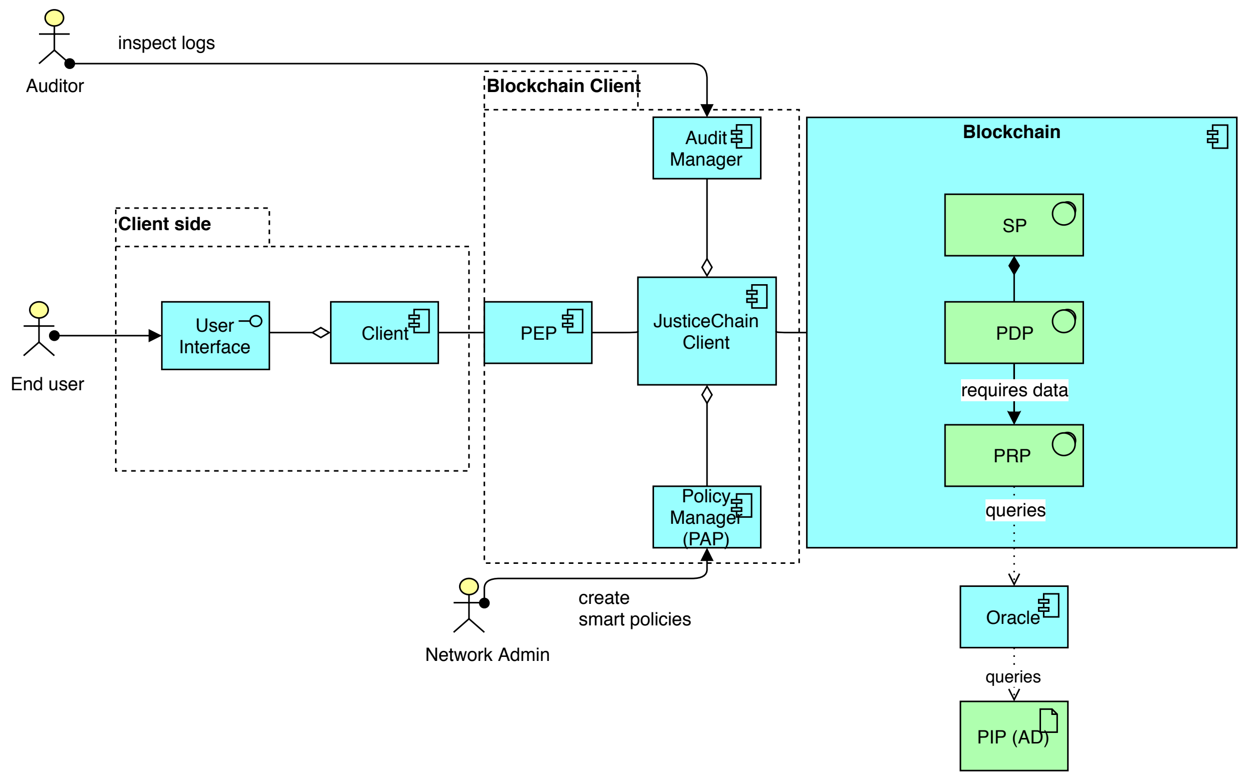
The workflow of users remain unaltered. The only difference is that the authorization requests are now mediated through several nodes representing the Citius system, as the evaluation of access control policies could be not trusted for the subject of the request who, instead, would like to have unfair access to resources.

Each interested government entity is a node that contributes to the network, by validating and auditing transactions. Other organisations' nodes participate in the ecosystem as auditors and validators. IGFEJ's staff will not be able to subvert the system to hide their faults, assuring the stakeholders non-repudiation and transparency. This way, IGFEJ provides guarantees that they are not taking advantage of their privileged position. This solution makes difficult to IGFEJ, judges, court officials, probation officers to abuse their privileges, as they cannot delete their traces.

4.2 JusticeChain v2 Architecture

In this solution, the blockchain will act as a mediator between the entity requesting access to a specific resource and the entity that manages that resource. The JSON markup language[1], is used to define attribute-based access control policies, based on the node-abac package [2], which allows IGFEJ staff to define access control policies. Alternatively, a language based on XML[3], XACML[4], can be used, and allows for fine-grain access control and scalability [29]. We may have IGFEJ as the only PAP, as the creator and manager of access policies. It is possible to decentralize the access control policies management process, by allowing more node to be a PAPs. Even if IGFEJ is the only PAP, the transparency offered by this solution distributes the trust and the responsibility about its access control policies. Figure 4.1 represents the architecture of

[1]https://www.json.org/
[2]https://www.npmjs.com/package/node-abac
[3]https://www.w3.org/XML/
[4]https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html



"

Figure 4.1: Blockchain-based access control architecture for IGFEJ

the proposed solution.

There are several options for the PIP to retrieve attributes. As mentioned on Section 3.1, IGFEJ depends partially on third-party companies to serve its infrastructure (stores user information in an access directory, stored in a cloud), one question may arise: should matters of national security be delivered to a third party with economic incentives? In case not, one option is storing the AMs and user attributes on the blockchain. This may arise privacy concerns, which we will later discuss. Each PIP attached to each SP would retrieve the necessary attributes from a specific AM (containing information concerning each user or role on the blockchain [31]), and pass them to the PDP. Although methods for revoking and upgrading such attributes can be programmed, this might not be the most scalable solution, as more information will be on-chain. As exposing users' information is a problem within the network's nodes, this approach is not suitable, because AMs expose users' information (i.e., username and password). Alternatively, the PIP can access the AD. By making this differentiation, we decouple the responsibility of evaluating a request with the responsibility of storing users' attributes. In case attributes are stored in the AD, the PIP needs to obtain that information via an oracle. Although we are relying on a centralized authority to keep and secure this repository, it is a more scalable solution, as user management can be done off-chain and with the already implemented technologies. After smart contracts evaluate SPs against their respective attributes, the PDP returns its decision to the PEP.

A critical architectural decision to make is which functionality is going to be implemented on the blockchain and which is going to be implemented on application [56]. In a first approach, only SPs will be on-chain. The JusticeChain client connects several off-chain components with the blockchain. This solution not only logs all accesses in a secure way but also provides a framework to control all access controls in respect to the participants in the network. Nodes from the private network can access the blockchain and check the transactions history, enabling auditability. Automatic auditing techniques can be developed. A fine-grain solution is obtained, enforcing right access validation on service providers.

4.3 Implementation

For implementation purposes, we utilized Hyperledger Fabric. We developed three smart contracts in javascript that implement the access control system, namely:

- Chaincode *Record*: stores access control policies, subjects, or resources.
- Chaincode *Update*: updates access control policies, subjects, or resources.
- Chaincode *Query*: queries subject or resource attributes based on their keys.
- Chaincode *PDP*: evalautes an access control request.

The chaincode record receives as arguments a $subject_{key}$ and a *subject*. A subject is a JSON object containing specific properties, while the subject_{key} is a unique identifier for the subject. The chaincode first checks if the subject_{key} already exists in the world state. If so, it throws an error. Otherwise, the world state stores the subject_{key}. Storing access control policies, subjects, or resources is similar: the difference lies in the input arguments. For recording an access control policy, the chaincode receives an access control policy. The update chaincode updates an access control policy, subject, or resource. It receives as an input a subject_{key} and a subject. It then updates the subject_{key} with the content of the subject. Regarding the update chaincode, updating access control policies, subjects, or resources is similar.

The query chaincode receives a key and returns the corresponding object. The PDP chaincode evaluates an access control request, returning TRUE (when the access is granted) or FALSE (when the access is denied). This chaincode receives a subject key, resource key, a rule and a policy key. The keys allow the system to retrieve the corresponding objects, while the rule specifies an entry on the access control policy. It retrieves the subject, resource and policy from the world state, and parses it into JSON objects. Nextly, an abac node[5] is built from the parsed policy. Finally, we evaluate the decision taking into account the subject, resource, and rule picked from the policy.

[5]Used package: https://www.npmjs.com/package/node-abac

Chapter 5

Evaluation

In this chapter, we evaluate JusticeChain and JusticeChain v2. We describe the environment used to evaluate the system, the used metrics and the evaluation methodology. Then, an approach to perform the system evaluation is presented, accompanied by the definition of metrics and goal targets. Later, the information provided by the evaluation is collected, analysed and discussed. Finally, we provide a discussion about the systems limitations and possible approaches to address them.

5.1 Evaluation Methodology

In this section, we will measure the performance of JusticeChain and JusticeChain v2. Definitions of terms used in this section are specified in Section 2.1.4.

We focus the evaluation on the costs of protecting applicational logs, as it is JusticeChain's main feature. The performance of actions such as changing a Logger level or editing the properties of a Logger is not going to be addressed, as such operations represent one lightweight standard transaction. Furthermore, in production the large majority of transactions aims to create applicational logs.

With the performed experiments, we tackle the following three questions: i) What is the maximum throughput JusticeChain can achieve, i.e. how many logs per second can it record per time unit? ii) what is the latency at the maximum throughput, i.e., what is the time window needed for logs to be secured?, and iii) what is the cost, in terms of storage, of protecting applicational logs, i.e., what is the scalability of JusticeChain? Answers to such questions allow us to conclude the suitability of JusticeChain about the proposed use case.

5.1.1 Load generating client

We study performance metrics as transaction success rate, transaction throughput, read throughput, transaction latency, read latency and resource consumption, as advised by the Hyperledger community [1]. The read latency corresponds to the difference between the read request and the received reply, whereas the read throughput is the ratio between the total read operations and the total time in seconds. The transaction Latency corresponds to the difference between the submit time and the transaction confirmation time. It is the time taken for the transaction to affect the network. The transaction throughput isthe ratio between the total committed transactions and the total time in seconds.

For that, we leveraged *Hyperledger Caliper* [2], a project baked by Hyperledger, which aims to facilitate the evaluation of blockchain solutions. The Caliper framework serves as a load generating client, and runs tests based on configuration files, which are used to create a Fabric blockchain topology, as needed, and run a series of tests.

