UNIVERSITY
OF TWENTE.

REDACTABLE BLOCKCHAIN

How to change the immutable and the consequences of doing so

DAMIANO SARTORI

MSc EIT - Cyber Security
University of Twente

August 2020

Damiano Sartori: *Redactable Blockchain,* How to change the immutable and the consequences of doing so, © August 2020

SUPERVISOR:
Dr. Maarten Everts

COMMITTEE MEMBERS:
Dr. Luís Ferreira Pires
Dr. Ansgar Fehnker

LOCATION:
Enschede

TIME FRAME:
August 2020

ABSTRACT

A blockchain is a peer-to-peer distributed ledger that registers cryptographically signed transactions in a sequence of blocks. Each block in the chain stores the hash of the previous block, thus creating a chain of blocks. Blockchain is thought to be immutable thanks to the properties provided by the hash function. More precisely, a blockchain can be described as a tamper-proof and tamper-evident chain of blocks.

The immutability of a blockchain is undoubtedly one of its strongest features. However, the inability to change or delete data might be an undesirable feature in specific contexts and represents another challenge in the use of blockchain in those situations in which personal data are at stake. Art. 16 and Art. 17 General Data Protection Regulation (GDPR), introducing the data subject's right to rectification and right to erasure, assumes that modifications and deletion of data are always possible. Therefore, there might be situations in which the actual deletion (or change) of data is mandatory, and the inability of doing so will result in a non-compliant system.

To facilitate the possibility of compliance, we propose and design the architecture of a blockchain that allows for modifications and deletions of data under particular circumstances. We employ chameleon-hash functions with ephemeral trapdoor as a substitute for the standard hash functions used in the blockchain. The ephemeral trapdoor is different for each newly created hash, which allows for targeted and fine-grained collision computation.

We distribute the ephemeral trapdoor to the data subject, the data controller, and the data processors using a verifiable and weighted secret sharing schemes in which the data subject holds the strongest share of the trapdoor. However, in case the data subject loses the key or is unwilling to engage in the protocol, the shares distributed to the other parties allow for the reconstruction of the ephemeral trapdoor. To maintain a sufficient level of integrity and keep the tamper-evident property of a blockchain, we publish a Proof-of-Redaction. This mechanism serves to prove that history has been modified and that the network agreed on the redaction.

We evaluate our proposal with blockchain and cryptography experts to validate our design. We show that, while the standard immutability is not maintained, a weaker version that accounts for authorized redactions is still achievable. The proposed architecture could have the ability to reduce the frictions between the immutability of a blockchain and the GDPR without improperly weakening an existing architecture.

ACKNOWLEDGEMENTS

First and foremost, I wish to express my sincere gratitude to my university supervisor, dr. Maarten Everts, and to my supervisor at the company hosting the research. You challenged me since the beginning of the thesis, helped me to shape my research, and guided me with your precious comments.

Many thanks go to colleagues and interns in Deloitte. You gave me the opportunity to make the most of out of this experience even if we enjoyed staying together for less than two months. I appreciated the moments I spent with you and I am looking forward to reconnecting together. In particular, I would like to recognize the support received from the thesis coordinators in Deloitte and all the colleagues who spent their time to provide valuable comments on the project.

I would also like to thank friends and classmates with whom I spent these two years. It would be too long to mention you all but I am sure you will recognize yourself as part of those. It was the first time I lived in a foreign country for a long time and you all contributed to this unforgettable adventure.

From the bottom of my heart, a special thank goes to Virginia. Even though we spent almost a year apart, I have always felt loved and supported during this time. I consider myself lucky for having the chance of spending the last four years together and I cannot wait to see what is up next.

Last, but not least and less important, thanks to my parents and my brothers. They supported during this process and I recognize I would have never come this far without you.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

CHET	Chameleon-Hash with Ephemeral Trapdoor
CP-ABE	Ciphertext-Policy Attribute-Based Encryption
DLT	Distributed Ledger Technologies
DPA	Data Protection Authority
DPO	Data Protection Officer
GDPR	General Data Protection Regulation
PBTF	Practical Byzantine Fault Tolerance
PBH	Policy-Based Chamaleon Hash
PoA	Proof of Authority
PoET	Proof of Elapsed Time
PoR	Proof of Redaction
PoS	Proof of Stake
PoW	Proof of Work
RR	Round Robin

1 INTRODUCTION

Looking back to the last half-century of computer history, we may recognise a slow but continuous movement towards a decentralised computing paradigm. Initially, mainframes were centralised, and they were hosting memory, data, and computing power. Access to those resources was performed via very simple terminals that contained little or no memory and computing resources. Years later, personal computers started to gain popularity and computing resources began their movement from a single centralised location to user's laptops. Significant computational power was still hosted by the servers and accessed by the clients, and access to data was still mainly centralised. The client-server architecture represents the first step of the decentralisation process. More recently, Internet and cloud computing enable wider - and almost global - access to data from a huge range of devices, from smartphones to sensors integrated into everyday objects. Nowadays, the decentralisation process is pushed by emerging technologies such as the blockchain.

A blockchain is a peer-to-peer distributed ledger that registers cryptographically signed transactions in a sequence of blocks. Its very first well-known application is dated back in 2008 when Satoshi Nakamoto introduced Bitcoin [1]. Bitcoin is a peer-to-peer electronic cash system developed to reduce the need to rely on a centralised third party to regulate financial transactions and give users control over their operations. It solved the problem of double-spending by using a distributed timestamp system to create a list of chronologically ordered transactions. Since the first transaction in 2009, Bitcoin gained popularity and paved the way for the creation of hundreds of cryptocurrencies in the last decade. Besides the hype in cryptocurrencies, the technology supporting Bitcoin has steadily gained interest. New applications of blockchain outside the financial and payment systems were and are being proposed. For instance, Internet of Things (IoT), public and social services, reputation systems, supply chain management, provenance, and healthcare have all been identified as potential sectors in which blockchain can provide added value.

One of the goals of decentralised systems is to give end-users additional control over their digital assets and their data, removing the trusted middleman. Additional control over a user's data is also one of the goals of the General Data Protection Regulation (GDPR). Adopted in 2016 and enforceable since May 2018, GDPR aims at simplifying the fragmented European regulatory environment concerning data protection and giving more control to individuals over their data. Although blockchain and GDPR have the common goal of empowering individuals and increasing the control they have other their data, some fundamental characteristics of a blockchain are in contrast with the rights identified by the GDPR. On the one side, blockchain is thought to be immutable[1] because, once a transaction has been approved and a block has been added to the chain, it is almost impossible to modify the content of that transaction without disrupting the chain. Modification of data will generate a different hash for that transaction, invalidating the block and all subsequent ones. On the other side, GDPR recognises the rights of individuals to request the deletion or the modification of their data. Once an authroity grants the request, data need to be deleted or modified accordingly, no matter the technology used to store and manage it.