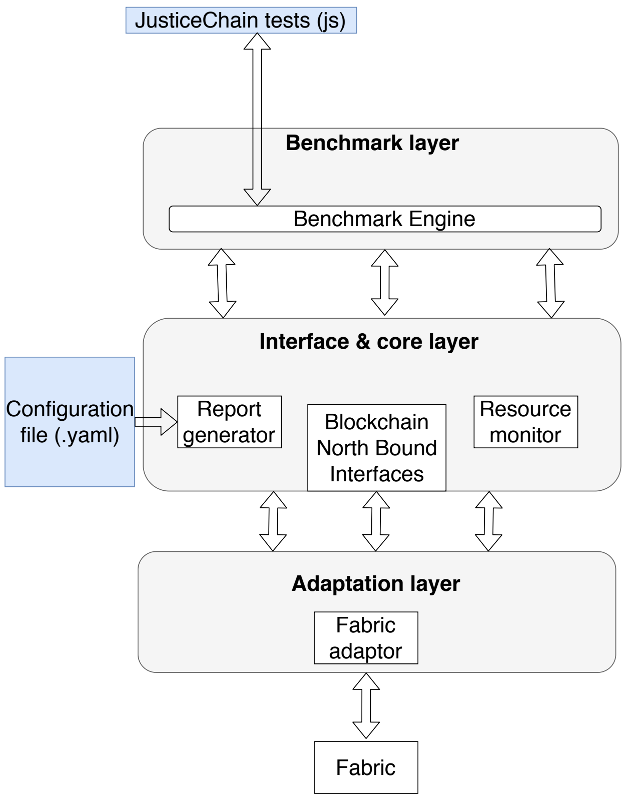


Figure 5.1: Testing JusticeChain with Hyperledger Caliper

Hyperledger Caliper has several layers, the *Adaptation Layer*, the *Interface & Core Layer* and the *Benchmark Layer*, as can be seen in Figure 5.1. The adaptation layer provides integration with different blockchain infrastructures (i.e., Fabric). The next layer provides an abstraction of

[1] https://www.hyperledger.org/resources/publications/blockchain-performance-metrics
[2] Caliper Docs - $https://hyperledger.github.io/caliper/docs/2_Architecture.html$

a given blockchain infrastructure, via a set of northbound interfaces (NBIs). There are several NBIs, which allow to deploy smart contracts on backend blockchain, query the ledger, monitor the resources being used on the benchmarking process, to analyse statistics and to generate a report. The report generator and resource monitor can be tuned, through a definition file. The applicational layer contains tests implemented for the JusticeChain and JusticeChain v2 scenarios (written in javascript).

Caliper uses a benchmarking engine to assess the performance of a blockchain solution. The benchmarking engine initialises the blockchain and starts running the tests, according to a workload specified on a *Yet Another Markup Language (YAML)* file. For each test, a context is created, and transactions are generated and submitted. With Caliper, it is possible to modify the rate and the mechanism at which transactions are submitted. Such tuning is made through *rate controllers*, in order to control the transaction flow of the system. For instance, the *fixed feedback rate controller* sends input transactions at a fixed interval. It is possible to specify a threshold, in which the system stops sending transactions momentarily (number of unfinished transactions). The *Fixed Backlog* rate controller drives the tests at a target loading, which corresponds to a defined backlog of transactions. The controller aims for the maximum possible transactions per second of a system under test while maintaining the backlog level. After each test is run, the context is released, and statistics are collected. Finally, after all, tests are run, a report is generated.

Regarding the load testing, Caliper features' allow us to simulate the behaviour of the Log Manager (tunning the number of Loggers, and the number of transactions issued per second). Citius' behaviour is implicit in the number of transactions that the Log Manager issues to the blockchain component, as every log entry generated by Citius goes to the Log Manager.

5.2 JusticeChain Evaluation

5.2.1 Setup and Test Environment

A replication of the real production environment was set up, with several distributed clients, emulated by Caliper's *client number* option. In our context, a client is a Logger. A machine was deployed in Google Compute Engine (GCE). At GCE, an instance was set up on London, England, UK, with 16vCPU and 60GB of memory, and a 50GB SSD disk.

Tests were performed on top of Docker containers running over Docker version 18.09.6. Each peer, certified authority node and orderer was running on the base image of Hyperledger Fabric. As a state database, instances of Hyperledger Fabric LevelDB images were used.

Regarding the test environment, a simplified Fabric v1.2.0 network with a solo channel comprising two organisations (Org1, Org2) with one peer each and one CA each was considered (See Figure 3.6 from Section 3.6.1). The consensus algorithm is solo orderer (broadcasts the transaction without establishing any real consensus, used for testing purposes). In this evaluation, the used maximum block size was 128MB, the default batch timeout 250ms and the number of transactions per block is 10. We considered a variable number of blockchain clients, as we want to see how JusticeChain reacts to transaction injetion from different nodes.

5.2.2 Throughput and Latency

Understanding JusticeChain's scalability in terms of the transaction throughput, i.e., how many transactions per second can the system handle, and how quickly they are incorporated into the distributed ledger (transaction latency) is essential to justify the suitability of the system with accordance to the Portuguese Justice's requirements.

On an information system with 10,000 users, if each one is performing 100 operations per day that generate a log entry, we obtain an average of approximately 12 tps. As most transactions will be issued during the peak hours, the system should be able to support at least a throughput of 12 transactions per second. Assuming every user creates 75% of the transaction workload during 9 a.m. and 6 p.m., the minimum admissible tps rate is around 20 tps. The target goal for throughput is, therefore, 20 tps. Considering latency, and in order to reduce the time window an attacker can act, we consider a maximum of 60 seconds as the target goal.

In the following tests, we issue 20 tps, the minimum admissible tps rate, at a constant rate. 100, 500, 2,500 and 5,000 transactions were issued, respectively. Furthermore, we varied the number of blockchain clients (in this case, Loggers). The number of Loggers was 1,3,5 and 10.

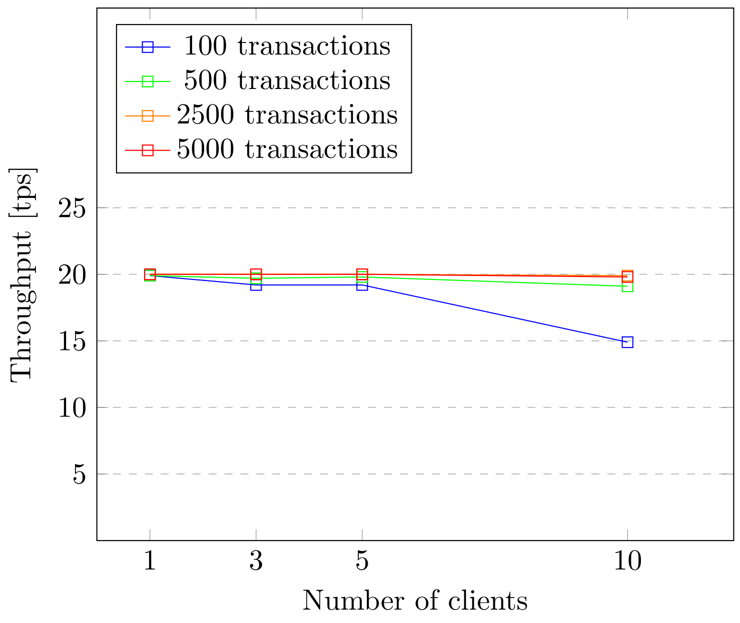


Figure 5.2: Variation of the throughput with the number of clients, at a 20 tps rate

Figure 5.2 shows the variation of the throughput with the number of clients, at a 20 tps rate. One can observe that the throughput decreases with ten clients, for a small number of transactions. This situation happens due to the transaction distribution in Caliper, as it takes considerable time to distribute them versus the time it takes to execute them.

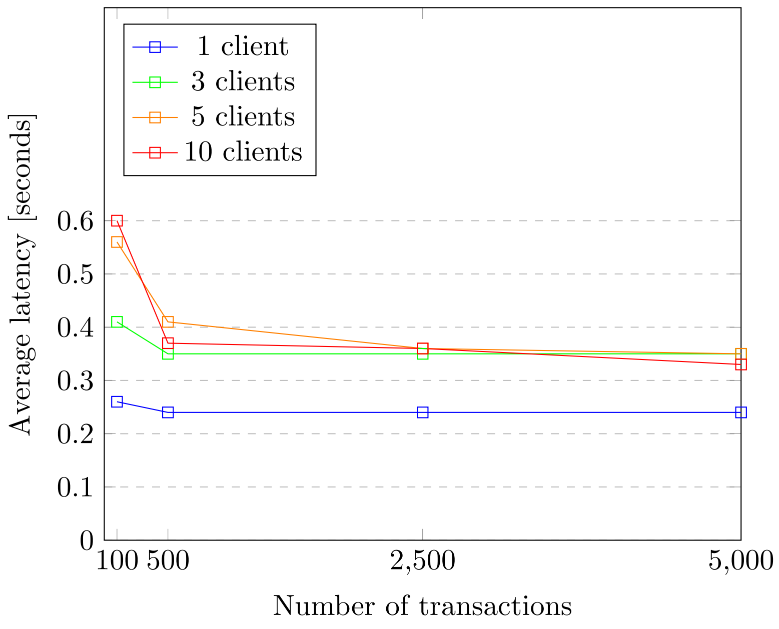


Figure 5.3: Variation of the average latency with the number of transactions, at a 20 tps rate

Figure 5.3 shows the variation of the average latency with the number of transactions, at a 20 tps rate. Usually, lower latency is related with a smaller number of blockchain clients, as the peers that execute transactions only need to return messages to a smaller number of clients. In general, the more clients, the higher the latency.

The results from Figures 5.2 and 5.3 do not provide conclusive information. The system behaves similarly, concerning a variable number of transactions and clients, when the transaction rate is 20, as such rate does not stress the system.

The rate at which transactions are submitted to the network influence its performance (throughput, latency). To understand JusticeChain's throughput capabilities, we issued 2500 transactions, are a variable tps rate, stipulated by the *Fixed backlog* rate controller.