1.1 MOTIVATION

The immutability of a blockchain is undoubtedly one of its strongest features. Even though it is usually referred to as immutability, we should specify that it is not absolute property in the sense that a blockchain *could* be modified. However, it is extremely difficult to modify it, and every modification is evident because it alters its structure. Hence, to be precise, we should refer to a blockchain as a tamper-proof and tamper-evident structure.

In most scenarios, a tamper-proof and tamper-evident structure perfectly fulfils the goal of maintaining an *immutable* log of transactions where parties do not trust each other and do not want to engage in a trust relationship with a third-party that supervises and grants the integrity of the transaction process. However, this feature presents itself in contrast with other requirements in some specific scenarios. For instance, when material that infringes copyright law or when personal data is posted on a blockchain.

To prevent users involved in the consensus algorithm and participants making use of the blockchain to conduct transactions from being liable of infringing laws or regulations, it might be useful to alter the content of the blockchain so that it is possible to remove unwanted and illegal material.

1.2 PROBLEM STATEMENT

The presence of unwanted or illegal material on a blockchain might be detrimental for the participants of the network. In cases where un-

1 To be precise, a blockchain does not achieve perfect immutability. A blockchain could be modified. However, it is challenging to do so because every modification to existing data alters the hash chain and requires to recompute the list of all hashes. Moreover, the fact that changed data modify the hash makes every modification visible to the other participants.

wanted content infringes laws or regulations, it should be possible to remove part of the content from the blockchain so that the network can operate in a compliant fashion. A very naive way of modifying the blockchain is making use of hard forks. However, hard forks require an off-chain agreement among the developers of a blockchain. All the confirmed transactions that have been removed due to the fork must be executed again. Another naive approach consists of pruning the blockchain to remove all blocks older than a specific date. However, we should note that pruning reduces the size of a blockchain, and it is not explicitly thought to remove unwanted content. We argue that it is worth to analyse the problem of modifying a blockchain more smartly, allowing for finer-grained modification targeted to remove unwanted content. Moreover, we believe such change should be evident and justified so that integrity and immutability of the blockchain can be maintained to a sufficient level to justify the use of a blockchain even when its modification is permitted.

1.3 OBJECTIVES

The first objective of this thesis is to define in which cases we should permit modification on a blockchain to comply with the General Data Protection Regulation (GDPR), hereinafter referred to as the Regulation. To achieve this goal, we discuss if and in which circumstances transactional data, public keys, and hashes should be considered personal data to determine whether the GDPR applies. We stress that this discussion is missing in most of the proposed solutions in the literature, which assume that modifications are necessary without providing a sufficient justification.

The second objective is to propose an architecture that reduces the contrasts between the Regulation and the way a blockchain manages and processes data. With the ultimate goal of simplifying the development of a compliant blockchain application, we analyse the existing frictions to define the requirements that our design should fulfil.

The third objective is to define who should have the rights to propose and approve modifications. While this definition depends on the particular application, it is nonetheless helpful to present our design and to facilitate its integration in a defined use-case.

Last, the fourth objective is to formalise some properties of a blockchain, namely integrity and immutability, and to examine to which extent these properties are weakened if we introduce the ability to modify the ledger.

1.4 RESEARCH QUESTIONS

To achieve the goals announced in the previous section, we formulate the following research questions:

RQ1: Should we modify blockchain technology so that it is possible to alter or delete transactions to comply with Art. 16 and Art. 17 of GDPR?

- *SQ1:* What obstacles are introduced by Art. 16 and Art. 17 of GDPR in the processing of personal data in a blockchain?
- *SQ2:* What are the requirements to build a compliant blockchain system?
- *SQ3:* Is the modification of the blockchain a possible way to comply with the Regulation?
- *SQ4:* Which technical building blocks should we leverage to produce a design that facilitates compliance?
- *SQ5:* In case changes are needed, who has the right to propose and approve modifications?

RQ2: How does the modification of the blockchain impact its properties?

- *SQ1:* To what extent the integrity of blockchain suffers from this modification?
- *SQ2:* To what extent the immutability of the ledger suffers from this modification?

1.5 METHODOLOGY

To answer our research questions, we adopt the *Design Science Research* methodology. Precisely, we refer to the methodologies presented by Hevner et al. in [2] and by Vaishnavi and Kuechler in [3]. The methodology revolves around a problem that can be solved by designing an artefact. Following the design, the artefact can be implemented, tested, and evaluated to reflect on whether or not the problem has been solved. According to the methodology presented in [3], our process is structured in five different phases:

Phase 1: **Awareness of the problem**. The awareness of the problem comes from multiple sources, including industry developments or a reference discipline [3]. In this situation, the specific problem we are investigating is the incompatibility of GDPR requirements and blockchain immutability. The problem comes from the tentative solution of applying blockchain to solve the issue of sharing data among organizations in an environment with limited trust. The output of the awareness phase was our project proposal.

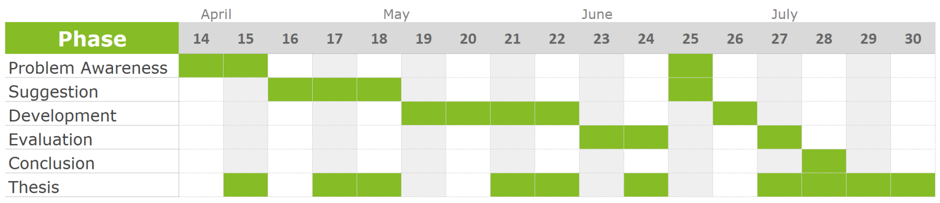
Phase 2: **Suggestion**. The suggestion phase uses as input the project proposal to envision new and creative configurations of the system with the potential of solving the problem [3]. In our research, this phase provided us with the design of a system able to overcome the limitation of current proposals.

Phase 3: **Development**. The development phase includes the implementation of the proposed design [3]. Depending on the artefact, the development output ranges from formal proofs to software development or reference architectures. In our project, the final artefact to be created has been based on whether a component needs to be added to the blockchain or modified from an existing architecture.

Phase 4: **Evaluation**. Once the artefact has been implemented, it is evaluated in the evaluation phase using implicit and/or explicit evaluation criteria [3]. The deviations of the system from the expected outcome "must be tentatively explained" [3] with the development of hypothesis to justify the unexpected behaviour.

Phase 5: **Conclusion**. The conclusion phase ends the research cycle and includes a strong communication component [3]. In case of a successfully implemented artefact, the conclusion phase presents a new tool that can be later applied to solve the identified problem. On the contrary, in case the artefact shows anomalous behaviour, the conclusion phase proposes a possible explanation and drives future research.

1.6 TIMELINE



	April				May					June				July			
Phase	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Problem Awareness	🟩	🟩										🟩					
Suggestion			🟩	🟩	🟩							🟩					
Development						🟩	🟩	🟩	🟩				🟩				
Evaluation										🟩	🟩			🟩			
Conclusion															🟩		
Thesis		🟩		🟩	🟩			🟩	🟩		🟩			🟩	🟩	🟩	🟩

Figure 1: Timeline of the Thesis Project

Figure 1 illustrates the phases of the thesis project. The phases include the following tasks:

Phase 1: **Awareness of the problem**. To build a theoretical awareness of the problem, we performed a focused review on redactable blockchain. We identified the limitations of the proposed solutions, as well as compliance and technical requirements. This phase is based on unstructured interviews carried out with legal and technical experts of the company hosting the research, on the literature review of the research topic, and on a set of documents suggested by the experts that address the conflicts between GPDR and blockchain technology. The goal is to develop theoretical knowledge of the problem, to gather a set of preliminary requirements and evaluate whether current solutions meet the requirements, and to state a list of assumptions to drive the design of the architecture.

Phase 2: **Suggestion**. The suggestion phase includes the selection of tools, technologies, and cryptographic primitives that have been used in the development phase. The output of the phase is the preliminary design based on the assessment of state-of-the-art approaches to modifiable blockchain. The design identified in this phase has been discussed with the company's experts to validate that requirements are theoretically satisfied in the design.

Phase 3: **Development**. The development phase consisted of the development of a reference architecture based on the design proposed in the previous phase. According to [3], an architecture is a "high level structure of systems". While we do provide a high-level design of the system, we do also provide a detailed discussion on the various building blocks that compose the system. The development phase included the implementation of some fundamental building blocks to provide performance evaluation in the following phase.

Phase 4: **Evaluation**. The evaluation phase includes a compliance check with the legal requirements, the evaluation of the impact on integrity and immutability, and the testing of the proof of concept of the implemented building blocks.The design science research methodology includes a continuous evaluation through "micro-evaluations" [3] performed by the designer throughout the whole design process. The concluding formal evaluation has been performed by using explicit state-of-the-art methods used in related works of redactable blockchain as well as semi-structured interviews with experts to check the impact of our work on some key properties that are discussed in Chapter 6.

Phase 5: **Conclusion**: The conclusion phase includes a summary of the research, the answer to the research questions, the discussion of the limitations and a proposal for future developments. Its goal is to communicate and summarise the research findings and to discuss various possible directions to improve the existing architecture and to develop a working proof-of-concept.

1.7 CONTRIBUTIONS

The contributions of this work are threefold:

1. Provide a legal discussion on whether content on a blockchain might be subject to GDPR requirements due to its classification as personal data.

2. Propose a design that allows for the modification of a blockchain to comply with the Regulation. The main contributions of the design are the involvement of the data subject into the modification process through the use of secret sharing and the introduction of proof of redaction to the ledger. Compared to existing work, the novelty of our approach is noticeable both in the way we distribute the trapdoors and in the presence of a Proof-of-Redaction that shows the ledger was modified.

3. Analyse and evaluate to what extent some key properties of a blockchain are weakened due to our modification and provide a performance evaluation of the hash function.

1.8 STRUCTURE

This document is further structured as follows. Chapter 2 provides background information on blockchain technology and Hyperledger Fabric. Chapter 3 presents the findings of the literature review and introduces a categorization of the proposed solutions. Chapter 4 builds the awareness of the problem from the legal perspective and identifies requirements for the design phase. Chapter 5 constitutes the main body of the research and presents the reference architecture as well as the integration into an existing blockchain. It also provides an implementation of the main building block of our solution to show the feasibility of our approach. Following the design, Chapter 6 evaluates the research through experts interviews to identify whether we reached our objectives and discusses the impact of such modification on a blockchain architecture. Last, Chapter 7 summarises the main findings, discusses the limitations of our approach and provides directions for future research.

2 BACKGROUND

The core ideas behind blockchain can be traced back to the late 1980s and the early 1990s [4]. In 1989, Lamport proposed Paxos, a consensus protocol to reach agreement in a distributed environment where the network might be unreliable [5]. In 1991, Haber and Stornetta introduced a procedure to certify the moment in which a digital document was created or modified by using a signed chain of information as a ledger [6]. In the early 2000s, Mazières and Shasha developed a block-based data structure and protocol for a multi-user file system that demonstrated the ability of a block to store data. In 2005, Szabo came up with an early attempt to build a decentralized currency to move control from a single and centralized entity to various smaller entities [8]. All these steps paved the way for the development of Bitcoin, the peer-to-peer electronic cash system proposed by an unidentified person or group of people under the name of Satoshi Nakamoto [1]. Bitcoin solved the problem of double-spending by using a distributed timestamp system that allows the creation of a timestamped and chronologically ordered list of transactions. Since the creation of Bitcoin, a steadily-growing interest around cryptocurrencies began. More recently, researchers showed interest in the technology supporting Bitcoin, i.e., the blockchain, and its applications outside the financial and payment systems.

This chapter provides background information on blockchain and distributed ledger technologies. In particular, Section 2.1 gives a brief overview of distributed ledger technologies and its components. Section 2.2 describes the core components of a blockchain with an introduction of the cryptographic primitives and the record-keeping elements. Last, Section 2.3 presents Hyperledger Fabric, an open-source consortium blockchain project hosted by the Linux Foundation. We use Hyperledger Fabric in Chapter 5 to show the integration of our design into an existing blockchain architecture.