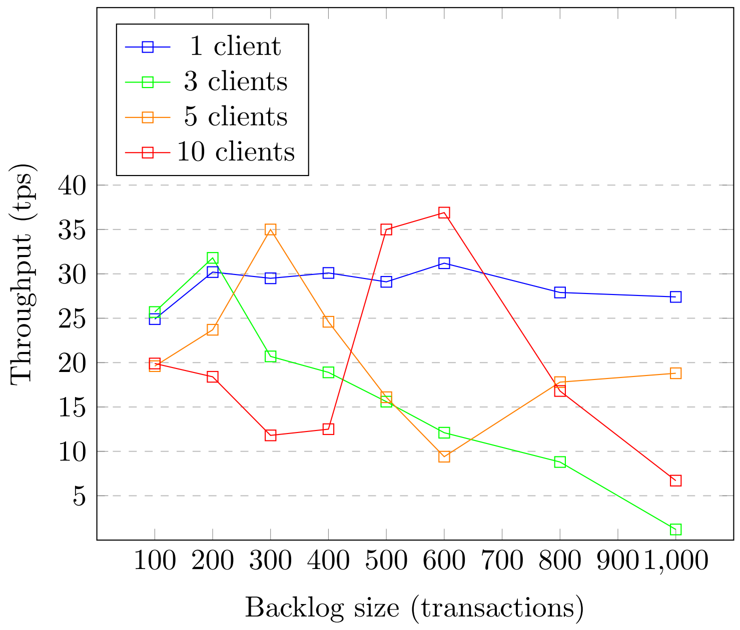


Figure 5.4: Variation of the throughput with the size of the backlog

Figure 5.4 shows the variation of the throughput with the size of the backlog. We can observe that the highest possible throughput is when we have a backlog of 600 pending transactions and ten clients that are sending those transactions. When using 3, 5 and 10 clients, the throughput starts to decrease when the backlog increases, at 200,300 and 600 transactions, respectively.

Figure 5.5 represents the variation of the latency with the size of the backlog, respectively.

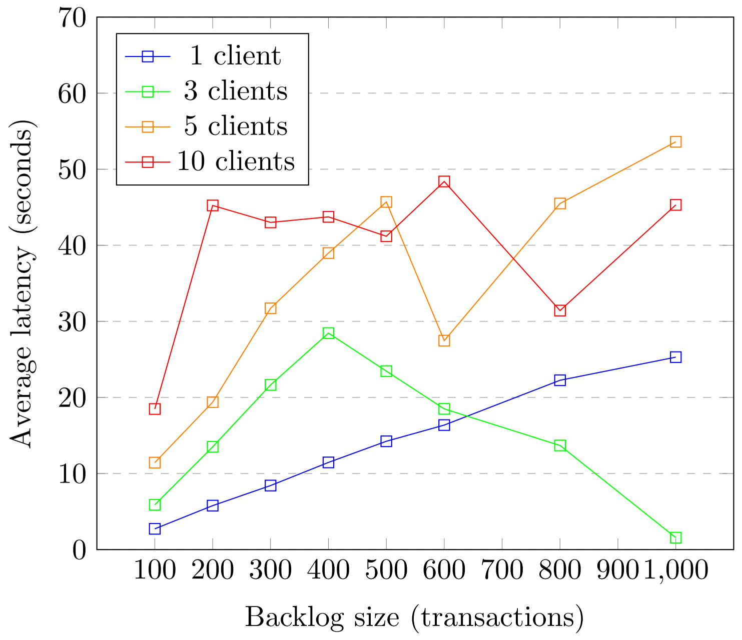


Figure 5.5: Variation of the latency with the size of the backlog

By crossing Figures 5.4 and 5.5, one can extract a relation between throughput and latency. In general, all tests have throughput and latency increased, except the one with three clients when the throughput dropped greatly, the latency as well, as there were a successful few transactions to be distributed.

The ideal client number is 1 for a backlog of 100 transactions if we take into account the throughput/latency ration. We achieve a throughput of around 31.2 tps, with a latency of 16.37 seconds. The maximum throughput we obtain is around 37 tps, at the expense of latency, 48.39 seconds. The test that yields the maximum throughput is in Table 5.1. To obtain such throughput, one needs ten clients. Table 5.1 holds the results of the best performing maximum tps test, including the success rate, send rate, latency and throughput.

Table 5.1: Maximum tps test

Succ	Fail	Send Rate	Max Latency	Min Latency	Avg Latency	Throughput
2500	0	47.6 tps	61.37 s	1.67 s	48.39 s	36.9 tps

5.2.3 Storage

Storage evaluation aims to predict how much storage the Portuguese Justice needs for a long-term solution. We present an experimental evaluation.

The transactions that were used on the evaluation were the creation of an applicational log, *CreateCitiusLog*. Different types of *CitiusLogs* (namely with 10, 20 and 30 attributes) were tested to infer the impact of different log sizes. The CitiusLog with ten attributes corresponds

Table 5.2: Citius Test 1: 100 issued transactions against 1 blockchain client, at a 20 tps rate

TYPE	NAME	Memory(avg)	CPU(avg)	Traffic In	Traffic Out	Disc Read	Disc Write
Process	node local-client.js(avg)	-	-	-	-	-	-
Docker	dev-peer0.org1.example.com	106.0MB	60.44%	640.0KB	563.4KBMB	0B	0B
Docker	dev-peer0.org2.example.com	133.4MB	57.61%	622.66KB	550.3KB	0B	0B
Docker	peer0.org1.example.com	270.2MB	17.25%	1.5MB	4.7MB	0B	1.4MB
Docker	peer0.org2.example.com	244.5MB	16.23%	1.5MB	4.7MB	0B	1.4MB
Docker	ca.org1.example.com	24.6MB	0.00%	42B	42B	0B	0B
Docker	ca.org2.example.com	9MB	0.00%	0B	0B	0B	0B
Docker	orderer.example.com	35.0MB	6.24%	838.0KB	1.5MB	28.0KB	880.0KB

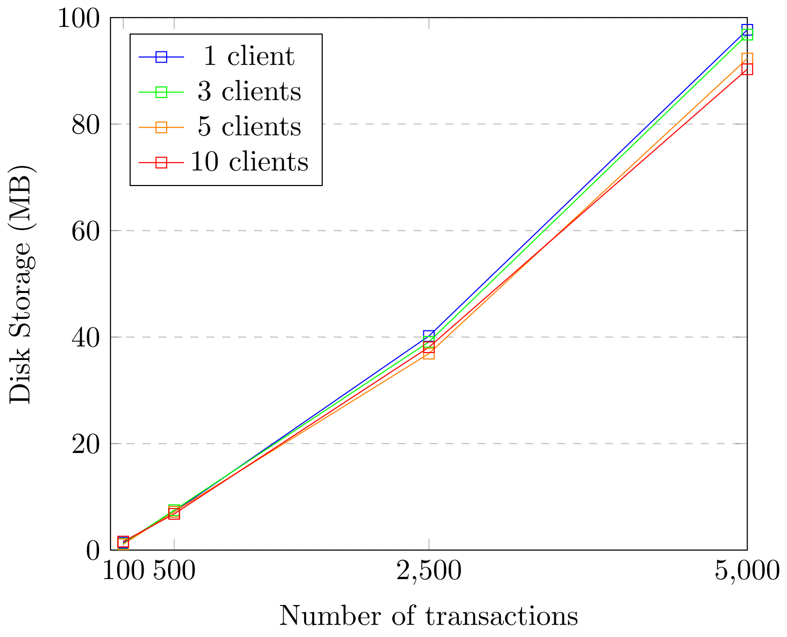


Figure 5.6: Disk Storage required in function of the number of transactions

to a simplified version of the original CitiusLog, which has 20 attributes. A CitiusLog with 30 attributes corresponds to a possible extension of the applicational log.

We executed chaincode that generates applicational log entries on the Logger peer. Caliper calls this chaincode a certain number of times, allowing to assess the blockchain performance of saving log entries. The time taking from the Log generation to Log store on the blockchain is neglected, i.e., the time between the Log generation on an external information system, its retrieval by the JusticeChain Oracle, the reception from JusticeChain Log Manager and the reception from the blockchain is not considered.

From the experience described on 5.2.2, we obtained similar results to the ones present on Table 5.2. From these, we plotted Figures 5.6 and 5.7.

The peers *peer0.org1.example.com* and *peer0.org2.example.com* maintain the global ledger, and have the same *Disc Write*, as both peers commit the same transactions. The *Memory (Avg)*, *CPU usage* and *Traffic* will not be analysed, as they do not comprise consequences to this particular study.

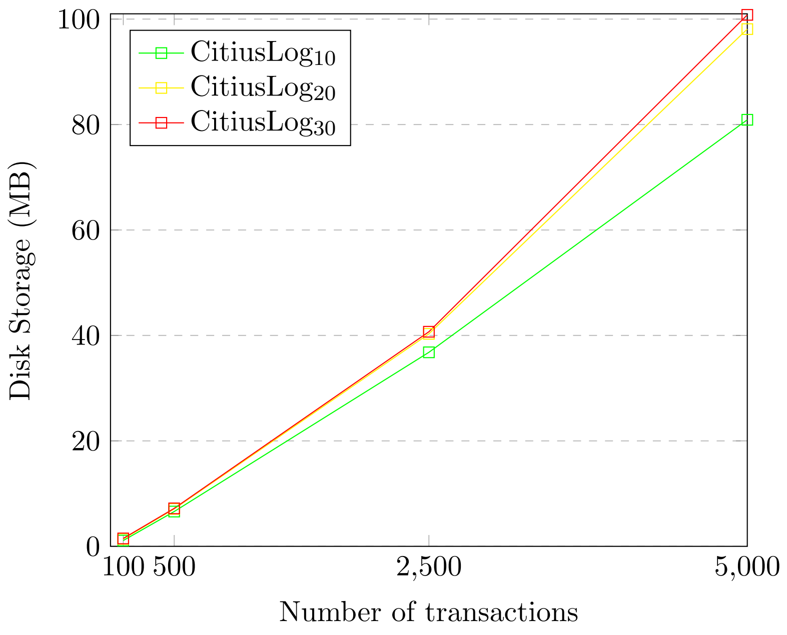


Figure 5.7: Disk Storage required in function of the type of Log

Next, we evaluated the impact of different types of Citius Logs, with 10, 20 and 30 attributes, called $CitiusLog_{10}$, $CitiusLog_{20}$ and $CitiusLog_{30}$, respectively.