2.1 DISTRIBUTED LEDGER TECHNOLOGY

A **distributed ledger** is a database that is synchronized and distributed across multiple devices and generally spread around different geographical sites and institutions. Distributed Ledger Technologies (DLT) is a system based on distributed ledgers, which needs a peer-to-peer network of interconnected devices, called nodes, and a consensus algorithm that allows the modification of the ledger correctly and consistently. A distributed ledger usually has the following model [4]:

- all participants share a consistent copy of the database, there is no central server, and optionally, some participants might not have a full copy;
- network connections are peer-to-peer;
- participants must comply with ledger rules;
- to agree on the validity of a given transaction, participants use a consensus protocol;
- transactions could be financial or exchanging of assets and rules for the transaction could be coded in smart contracts;
- digital signatures are used to sign transactions on the ledger;
- the ledger represents a temporal order of how assets evolve.

2.2 BLOCKCHAIN

Broadly, a blockchain can be seen as a distributed data structure similar to a peer-to-peer database that records transactions in a ledger. Anybody can propose a change to the database but only the changes approved by the other participants are considered to be valid and added to the ledger. The consensus mechanism allows participants to accept a transaction and to agree on a specific history. Due to its novelty, however, the literature lacks agreement on the concept of blockchain. It can be seen as a data model, i.e., a chain of transactions grouped into blocks, or as a technology, i.e., a type of distributed database.

More formally, a blockchain is a peer-to-peer distributed ledger that registers cryptographically signed transactions in a sequence of blocks. Each block in the chain stores the hash of the previous block, thus creating a chain of blocks. Blocks in the chain have only one parent block, and the first block is called *genesis block*. Participants in the peer-to-peer ledger are referred to as *nodes*. Every node in the network saves a copy of the ledger and, depending on the type of the blockchain, proposes and validates *transactions*, participating in the *consensus algorithm*. Blockchain might be challenging to understand as a whole. Therefore, in the following, we present the core technologies a blockchain relies on according to [9]. First, we present the cryptographic primitives that support the building blocks of a blockchain. Second, we examine the record-keeping components. Third, we present the taxonomy of existing blockchain. Last, we discuss some frameworks that allow an individual or organization to understand whether there might be the need to implement a blockchain in a particular use case.

2.2.1 *Cryptographic Hash Functions*

A **hash function** is a compression function that takes a message $\bar{x}$, represented as a string of bit of arbitrary length, and maps it into a string of fixed length y, called the *digest*. A hash function is designed to be a one-way function, meaning that it is practically infeasible to invert and the only way to find the original message is through a brute-force search of all the possible inputs. A **cryptographically secure hash function** is a hash function $h()$ that satisfy the following three properties [9]:

1. **Pre-image resistance**. A hash function $h()$ is said to be *pre-image resistant* if, given a digest y, it is computationally infeasible to find $\bar{x}$ such that $y = h(\bar{x})$.

2. **Second pre-image resistance**. A hash function $h()$ is said to be *second pre-image resistant* if, given a digest y and a message $\bar{x}$ such that $y = h(\bar{x})$, it computationally infeasible to find another message $\hat{x} \neq \bar{x}$ such that $y = h(\hat{x})$.

3. **Collision resistance**. A hash function $h()$ is said to be *collision resistant* if it is computationally infeasible to find two messages $\bar{x}$ and $\hat{x}$, $\hat{x} \neq \bar{x}$ such that $h(\hat{x}) = h(\bar{x})$.

The use of cryptographically secure hash functions in a blockchain varies from the creation of unique identifiers to securing and connecting block of data [9]. Blocks of data are linked through hash pointers, a cryptographic hash pointing to the location in which data is stored, i. e., the previous block in the chain. Hash pointers can be used to verify whether a block has been tampered with thus ensuring the integrity of data [10].

2.2.2 *Digital Signatures*

Digital signature schemes are made of three components [10]. The first is the key generation algorithm, which creates a pair of keys. To sign a message, a signer uses its private key - which should remain secret - and the signature can be later verified with the public key. The second component is the signing algorithm. The **digital signing algorithm** takes a digest of a message $h(\bar{x})$, the private key of the signer sk, and a random quantity, to produce a **signature** s. Once a party receives the signature, a verification algorithm (the third core component) checks its validity. A **verification algorithm** takes a message $\bar{x}$, the signature s, and the public key pk of the sender to check whether the signature is valid. Digital signature algorithm's goals are authentication, non-repudiation, and integrity.

2.2.3 *Asymmetric-Key Cryptography*

Asymmetric-Key Cryptographyy (also known as **Public-Key Cryptography**) includes the cryptographic algorithms that make use of a pair of keys: a public key and a private key [9]. The two keys are mathematically related, but it must be infeasible to derive the private key starting from the public key. The **public key** can be revealed to the public without hindering the security of the algorithm. On the contrary, the private key must be kept secret. The two keys are interchangeable in the sense that it is possible to (i) encrypt a plaintext with the private key and decrypt the ciphertext with the public key or (ii) encrypt using the public key and decrypt with the private key. In case (i), the algorithm is used to ensure the integrity and prove the authenticity of a message. In contrast, in case (ii) the algorithm is used to ensure confidentiality of the message.

2.2.4 *Block*

From a data structure point of view, a blockchain is a chain of blocks. A **block** contains an ordered list of cryptographically signed transactions. Blocks in the chain are linked through a hashing mechanism: a block n stores the hash of the previous block $n-1$ in its header. This hashing feature makes the blockchain tamper-proof and tamper-evident [9]. It is noteworthy to specify that it is not impossible to modify a blockchain. However, doing so requires a huge amount of power, and it is extremely difficult. Moreover, the longer the chain of hashed blocks, the more difficult it becomes to modify their history.

Every blockchain implementation defines the exact structure of the block. However, most of the implementations divide the block into two parts [9]:

1. **Block header**. A block header includes metadata for a block. It might include:
   - the block number, sometimes known as block height;
   - the hash of the previous block, although in some implementations a block contains the hash of the previous two blocks;
   - a hash value representing the list of transactions bundled into the block;
   - a timestamp that records the moment in which the block has been created;
   - the size of the block;
   - the nonce value, which is used by the node that publishes the block to solve the cryptographic challenge.

2. **Block data**. Data stored in a block includes the list of cryptographically signed transactions.