Performing a linear regression over CitiusLog_{20} and CitiusLog_{10} allow us to predict how much storage is necessary for one year of transactions.

We neglect the constants from the linear regression, as those are irrelevant when studying the system behaviour on 5,000 transactions or more. Assuming that all 10,000 users perform 100 operations that generate a transaction per day (tpd = 1,000,000), the total storage required for a year, for each peer node holding CitiusLog_{10} is given by:

$$S_{Citius10} = 1.63.10^{-2}.tpd.365 \tag{5.1}$$

For CitiusLog_{20}, the total storage is:

$$S_{Citius20} = 1.81.10^{-2}.tpd.365 \tag{5.2}$$

For CitiusLog_{30}, the total storage is:

$$S_{Citius20} = 2.02.10^{-2}.tpd.365 \tag{5.3}$$

With $tpd = 1,000,000$, and from Equations 5.1 and 5.3, $S_{Citius10}$ and $S_{Citius30}$ yield, respectively, 5.67TB and 7.02TB.

Considering the Amazon Web Services (AWS), namely AWS S3 storage solution, the price

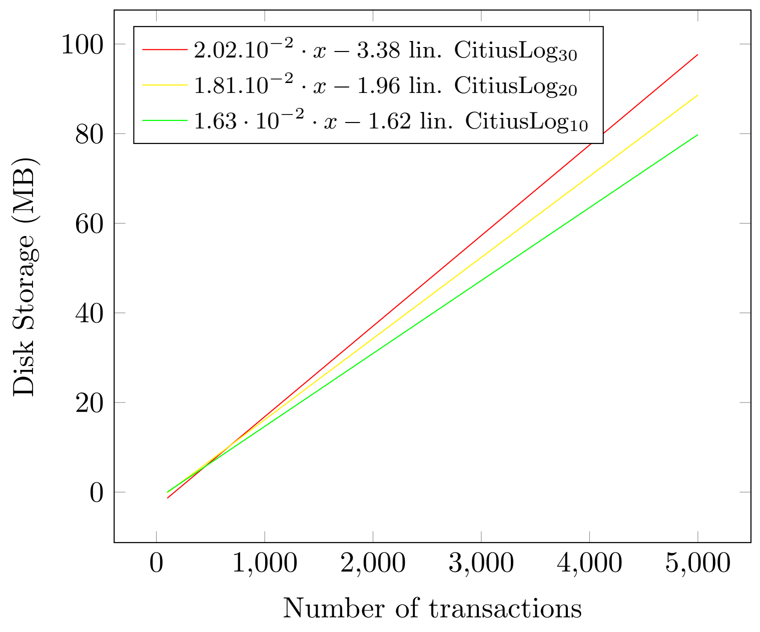


Figure 5.8: Linear regressions of CitiusLog$_{10}$, CitiusLog$_{20}$ and CitiusLog$_{30}$

# Peers	***Storage (TB)***	***AWS S3 Standard ($)***	**Price AWS Glacier ($)**	**Google Cloud Storage ($)**	**Azure Standard GPv2 - Archive ($)**
2	140	294	54	372.7	13.9
3	210	441	84	545.8	20.8
4	280	588	112	566.2	27.7
5	350	735	140	716.8	34.7

Table 5.3: Peer cost on different cloud providers

for five different organizations with one peer each one to store one year of CitiusLog$_{30}$ would be approximately 35TB, or $735. If we consider cheaper options, such as AWS Glacier, the cost is $140 per year, which makes this solution economically viable. Table 5.3 shows the relation between the number of peers, the storage required and its price on different cloud providers.

5.2.4 Discussion

In conclusion, JusticeChain can achieve an optimal throughput of around 37 tps, at the expense of latency, being the average 48.39s. In a practical scenario, with only one blockchain client, JusticeChain achieves 25 tps, but there is a better trade-off throughput-latency, being the average latency 16.37s. It is worth to note that there is a trade-off between throughput and latency. The higher the throughput, the higher the latency, as more transactions take longer to be committed to the ledger.

Regarding storage, we could observe in Figure 5.6 that the number of blockchain clients (Loggers) issuing transactions does not have an impact on disk storage. The number of blockchain clients issuing transactions does not affect the amount of information that has to be written at a peer's disk. Nonetheless, it affects the traffic that goes out of peers, as they need to communicate the transaction's endorsement/committing to more clients. Figure 5.7 shows that for up to 5,000 transactions there is not a substantial difference between saving CitiusLog$_{10}$, CitiusLog$_{20}$ and CitiusLog$_{30}$. Despite not being relevant initially, in the long term, as can be concluded from the linear regressions in Figure 5.8, the difference is significant. Table 5.3 shows the storage costs from several cloud providers. Although costs are one of the most important factors to consider, other features such as replication and instant data retrieval should be taken into account when choosing a cloud provider. A trade-off between security requirements, distribution of responsibilities and monetary cost must be done at IGFEJ, to evaluate how many peers are worth to deploy.

The storage evaluation did not take into account the information stored by the orderer. The orderer saves transaction information (e.g., transaction history, transaction proposals). For a more accurate evaluation, the storage required by the orderer should be taken into account. Our results yield, therefore, an optimistic evaluation concerning storage.

We conclude that the number of attributes has practical implications for the long run of the system. Consequently, the final Log model should only contain the essential attributes. A blockchain solution with more nodes increases the resilience of the system in terms of integrity and availability. Despite having benefits, those are at the expense of performance and storage. In particular, the storage required scales linearly with the number of nodes, scaling to dozens or hundreds of TB per year.

JusticeChain has some limitations. Each peer of the blockchain is required to have its copy of the ledger. This fact leads to an increased storage footprint, as more logs are added through JusticeChain Log Manager. The size footprint also increases the search complexity for verification, in cases the blockchain transaction history has to be verified. In JusticeChain, its space complexity is similar to most permissioned blockchain systems, $O(n)$.

Concerning the time complexity, it depends on the used consensus algorithm. If using a PBFT consensus, in a network with n replicas, the number of messages exchanged are n^2 - n. For each transaction generated within the system, the overall time complexity is $O(n^2)$. This fact can lead to performance problems when replicas are several dozens.

Although Fabric is not theoretically tamper-proof, the existence of a consortium and a strict endorsement policy can significantly reduce the risk of collusion: as more endorsement peers belonging to different organizations are required to endorse a transaction, the more organizations have to collude to change the world state [3]. The endorsement policy to follow can be AND(Org$_1$,Org$_2$, ... , Org$_n$). If there is the need of a trade-off between performance and security, the endorsement policy can be changed to, for example OR(Org$_1$,Org$_2$, ... , Org$_n$) [52].

The throughput limitations, concerning Fabric's maximum theoretical throughput, are due to the several layers that form JusticeChain. The transaction is first generated at Citius, sent to the Log Manager, and from there a blockchain transaction is created and sent to Composer. Composer then transforms the transaction into a Fabric transaction. Composer might be a considerable bottleneck.

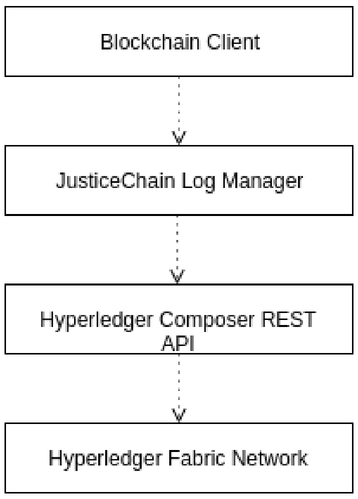


Figure 5.9: JusticeChain layers [32]

Besides removing Composer, other improvements can be done to JusticeChain, such as tuning Fabric configurations. In particular, recommendations from Thakkar et al. [48] to tune the block size, endorsement policy, channels, resource allocation, and state database can be followed. In order to maximize throughput, the batch timeout can be raised (from 250ms to 1s), as well as the maximum number of transactions in a block (from 10 to 100), while the maximum block size can remain at 128 MB.

Lastly, one can see a possible limitation of the proposed solution concerning trust distribution. The limitations tackle the following questions: i) What if there is only one Logger available? ii) What if there is only one oracle that retrieves logs from Citius, controlled by on organisation?, and iii) What if there is only one JusticeChain client, administered by one organisation? JusticeChain appears to have the risk of centralisation, which can, ultimately, defeat the whole purpose of a blockchain solution. Nonetheless, some relievers can be applied in order to assure decentralisation. Regarding the first two questions, one could deploy multiple oracles and Loggers, each pair corresponding to a different organisation. Each Logger would send a unique log entry, with the same ID. Chaincode can be developed to compare the same log entries created by different Loggers. Such chaincode would agree on the log entry to be stored (following a consensus mechanism, for instance, the majority). Regarding question three, administrators can deploy multiple JusticeChain clients, diminishing the risks associated with centralisation.

5.3 JusticeChain v2 Evaluation

In this section, we evaluate the extension of JusticeChain, JusticeChain v2, using the same evaluation methodology from Section 5.1. For evaluation purposes, we take as performance metrics the throughput and latency, as those are vital metrics concerning an access control system. After that, we provide a theoretical evaluation of the storage required to save access control policies, subjects, and resources.

We defined a set $\langle o, s, p \rangle$, in which subject s is authorised to execute privilege p on object o, and operated over them. In order to understand the behaviour of our system, we varied the following parameters: number of transactions, number of peers, send rate, and type of database. We varied the available consensus mechanisms, Kafka [26] and Raft [36]. Kafka is a messaging based log aggregator and distributor, in which producers publish messages to message streams. Messages are stored in brokers, where consumers pull the requested data. Kafka includes a consensus mechanism implemented through Zookeeper, a hierarchical key-value store called ZAB. Raft is a distributed consensus algorithm, in which a leader on a cluster of machines is elected and starts the consensus process.

Finally, we provide a discussion about the limitations of JusticeChain v2 and possible approaches to address them.

5.3.1 Setup and Test Environment

A replication of the real production environment was set up, with several distributed clients, emulated by Caliper's *client number* option. A machine was deployed in Google Compute Engine (GCE). At GCE, an instance was set up on London, England, UK, with 8vCPU and 8GB of memory.

The consensus algorithm is solo orderer (broadcasts the transaction without establishing any real consensus, used for evaluation purposes). In this evaluation, the used block size and batch timeout were the default values.

5.3.2 Throughput and Latency

From Section 5.2.2, we inferred the minimum admissible tps rate, that tackles the average operations per second at Citius. Therefore, the proposed access control system should be able to handle at least 20 access control requests per second. Low latency is highly desired, as access control requests should be dispatched as soon as possible. We set a goal of 15 seconds, as the average latency.

Figure 5.10 shows the variation of the throughput concerning the send rate, both measures in transactions per second. By analysing the graph, we can conclude that the Raft consensus mechanism achieves a higher throughput (around 252) than with Kafka (around 230).

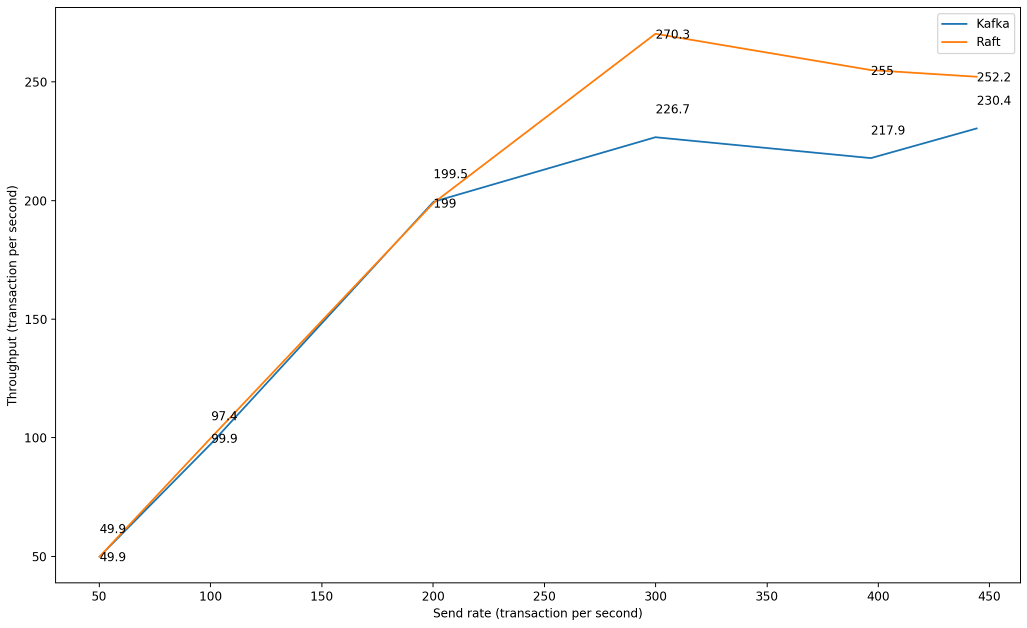


Figure 5.10: Throughput in function of the send rate

As higher throughput usually comes with higher latency, we analysed the impact on the consensus mechanism on the latency. Figure 5.11 depicts the latency in function of the send rate. One can observe that Raft achieves a lower latency (11.69s) than Kafka (13.97s).

We measure the Average latency for the available different chaincodes (Query, Record, and PDP). Figures 5.12 and 5.13 illustrates the comparison.

Finally, we compare the impact of different databases that hold the world state, as well as a different number of peers on the network. Figure 5.14 shows the benchmark between using LevelDB[3], CouchDB[4], and using two organisations with one peer each and three organisations with two peers each.

5.3.3 Storage

For the evaluation of the storage required to store access control policies, subjects and resources, we assume that the storage occupied by each is 1165, 225 and 37 characters, respectively. Assuming each character occupies 1 byte, one can conclude that:

[3]https://github.com/google/leveldb
[4]https://couchdb.apache.org/

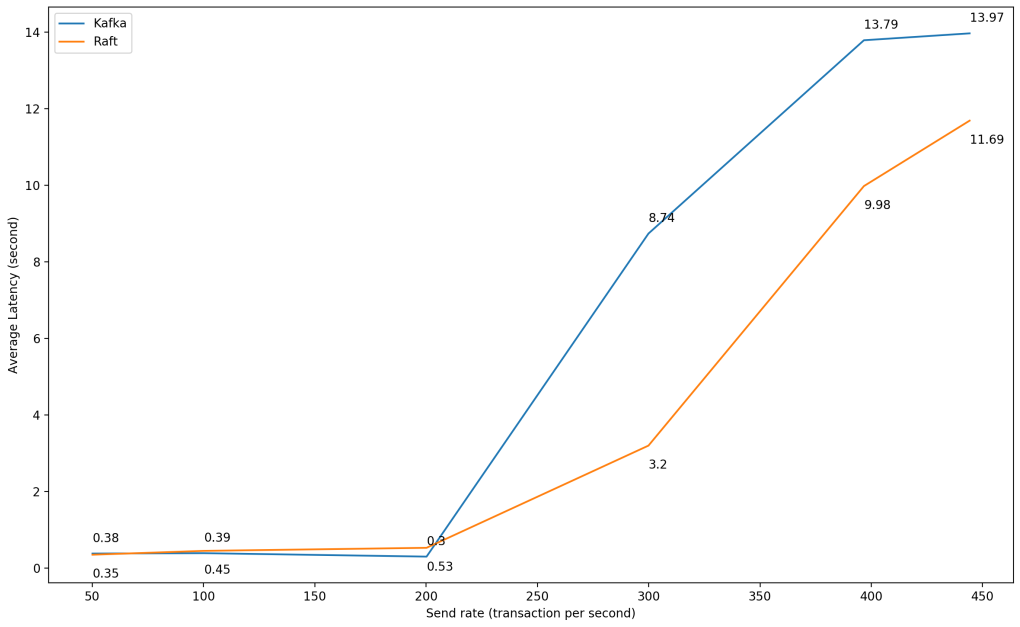


Figure 5.11: Latency in function of the send rate

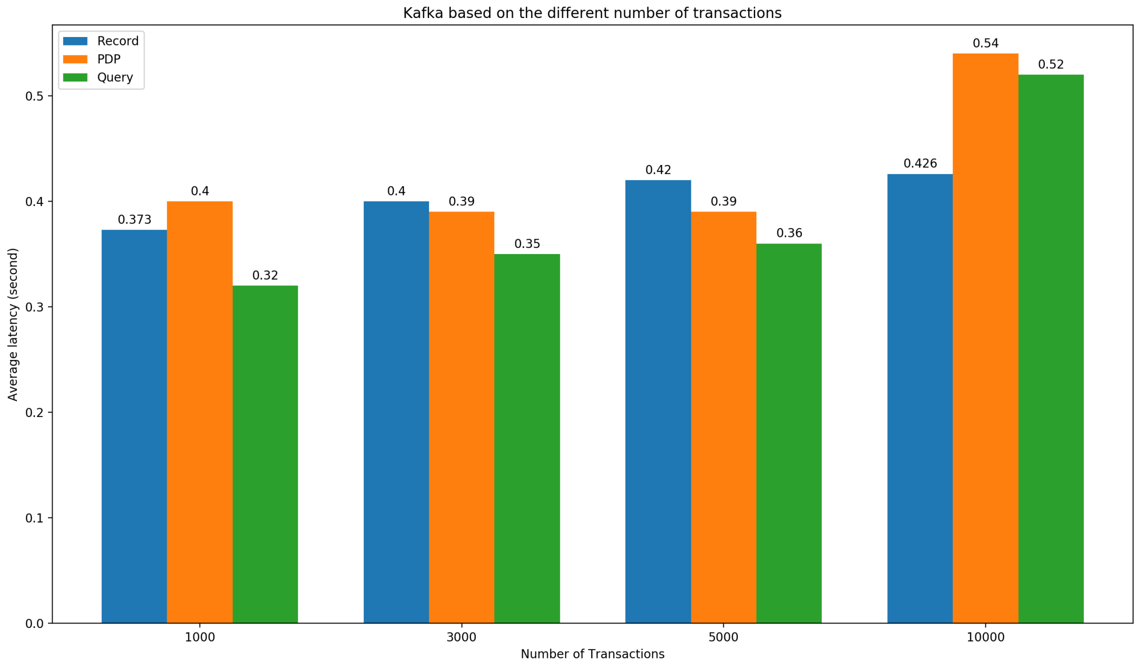


Figure 5.12: Average latency using Kafka, depending on the number of issued transactions

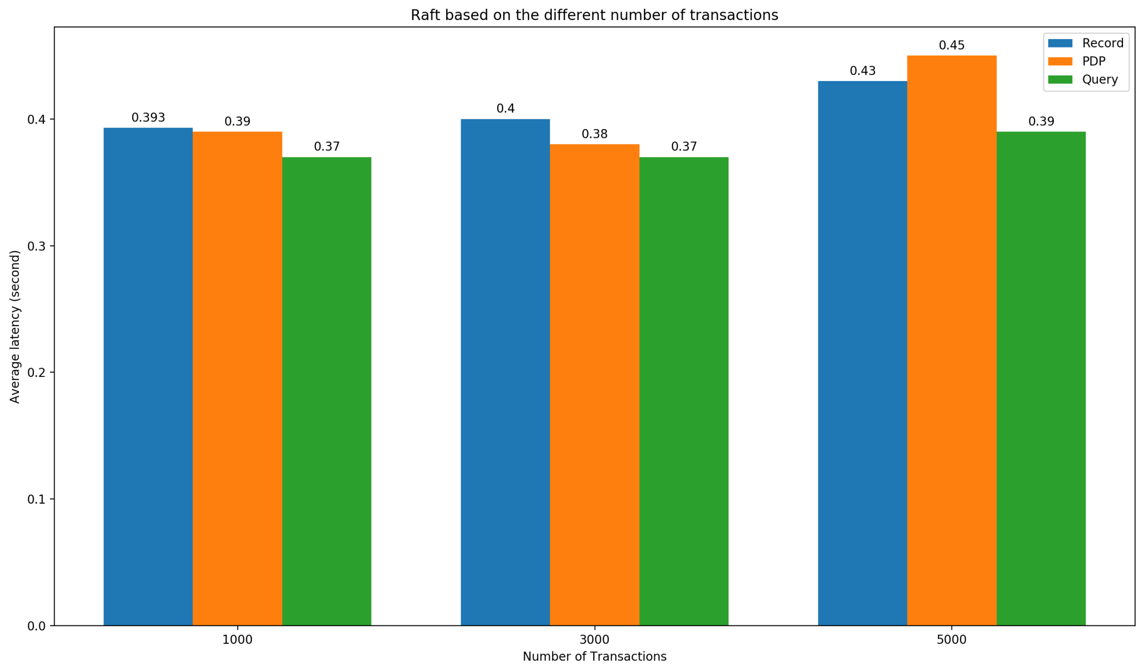


Figure 5.13: Average latency using Raft, depending on the number of issued transactions