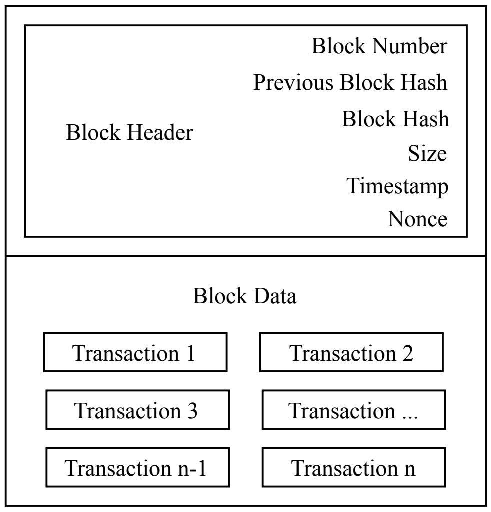


Figure 2: Generic structure of a block containing block header and block data

2.2.5 *Node*

A **node** is a participant in a blockchain network. It is often referred to as **peer**. Nodes in the network are responsible for storing the ledger, bundling transactions, creating, validating, and broadcasting blocks to the other nodes. We can identify different types of nodes depending on their role:

- full nodes ensure that transactions are valid by storing the complete blockchain; among these, publishing nodes also participate in the process of adding new nodes to the blockchain;
- lightweight nodes do not store or maintain the complete blockchain and pass transactions to full nodes for approval.

2.2.6 *Transaction*

In the blockchain, a **transaction** is an interaction between two entities E_1 and E_2 in the network [9]. Transactions are initiated by the sender through software and are sent to one or mode nodes in the network. Transactions are packed with other transactions to form a block, and the block is broadcasted to the other nodes. A transaction is finally added to the ledger when the network reaches an agreement on the fact that transactions inside a block are valid and authentic. Once consensus is reached, the new block is propagated in the network to update participants.

Data stored in a transaction depends on the particular implementation of the blockchain. However, the mechanism used by the participant to create transactions is quite similar in most of them [9]. A network user, the sender, initiates a transaction by using dedicated software. The sender specifies its identifier and the identifier of the receiver as well as the input and the output of the transaction. In the standard settings, the **input** of a transaction includes the list of digital assets to be transferred to the recipient. An entry in the list is a reference to the source of that digital asset, which is either the transaction in which the sender received the asset or the event in which the asset has been created. The **output** of a transaction includes the identifier of the recipient and the number of assets to be transferred.

2.2.7 *Ledger*

A **ledger** is a structured collection of transactions [9]. At first, ledgers were paper-based and used to keep track of the exchange of assets among parties. With the development of digital technologies, paper-based ledgers became digital and stored in large databased, often controlled by a single and trusted third-party organization. In recent times, there is a growing interest in distributed ledgers, and blockchain is one of the technologies that enable distributed ownership and distributed infrastructure.

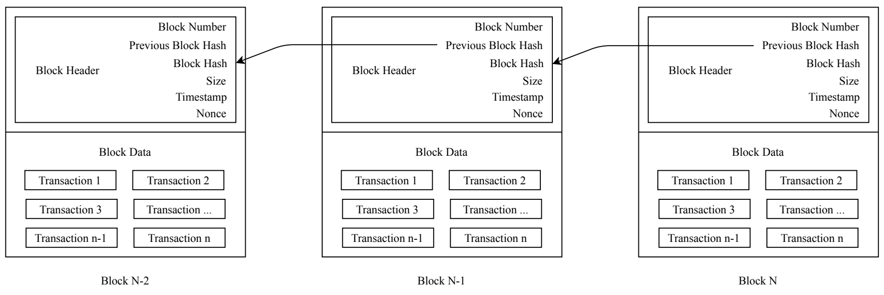


Figure 3: Generic blockchain ledger built as a chain of blocks

2.2.8 *Consensus*

As there is no trusted third-party authority in the network that regulate transactions and resolves disputes, there is the need for a mechanism that enables parties in the network to agree on a common state of the ledger. Such agreement is reached through the use of a **consensus** mechanism. The consensus mechanism determines which blocks will be accepted as part of the blockchain and in which order. The problem of reaching consensus among parties in blockchain network can be seen as a specialization of the Byzantine Generals problem [11]. Firstly identified in [12], the problem refers to a group of generals that are chasing a city, and they must collectively decide to attack or retreat from the campaign.

Depending on the type of blockchain network, a different consensus mechanism can be employed:

- **Proof of Work**. Proof of Work (PoW) is a computationally intensive consensus mechanism by which the node that wants to publish a new block to the blockchain must solve a complex challenge. The challenge is usually in the form of finding a value, the **nonce**, such that the hash of the block is lower than a certain value. The complexity of the challenge is modified to regulate the rate at which new blocks are published. As a node finds the right nonce that satisfies the requirements, it broadcasts the block to all other nodes to be validated. When a node validates a block, it forwards it to the others to improve the update speed.

- **Proof of Stake**. Proof of Stake (PoS) is a consensus mechanism that uses the stake a user invested into the network to decide the right candidate to add a new block to the blockchain. The rationale behind the mechanism is that the higher the stake a user has traded in the network, the lower is the probability it is willing to subvert it.

- **Proof of Authority**. Proof of Authority (PoA), sometimes known as Proof of Identity (PoI), is a consensus mechanism where the identity of publishing nodes have been verified through their link with real-world identities. The likelihood of being assigned with the task of adding a new block is proportional to the reputation a node has.

- **Proof of Elapsed Time**. Proof of Elapsed Time (PoET) consensus mechanism is based on a random waiting time generated by trusted secure hardware. Each participant in the network requires a waiting time to the secure hardware time generator and stays idle for the selected time. After being idle, it wakes up and publishes a new block.

- **Practical Byzantine Fault Tolerance**. Practical Byzantine Fault Tolerance (PBFT) is a consensus mechanism that relies on replication to tolerate Byzantine faults [13]. PBFT tolerates the presence of at most $\lfloor \frac{n-1}{3} \rfloor$ faulty nodes in the network. It works in three phases: pre-prepare, prepare and commit. Pre-prepare and prepare are used to order requests, whereas prepare and commit phases are used to ensure that committed requests are ordered [14]. Before moving among phases, each node waits to receive a confirmation from at least $\frac{2}{3}$ of nodes.

- **Round Robin**. Round Robin (RR) consensus mechanism is used in some permissioned blockchain and is based on rounds. At each round, a node is selected to add a new block to the blockchain. At the next round, a new node is selected, thus turning the task of creating blocks among the network participants.

2.2.9 *Smart Contract*

Nick Szabo has coined the term smart contract in 1994 as [15]

> [...] a computerized transaction protocol that executes the terms of a contract. The general objectives of smart contract design are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitration and enforcement costs, and other transaction costs.