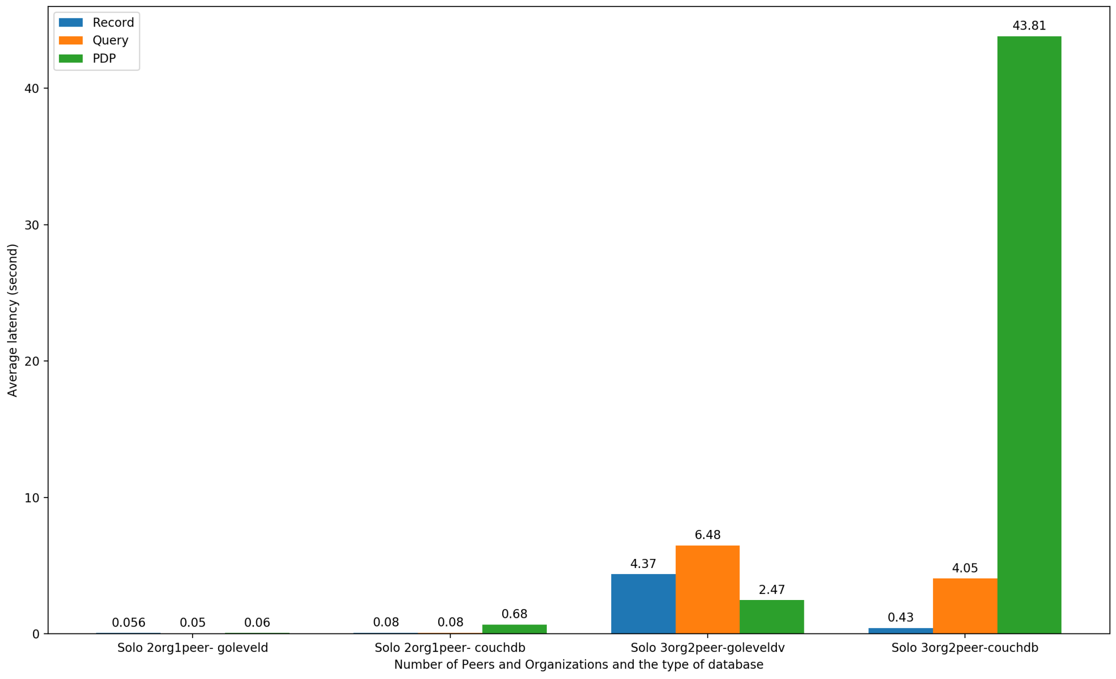


Figure 5.14: Average latency, in fuction of the world state database, number of organisation and number of peers

- Simple access control policies occupate around 1KB.
- User representations occupate 225 bytes.
- Resources occupate around 100 bytes.

The number of KB required for each access control policy to be stored is given by the following equation:

$$AccessControlPolicy_{bytes} = 1 \times N_{policies} \tag{5.4}$$

$N_{policies}$ is the number of access control policies that the system has. Assuming $N_{policies} =$ 1000, our system needs approximately 1MB to store all policies, per peer node. Assuming 1KB user representations, and taking into account that Citius holds approximately 10,000 users, the storage for users, for each peer is around 10MB. Finally, assuming each resource is defined by a pointer to its original location, each resource demands very little storage. Assuming one million resources, the storage yielded is around 100 MB.

The total storage occupied by definitions of access control policies, subjects and resources is approximately 111 MB, per peer node.

5.3.4 Discussion

There are important considerations to do with regard to JusticeChain v2. One can observe that generally, the PDP chaincode takes the longest to execute, being it more notorious for a high number of transactions (above 5,000). This result is expected, as the PDP has to retrive the subject, and access control components to evaluate the request. The query chaincode is usually the quickest to execute.

We can conclude that using CouchDB dramatically increases the average latency, in comparison with LevelDB. Furthermore, introducing a new organisation increases the average latency from 5 to 130 times. Including more than 2 organisations in the process of evaluation access control policies might be a bottleneck for the system.

JusticeChain v2 evaluation has some limitations. First, the evaluation was made taken into account a simple, general case. In order to obtain reliable results, one should be able to access IGFEJ's access control policies, definitions of subjects and rules, which we cannot, for privacy reasons. As access control policies are most likely more complex than the ones we defined the execution time taken for validating an access request to an object might be higher in a real world scenario. Concerning storage, access control policies can be occupying more storage.

This blockchain-based access control mechanism is more expensive than conventional ones. As it is not centralized, it has a higher latency and lower throughput. Nonetheless, this is a reasonable trade-off, as our method provides trust distribution and decentralized access control decisions.

The usage of a blockchain-based access control system required information systems to change their PEPs. Developers have, therefore, to make the PEP a client of the blockchain, which allows the blockchain to mediate the access controlflow. Besides building the new PEP, adminstrators have to record the access control policies, subjects and resources in the JSON format, and load them into the blockchain.

Finally, some privacy concerns may arise between participants. Should different organisations have access to each others' access control policies? In case that such consortium needs privacy, the private data feature from Fabric can be used. Alternatively, one can use zero-knowledge proof (ZKP) technologies in Hyperledger Fabric. ZKPs accommodate privacy-preserving asset management, with audit support.

5.4 JusticeChain vs JusticeChain v2

This section summarizes how JusticeChain and JusticeChain v2 work, and their main contributions to the justice ecosystem.

In JusticeChain, end-users (as judges and court clerk) utilize the information system from justice that manages court processes (i.e., Citius). When users provide a request through the PEP, it redirects the request to the local, centralized access control system, based on RBAC. The local PDP then enforces access control requests from users and may record actions performed by users on objects. Some actions from the users are stored (i.e., via a log entry), in particular, the ones explicitly defined by programmers, in a database. Such database fires trigger when a log is stored, and the information is redirected to the JusticeChain Client (Log Manager), by the oracle. Logs are parsed and processed, if needed, and sent to the blockchain on behalf of the information system that originated them.

The blockchain provides data integrity. Moreover, the blockchain provides accountability in two ways: i) towards end-users, as the files containing their actions are immutable and non-deletable, and ii) concerning who access log entries. Figure 5.15 represents JusticeChain.

Retrieving Table 2.2, one can verify that JusticeChain aims to all the desired characteristics we analysed regarding audit logs: integrity checking, integrity assurance, a decentralized solution, which keeps data private, and provides different right access to data.

In JusticeChain v2, the access control system is decentralized. For this, each PEP of the information system participating in the network serves as a blockchain client. Instead of authorizing access control requests locally, recurring to hardcoded rules and user attributes from a

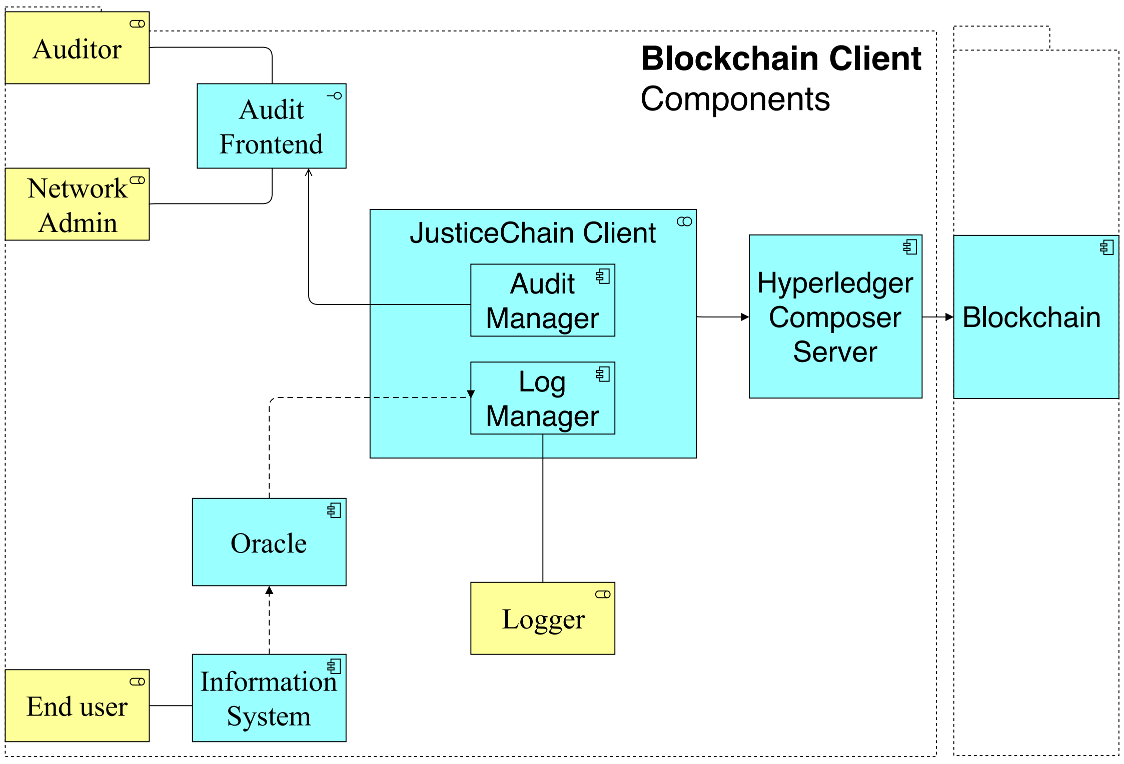


Figure 5.15: JusticeChain v1

centralized database, the PEP sends the request to the blockchain, which acts as PIP and PDP, decentralizing the flow. The blockchain processes the request based on smart policies, stored on the blockchain and from information retrieved by the PIPs. Smart policies can be created and updated at the PAP, by network administrators. The PDP enforces the decision and implicitly saves the user action by recording the transaction corresponding to an access control request on the blockchain.

JusticeChain v2 provides the same advantages as JusticeChain (data integrity, accountability towards users and accountability towards auditors) and more: it allows a fine-grain authorization control, leading to authorization decentralization. This contribute allows a more enhanced trust distribution to stakeholders, as they can check if i) their access is not being revoked unduly, and ii) only users with the right permissions can access restricted resources. Figure 5.16 represents JusticeChain v2.

Retrieving Table 2.3, one can verify that JusticeChain v2 aims to all the desired characteristics we analysed regarding blockchain access control.

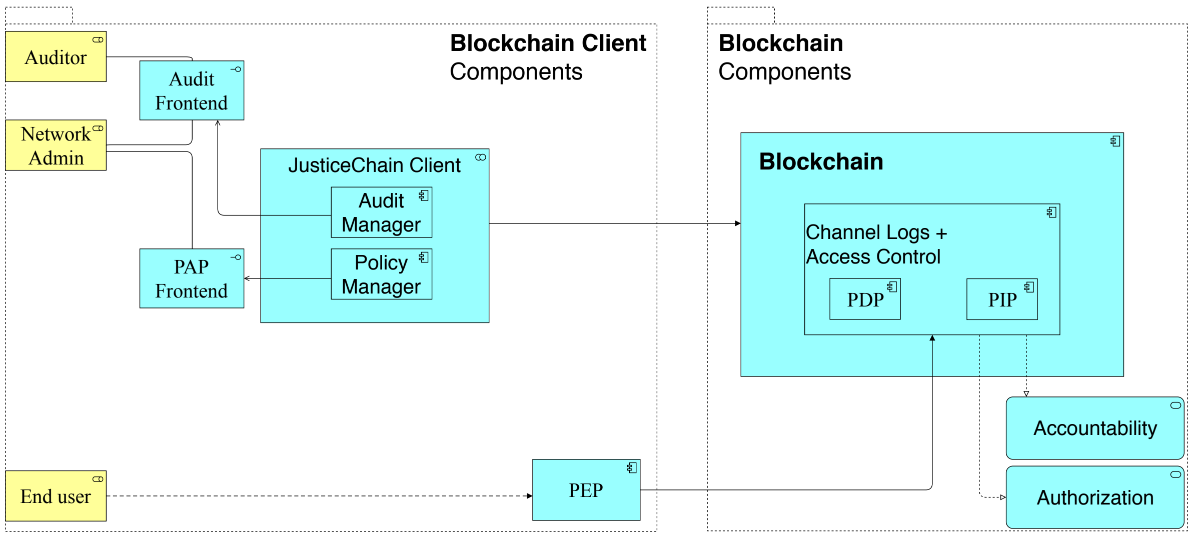


Figure 5.16: JusticeChain v2

Chapter 6

Conclusions

This dissertation presented JusticeChain, a blockchain-based system which increases trust in information systems managed by third-parties, regarding saving and accessing audit data. JusticeChain increases the resilience of applicational logs at justice, by assuring integrity, redundancy, and transparency of applicational logs, increasing their resiliency. JusticeChain improves traditional logging systems by distributing logs through the blockchain technology, where stakeholders depend on a centralised information system to conduct their activities, which cause trust issues. The experimental evaluation results show that JusticeChain can save 25 log entries per second, with a latency of 16 seconds, with a solo Logger. This option is cheaper in terms of storage, compared to the 10 Loggers alternative, that is capable of storing 37 applicational log entries per second, with a latency of under 1 minute. The evaluation showed that the storage required for each peer node is in the order of terabytes per year. Using JusticeChain to manage, inspect and analyse logs brings decentralisation and trust, at the expense of storage and performance overheads compared to traditional systems. Performance can be enhanced by tuning network parameters and utilising Fabric APIs instead of Composer's API. As a way to distribute trust on a larger scale, we propose JusticeChain v2. We proposed a blockchain-based access control system which has the potential to solve the trust and interoperability problems regarding the authorisation process. JusticeChain v2 yields the advantages that JusticeChain does (as the guarantee log integrity and audits mediated by the blockchain). Our evaluation shows that our system can achieve a throughput of around 250 transactions per second, with a latency of around 12 seconds, using the Raft consensus mechanism. This performance is suitable for systems administrated by IGFEJ, such as Citius. Our contributions can foster digital transformation at public administration, providing a framework to distribute trust and promote synergies.

6.1 Achievements

This work yields several contributions to IGFEJ, the scientific community, and the open-source community. We idealised and developed *JusticeChain*, a blockchain-based system that mediates audits and increases the resiliency of log entries, directed to the reality of IGFEJ. Furthermore, we proposed a *blockchain-based access control* system at IGFEJ, which extends JusticeChain. Three papers were produced along with the writing of this dissertation, which shows scientific validation and interest for the work developed. First, R. Belchior, M. Correia & A. Vasconcelos (2019). *JusticeChain: Using Blockchain to Protect Justice Logs*, CoopIS 2019: 27th International Conference on Cooperative Information Systems, Rhodes, Greece, Oct 21-25. Berlin, Germany: Springer Verlag, which presents JusticeChain. Second, R. Belchior, M. Correia & A. Vasconcelos (2019). Second, *Towards Secure, Distributed, and Automatic Audits with Blockchain*, submitted to the European Conference on Information Systems, which presents details about how the blockchain technology can be used to enhance audit techniques. In the context of JusticeChain v2, the author proposed a project to the Hyperledger Foundation[1], called Hyperledger Fabric Based Access Control[2]. The project is backed by the Linux Foundation[3] and it is being implemented as an official Hyperledger Lab Project[4]. This project aims to create a general case blockchain-based access control system using Hyperledger Fabric. The results of this project will be disseminated to the scientific community through the following publication: S. Rouhani, R. Belchior, R. Cruz & R. Deters (2019). Hyperledger Based Access Control, Submitted to IEEE Access.

6.2 Future Work

This work paves the way to appealing future works, as the blockchain can be explored to further enhance synergies not only in the public administration sectors but also a variety of them in which there is a lack of trust. Organisations are testing the potential of blockchain technology in areas such as data management, data privacy and access control [10, 30], bringing implications to record-keeping and the business processes associated with it, namely audits. Permissioned, private blockchain frameworks could contribute to the processes behind audits by *i)* alleviating auditor's work, *ii)* hindering fraud and collusion between organisations and auditors, *iii)* promote synergies between organisations and their stakeholders, and *iv)* protecting access to sensitive information. In particular, the combination of artificial intelligence techniques with

[1]https://www.hyperledger.org/
[2]https://wiki.hyperledger.org/display/INTERN/Hyperledger+Fabric+Based+Access+Control
[3]https://www.linuxfoundation.org/
[4]https://github.com/hyperledger-labs/hyperledger-fabric-based-access-control

the decentralised enforces execution that smart contracts provide the necessary infrastructure to semi-automatic or even automatic audits. Therefore, we aim to provide pointers to the utilisation of blockchain technology as an enhancer of secure, distributed, and more automatized audits.

Furthermore, this work can be extended to domains different than justice. Public administration in Portugal have processes in which there are stakeholders with different incentives, using third-party information systems, like the ones managing *registo predial*, or *custas judiciais* or even *electronic voting*. Any industry which has trust issues between stakeholders, and should benefit from a decentralised access mechanism to data (for provenance, transparency and traceability) can benefit from JusticeChain. From this reasoning, it follows that JusticeChain may be applied to fundamentally different areas than justice, such as healthcare, banking, smart cities, electronic identity, education, insurance, wills and inheritances, and loyalty programs. As a mean to overcome JusticeChain's limitations, different blockchains can be taken into account. Multichain 2.0 [5], provides a smart contract functionality and its optimised for read operations, which can provide an enhanced system for automatic audits. A combination of Hyperledger Fabric with Multichain could be interesting to explore. To do so, we can consider the Cosmos blockchain [6]. Cosmos is based on Tendermint [7], and aims to solve scalability and interoperability problems, by aggregating several different blockchains. The theoretical maximum throughput of Tendermint is around 14,000 transactions per second, which would leverage JusticeChain to secure records from more demanding systems and, on the other hand, providing a more robust system to auditors and automatized audits.

Concerning JusticeChain v2, we could extend the blockchain-based access control system to handle authentication, as well. By exploring an end-to-end blockchain-based AAA system, we can explore different ways of facilitating the integration of different systems with the blockchain technology. In particular, there are lots of different information systems for lots of governmental infrastructures that share no authentication, authorisation and accountability processes. Managing to interoperate such systems, we can have a holistic and hermetic vision of what happens in social structures. In particular, data mining techniques might provide insight regarding the accesses made on which structure, taking into account a broader context. This appealing direction can contribute directly to the development of society.

[5]https://www.multichain.com/white-paper/
[6]https://cosmos. network/cosmos-whitepaper.pdf
[7]https://tendermint.com

Bibliography

[1] Ashar Ahmad, Muhammad Saad, and Aziz Mohaisen. Secure and transparent audit logs with BlockAudit. *Journal of Network and Computer Applications*, 145(1), 2019.

[2] Jessie Anderson and Sean Smith. *Securing, standardizing, and simplifying electronic health record audit logs through permissions blockchain technology*. PhD thesis, Dartmouth College, 2018.

[3] Elli Androulaki, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, and Et al. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. *Proceedings of the Thirteenth EuroSys Conference*, 2018.

[4] Mihir Bellare and Bennet S Yee. Forward Integrity For Secure Audit Logs. Technical report, University of California at San Diego, 1997.