In other words, smart contracts are contracts whose terms are recorded in a computer language instead of legal language. They can be automatically executed by a computer system to perform a transaction when certain conditions are met [16]. According to [17], smart contracts should carry three key characteristics:

- **observability**: the ability of the principals to observe each other's performance of the contract, or to prove their performance to other principals [17];
- **verifiability**: the ability of a participant in a contractual agreement to prove to an arbitrator that a contract has been performed or breached, or the ability of the adjudicator to find this out by other means [17];
- **privity**: the principle that knowledge and control over the contents and performance of a contract should be distributed among parties only as much as is necessary for the performance of that contract. This is a generalization of the common law principle of contract privity, which states that third parties – other than the designated adjudicators and designated intermediaries – should have no say in the enforcement of a contract [17].

As an additional constraint, smart contracts are required to be deterministic, i. e., if the same input is submitted, the same output should be returned. Therefore, smart contracts can work only with the data specified when they are called. A smart contract cannot perform network requests, read data from the disk, or retrieve data from external sources.

2.2.9.1 *Ethereum and Smart Contracts*

Ethereum is a blockchain-based platform proposed by Vitalik Buterin in late 2013. Building on the limitations of Bitcoin, Buterin suggested the development of "*an alternative protocol for building decentralized applications*" [18]. To achieve this goal, the author envisioned a blockchain with a Turing-complete programming language that allows anyone to develop decentralized applications and smart contracts. Ethereum was the first blockchain project that offered the possibility to develop smart contracts to improve the limited possibility of the scripting offered by Bitcoin.

In Ethereum, the state is made of *accounts*. Each account has a 20-byte address, and transitions between two states happen when value or information is transferred between accounts. Accounts can be of two types: externally owned accounts and contract accounts. While a user's private key controls externally owned accounts, contract accounts are controlled by their contract code [18]. Similarly to other transactions, smart contracts are deployed on the blockchain by issuing a transaction that will create the address for the contract account. When the smart contract is deployed, every message received by the corresponding contract account results in the activation of its code to perform certain operations.

2.2.10 *Blockchain Taxonomy*

There is an increasing agreement on the taxonomy of blockchain proposed in [19]. This taxonomy includes **public** blockchain, **private** blockchain, and **consortium** blockchain. In a public blockchain, anyone is allowed to send transactions and to participate in the consensus process. Additionally, anyone can read the content of the transactions that happened in the network. On the other hand, a private blockchain is governed and controlled by a single organization and only nodes belonging to that organization are allowed to perform transactions and participate in the consensus algorithm. Read permissions might be restricted to the organization's nodes or open to external parties depending on the particular application. Last, a consortium blockchain is one in which the consensus mechanism is controlled by a group of pre-defined nodes belonging to different organizations. As in a private blockchain, read permissions might be restricted to participants only or open to the public. Regardless of the type of blockchain, we can identify three common characteristics [10]: (i) all types make use of a peer-to-peer network to send and process transactions, (ii) all types require that transactions are digitally signed, the chain is append-only, and participants maintain a shared copy of the ledger, and (iii) all types employ a consensus mechanism to agree on a consistent state.

2.2.11 *Blockchain Decision Models*

Multiple frameworks have been proposed to discuss when it might make sense to evaluate the implementation of a blockchain instead of a centralized or distributed database. In the following, we present some of the proposed ones that will be useful during the evaluation of the existing literature.

One of the first known discussions on whether blockchain fits a particular use case has been proposed by Gideon Greenspan in an online blog [20]. The author identifies eight conditions that need to be fulfilled before starting a blockchain project: (i) the need of a shared database, (ii) the presence of multiple entities writing to the database, (iii) a certain level of mistrust between the involved parties, (iv) the willingness to remove a centralized authority to disintermediate the process, (vi) the presence of a relationship among transactions, (vi) the agreement on a set of legitimate transactions, (vii) the agreement on a set of validators (miners or node that execute the consensus protocol), and (viii) the presence of a connection between real-world assets and their representation as transaction assets. The first structured methodology has been proposed Wüst and Gervais in [21]. The authors provided a flow chart to determine if blockchain application is suitable for a particular use case depending on several properties. Wüst and Gervais decision model helps understand which type of blockchain should be employed for a given use case. In a use case where writers are not known a-priori, then the only alternative is a public (permissionless) blockchain. Instead, if all writers are known, it is worth evaluating the presence of a trusted third-party. If such third-party exist, and it is always online, there is no need for a blockchain, and a standard database with shared access better fits the situation. On the contrary, if the third party is offline and the participants do no trust each other, it can play the role of a certification authority on a permissioned blockchain.

As observed by Koens and Poll, many frameworks have been proposed. However, most of them ignore possible alternative solutions to blockchain technology [22]. To overcome that limitation, the authors propose an additional framework that answers three main questions: (i) should you use a blockchain? (ii) if so, which type of blockchain is best? and (iii) if not, which alternative is best? Figure 4 illustrates the flowchart proposed by Koens and Poll.

2.2.12 *Our Definition of Blockchain*

Due to its relatively short history, there is not a single definition blockchain. Therefore, we felt the need to introduce the definition that we will use throughout the document. By providing this, we are not try-

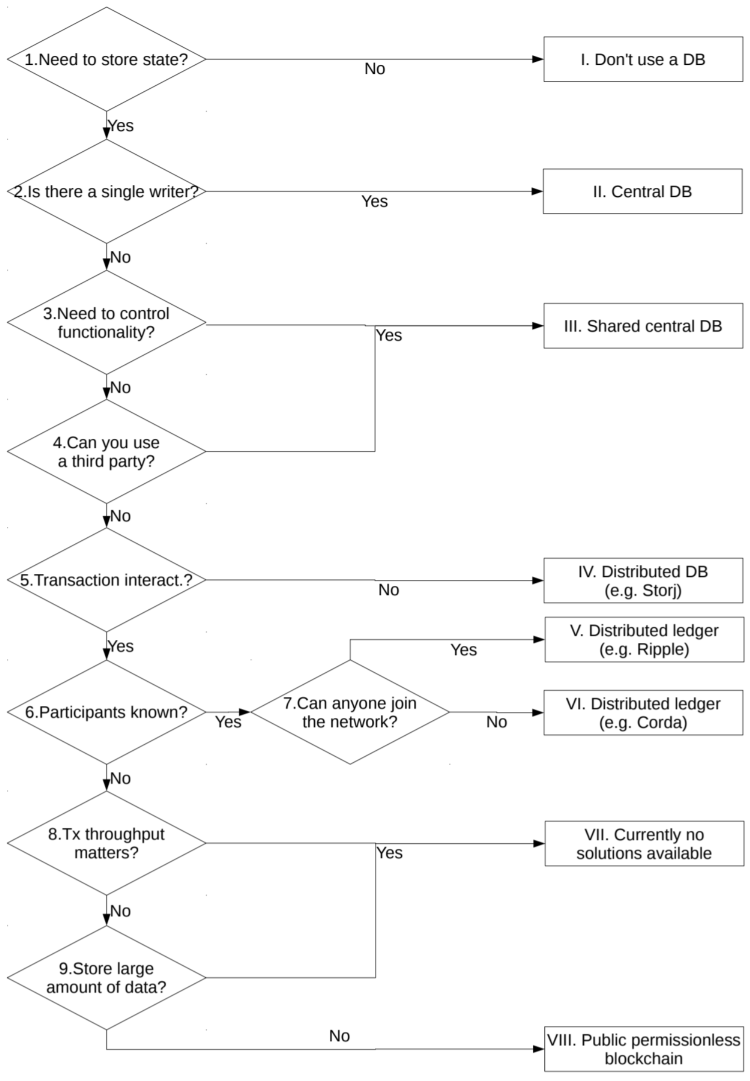

1.Need to store state?
No
I. Don't use a DB
Yes
2.Is there a single writer?
Yes
II. Central DB
No
3.Need to control functionality?
Yes
III. Shared central DB
No
4.Can you use a third party?
No
5.Transaction interact.?
No
IV. Distributed DB (e.g. Storj)
Yes
V. Distributed ledger (e.g. Ripple)
Yes
6.Participants known?
Yes
7.Can anyone join the network?
No
VI. Distributed ledger (e.g. Corda)
No
8.Tx throughput matters?
Yes
VII. Currently no solutions available
No
9.Store large amount of data?
No
VIII. Public permissionless blockchain

Figure 4: Koens and Poll blockchain decision framework from [22]

ing to introduce a universally accepted definition of the technology. Rather, we are trying to provide a solid basis for our arguments.

For the purposes of this document, a blockchain is a peer-to-peer distributed ledger that stores transactions in a chain of blocks connected through the use of a cryptographic hash function. The ledger is shared and replicated among the nodes in the network. Nodes agree on the changes to the ledger by approving and validating transactions through a consensus mechanism.

Nodes might have different roles, such as miners or validators. The formers are involved in the creation of new blocks, whereas the latter participate in the validation of newly created blocks. Note that a blockchain may not need such distinction as nodes can take on different roles in different moments depending on how they interact with the network.

Users interact with the network to submit transactions. Typically, users do not need to store the complete ledger of the blockchain and are allowed to store only the information they need for their interaction.

2.3 HYPERLEDGER FABRIC

Hyperledger Fabric is an open-source enterprise-grade distributed ledger technology platform developed under the umbrella of the Hyperledger project by the Linux Foundation. Fabric is a permissioned DLT platform in which parties are known to each other, but they do not necessarily need to trust one another fully. It has a modular design that allows some components to be switched based on the particular use-case needs. For instance, the consensus protocol is pluggable and can be adjusted depending on the trust models on which the network operates. Hyperledger Fabric does not require a native and built-in cryptocurrency and supports the development of smart contracts - called chaincode - in general-purpose programming languages. In this background section, we present the building blocks of Hyperledger Fabric, and we will explore its new architecture for transactions.

2.3.1 *The Blockchain Network*

In this section, we present the main components of a Fabric network. While some of these are quite common and can be found in many other blockchain platforms, such as peers, other components are unique to Fabric, such as the ordering service.

2.3.1.1 *Peers*

A peer is a fundamental component of a blockchain network. It hosts the ledger and smart contracts. As usual, the ledger records the immutable history of all transactions, and smart contracts are used to interact with the ledger to read or modify assets. In Hyperledger Fabric, a ledger is made of two components: the world state and the blockchain. The world state holds the current values for each asset. Each asset is associated with a version number that represents the number of times its values has been updated. The blockchain, instead, is the log of all transactions representing the changes that have been done to achieve the current world state's values.

2.3.1.2 *Channels*

Hyperledger Fabric supports the privacy of data through different means. Channels can be considered as subnetwork that includes two or more network participants. The purpose of channels is to allow participants to conduct confidential transactions without disclosing the content of a transaction to the whole network. Each channel has its ledger with its world state and blockchain. Similarly, chaincode is installed in a channel, and all peers in that channel will have an instance of the chaincode. Chaincode can also be designed to communicate between channels so that ledger information from another channel can be accessed if needed.

2.3.1.3 *Ordering Service*

The ordering service is a unique feature of Hyperledger Fabric. It is composed of many nodes called orderer - or ordering nodes - and its function is to generate an ordered list of transactions and to create blocks. The ordering nodes provide a deterministic order of a set of transactions, thus avoiding forks in the blockchain. As we will explore in Section 2.3.2, the ordering service plays a central role in transaction hashing and block creation. Thanks to the separation between the chaincode execution and transaction ordering, Fabric has been able to achieve high performance limiting the scalability issues of many other blockchain platforms.

2.3.2 *A New Architecture for Transaction*

Many existing blockchain platforms supporting smart contracts employ an *order-execute* approach to handle transactions. According to this approach, transactions are validated, ordered, and propagated to all peers in the network, which then execute the set of transaction sequentially. Hyperledger Fabric introduces a new approach to handle transactions called *execute-order-validate*. Instead of breaking the process into two steps, the Fabric model is constituted of three steps:

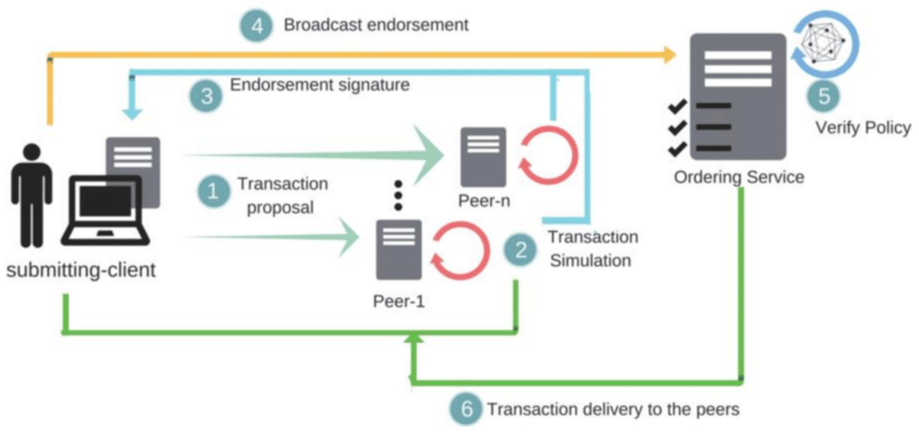


Figure 5: Hyperledger Fabric Transaction Flow[1]

1. transactions are executed, checked, and endorsed;
2. transactions are ordered and bundled into blocks by the ordering service;
3. transactions are validated to check their compliance with endorsement policies before committing them to the ledger and applying their changes.