[5] P Bharathi and S Rajashree. Secure file access solution for public cloud storage. In *International Conference on Information Communication and Embedded Systems*, pages 1–5, 2014.

[6] Saikat Biswas, M Sohel, Md.Mizanur Sajal, Tanjina Afrin, Touhid Bhuiyan, and Md Maruf Hassan. A Study on Remote Code Execution Vulnerability in Web Applications. In *International Conference on Cyber Security and Computer Science*, 2018.

[7] Fernando Boavida, Mário Bernardes, and Pedro Vapi. *Componentes AAA*, pages 190–206. FCA - Editora de Informática, Lda., 2011.

[8] Jasper Bogaerts, Maarten Decat, Bert Lagaisse, and Wouter Joosen. Entity-Based Access Control: Supporting More Expressive Access Control Policies. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 291–300, New York, NY, USA, 2015.

[9] Amy Bruckman. *Analysis of Log File Data to Understand Behavior and Learning in an Online Community*, pages 1449–1465. 2006.

[10] Fran Casino, Thomas K Dasaklis, and Constantinos Patsakis. A systematic literature review of blockchain-based applications: Current status, classification and open issues. *Telematics and Informatics*, 36:55–81, 2019.

[11] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems*, 20:398–461, 2002.

[12] Zhiwei Chen, Yatao Yang, Ruoqing Zhang, and Zichen Li. An efficient scheme for log integrity check in security monitoring system. In *IET Conference Publications*, pages 246–250, 2013.

[13] Fred Cohen. A cryptographic checksum for integrity protection. *Computers & Security*, 6: 505–510, 1987.

[14] Mauro Conti, E Sandeep Kumar, Chhagan Lal, and Sushmita Ruj. A Survey on Security and Privacy Issues of Bitcoin. *IEEE Communications Surveys & Tutorials*, 20:3416–3452, 2017.

[15] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On Scaling Decentralized Blockchains. In *Bitcoin and Blockchain*, volume 9604, pages 106–125, 2016.

[16] Jordi Cucurull and Jordi Puiggalí. Distributed Immutabilization of Secure Logs. In Gilles Barthe, Evangelos Markatos, and Pierangela Samarati, editors, *Security and Trust Management*, pages 122–137, Cham, 2016.

[17] Whitfield Diffie and Martin E Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[18] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. Blockbench: A framework for analyzing private blockchains. In *ACM International Conference on Management of Data*, SIGMOD '17, pages 1085–1100, New York, NY, USA, 2017.

[19] D Ferraiolo, R Sandhu, S Gavrila, D.Kuhn, and R Chandramouli. A Proposed Standard for Role-Based Access Control. *ACM Transactions on Information and System Security*, 4 (3), 2001.

[20] David Ferraiolo and Richard Kuhn. Role-Based Access Control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[21] Edoardo Gaetani, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. Blockchain-Based Database to Ensure Data Integrity in Cloud Computing Environments. In *The Italian Conference on CyberSecurity*, 2017.

[22] Morrie Gasser. *Building a Secure Computer System.* 1988.

[23] Rainer Gerhards. The Syslog Protocol. RFC 5424, RFC Editor, 2009.

[24] Trent Jaeger. Reference Monitor. In Henk C A van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security, 2nd Ed.*, pages 1038–1040. 2011.

[25] Abdul Raouf Khan. Access Control in Cloud Computing Environment. *ARPN Journal of Engineering and Applied Sciences*, 7:613–615, 2012.

[26] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka : a Distributed Messaging System for Log Processing. In *NetDB*, pages 1–7, 2011.

[27] Hendrik Leppelsack, Holger Kinkelin, Stefan Liebald, and Fabian Geyer. *Experimental Performance Evaluation of Private Distributed Ledger Implementations.* PhD thesis, Technical University of Munich, 2018.

[28] Di Ma and Gene Tsudik. A New Approach to Secure Logging. *Data and Applications Security XXII*, 5(1):2:1—-2:21, 2009.

[29] Damiano Maesa, Di Francesco, Paolo Mori, and Laura Ricci. Blockchain based access control. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 206–220. Springer, 2017.

[30] Damiano Maesa, Paolo Mori, and Laura Ricci. Blockchain based access control services. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1379–1386, 2018.

[31] Damiano Maesa, Paolo Mori, and Laura Ricci. Blockchain based access control services. In *The 2018 IEEE International Conference on Blockchain*, 2018.

[32] Damiano Maesa, Paolo Mori, and Laura Ricci. A blockchain based approach for the definition of auditable Access Control systems. *Computers & Security*, 84, 2019. doi: 10.1016/j.cose.2019.03.016.

[33] Jonathan Margulies. A Developer's Guide to Audit Logging. *IEEE Security Privacy*, 13 (3):84–86, 2015.

[34] Laurent Maryline, Kaaniche Nesrine, and Le Christian. A Blockchain based Access Control Scheme. In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications*, pages 168–176, 2018.

[35] National Computer Security Center. Trusted Computer Systems Evaluation Criteria, 1983.

[36] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. In *USENIX Conference on USENIX Annual Technical Conference*, pages 305–320, 2014.

[37] Charles P Pfleeger, Shari Lawrence Pfleeger, and Jonathan Margulies. *Security in Computing (5th Edition)*. Upper Saddle River, NJ, USA, 5th edition, 2015.

[38] William Pourmajidi and Andriy Miranskyy. Logchain: blockchain-assisted log storage. In *IEEE 11th International Conference on Cloud Computing*, pages 978–982, 2018.

[39] William Pourmajidi, Andriy Miranskyy, Whitfield Diffie, Martin E Hellman, R S Sandhu, P Samarati, and Fred Cohen. New Directions in Cryptography. *Computers & Security*, 32 (6):644–654, 2018.

[40] Indrajit Ray, Kirill Belyaev, Mikhail Strizhov, Diedonne Mulamba, and Mariapann Rajaram. Secure logging as a service - delegating log management to the cloud. *IEEE Systems Journal*, 7:323–334, 2013.

[41] Sara Rouhani, Vahid Pourheidari, and Ralph Deters. Physical Access Control Management System Based on Permissioned Blockchain. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2019.

[42] Ravi S Sandhu and Pierangela Samarati. Access control: principle and practice. *IEEE Communications*, 32(9):40–48, 1994.

[43] Bruce Schneier and John Kelsey. Cryptographic Support for Secure Logs on Untrusted Machines. In *7th Conference on USENIX Security Symposium*, 1998.

[44] Bruce Schneier and John Kelsey. Secure Audit Logs to Support Computer Forensics. *ACM Transactions on Information and System Security*, 2(2):159–176, 1999.

[45] Richard T Snodgrass, Shilong Stanley Yao, and Christian Collberg. Tamper Detection in Audit Logs. In *13th International Conference on Very Large Data Bases*, VLDB '04, pages 504–515, 2004.

[46] Andrew Sutton and Reza Samavi. Blockchain Enabled Privacy Audit Logs. In *The Semantic Web – ISWC 2017: 16th International Semantic Web Conference*, pages 645–660, 2017.

[47] Paul J Taylor, Tooska Dargahi, Ali Dehghantanha, Reza M Parizi, and Kim-Kwang Raymond Choo. A systematic literature review of blockchain cyber security. *Digital Communications and Networks*, 2019.

[48] Parth Thakkar, Senthil Nathan, and Balaji Viswanathan. Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform. In *IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 264–276, 2018.

[49] TO Group. *ArchiMate®3.0 Specification.* Van Haren Publishing, 2016. ISBN 9789401806305.

[50] Uchi Ugobame Uchibeke, Sara Hosseinzadeh Kassani, Kevin A Schneider, and Ralph Deters. Blockchain access control Ecosystem for Big Data security. *Green Computing and Communications (GreenCom), IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing*, 2018.

[51] Marco Vieira and Henrique Madeira. Benchmarking the dependability of different OLTP systems. In *International Conference on Dependable Systems and Networks*, pages 305–310, 2003.

[52] Marko Vukolić. Rethinking Permissioned Blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, pages 3–7, 2017.

[53] Hai Wang, Yong Wang, Zigang Cao, Zhen Li, and Gang Xiong. An Overview of Blockchain Security Analysis. In *Cyber Security*, pages 55–72, 2019.

[54] Daniel J Weitzner, Harold Abelson, Tim Berners-Lee, Joan Feigenbaum, James Hendler, and Gerald Jay Sussman. Information Accountability. *Communications of the ACM*, 51 (6):82–87, 2008.

[55] Matthias Wiesmann, Fernando Pedone, André Schiper, Bettina Kemme, and Gustavo Alonso. Database replication techniques: A three parameter classification. In *9th IEEE Symposium on Reliable Distributed Systems*, pages 206–215, 2000.

[56] Xiwei Xu, Cesare Pautasso, Liming Zhu, Vincent Gramoli, Alexander Ponomarev, and Shiping Chen. The Blockchain as a Software Connector. In *13th Working IEEE/IFIP Conference on Software Architecture*, pages 182–191, 2016.

[57] Congcong Ye, Guoqiang Li, Hongming Cai, Yonggen Gu, and Akira Fukuda. Analysis of Security in Blockchain: Case Study in 51%-Attack Detecting. In *5th International Conference on Dependable Systems and Their Applications*, pages 15–24, 2018.

[58] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town Crier: An Authenticated Data Feed for Smart Contracts. In *ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 270–282, New York, NY, USA, 2016.

[59] Yuanyu Zhang, Shoji Kasahara, Yulong Shen, Xiaohonh Jiang, and Jianxiong Wan. Smart Contract-Based Access Control for the Internet of Things. *IEEE Internet of Things Journal*, 6(2):1594–1605, 2019.

[60] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In *IEEE International Congress on Big Data*, 2017.

[61] A Zúquete. *Segurança em Redes Informáticas, 4a Edição Actualizada e Aumentada.* FCA - Editora de Informática, Lda., 2013.

[62] G Zyskind, O Nathan, and A. Pentland. Decentralizing Privacy: Using Blockchain to Protect Personal Data. In *IEEE Security and Privacy Workshops*, pages 180–184, 2015.