Transaction ordering and block creation are tasks handled by the ordering service. The consensus on which transactions are valid is achieved through the use of endorsement policies. An endorsement policy specifies which peers of an organization need to execute and check the validity of a transaction before submitting it to the ordering service.

2.3.2.1 *Transaction Flow*

We have just seen how Hyperledger Fabric employs a unique approach to handle transaction execution and ordering. In this section, we will take a closer look at the Fabric's transaction flow that will be later useful to understand our design. A graphical representation of the transaction flow can be found in Figure 5[1].

Hyperledger Fabric makes use of the *execute-order-validate* approach to transactions. This 3-phase approach can be further divided into six distinct steps:

1. **Execute**. The execute phase is also known as the proposal phase and is the first of the three-phase approach.
   a) **Initiation**. The first step is executed by the client application that submits a transaction proposal to a set of peers

1 Image credit goes to Olivia Choudhury et al., Enforcing Human Subject Regulations using Blockchain and Smart Contracts

based on the endorsement policy. The proposal specifies the chaincode function to execute and the input parameters for that function.

   b) **Endorsement and Execution**. A subset of peers - called endorsing peers - receives the transaction proposal from the client application. They verify the proposal to check that it is well-formed, it has not been already submitted, the signature is valid, and the client is authorized to propose modifications to the channel's ledger. Once validated, the chaincode function is executed to create the *read-write set*, i.e., a set of assets specifying the value of each asset before and after the transaction execution. The proposal response is sent back to the client application.

2. **Order**. The order phase is the second phase of the transaction flow in which transactions are ordered and bundled into blocks by the ordering service nodes.
   a) **Inspection**. Once the client application receives the proposal responses, it verifies the signatures of the endorsing peers and checks whether the proposal responses are the same. If the client application was only querying the ledger, the transaction is not submitted to the ordering service. If instead, the client application wants to update the ledger, the transaction is broadcasted to the ordering service nodes.
   b) **Ordering and Bundling**. Upon receiving transactions, the ordering service orders them chronologically for each channel and creates blocks of transactions for each channel.

3. **Validate**. The validate phase is the third and last phase of the approach in which committing peers validate the transactions and update the ledger based on the output of valid transactions.
   a) **Validation and Commitment**. Blocks of transactions are delivered by the ordering service to all peers. Peers validate the transaction to ensure the endorsement policy is satisfied and no ledger updates for the assets specified in the transactions have been performed since the *read-write set* was created.
   b) **Update**. Each peer will append the block to the channel's blockchain and will update its ledger based on the output of the valid transactions. Invalid transactions remain into the block but are not executed, and their output does not update the ledger.

3 RELATED WORK

In this chapter, we present and discuss the proposals to achieve a modifiable blockchain architecture. We categorize the literature into four different categories, namely (i) structural approach (ST), (ii) local approach (LO), (iii) layered approach (LA), and (iv) account-based approach (AC). We first present each paper, and we conclude each presentation by identifying its limitations. We build on those limitations to produce an improved version of redactable blockchain. We conclude with a tabular representation and a summary of the related work.

3.1 STRUCTURAL APPROACH

Perhaps the first proposal to modify the content of a blockchain was given in [23]. Ateniese et al. argue that several reasons call for an editable blockchain, ranging from the removal of inappropriate content to compliance with regulations.

They propose the use of chameleon-hash functions that leverage the ability to efficiently find collisions for a given hash by knowing a secret trapdoor. The presented approach allows the modification of a blockchain that can be categorised into three different types: (i) modification of a block, (ii) compression of a set of blocks into a smaller set, and (iii) insertion of one or more blocks. Ateniese et al. adapted a general chameleon-hash function to a specialised chameleon-hash function that does not suffer from key exposure problems. These problems arise in old chameleon-hash functions when a party, once it sees a collision, can find other collisions or recover the secret trapdoor. The presence of a trapdoor that supports the editability of the blockchain introduces trapdoor management problems. The authors envisage solutions for the three types of blockchain that works either by giving power to a central authority or by sharing portions of the trapdoor with a pre-defined set of parties in the network that can derive the complete secret through multi-party computation schemes.

We identified various limitations in [23]. First, it is not possible to distinguish between an original block and a modified one because its deletion does not leave any trace. While the use of chameleon-hashes is a very elegant way to preserve the integrity of the chain, we argue that there should be an alternative mechanism to show that a modification has happened. Second, the presence of trapdoor introduces additional key management concerns. However, we note that with sufficient governance and a clever key distribution algorithm, this limitation may be overcome. Third, deletions are limited to a block level, meaning that whenever a transaction includes content that wants to be removed, the whole block containing the transaction needs to be deleted. This last limitation seems, from a security standpoint, the less worrying. However, it is a clear, practical limitation as a block contains many transactions that belong to different users. At first sight, it seems unnecessary to alter a whole block when a more fine-grained mechanism targeted at modifying a single transaction could be implemented using a similarly elegant implementation of chameleon-hashes.

A finer-grained and controlled version of a rewritable blockchain is proposed in [24]. Building on the proposal of Ateniese et al., the authors address the block-level redaction limitation presented in [23] to support transaction-level redaction. To achieve the desired goal, Derler et al. introduce the concept of policy-based chameleon-hash (PCH) functions by associating access policies to the hash computation. In particular, they combine CP-ABE functionalities with chameleon-hash with ephemeral trapdoors (CHET) functions, a variant of chameleon-hash functions where two trapdoors are required to compute a collision. Precisely, in addition to the trapdoor used in the standard construction of chameleon hashes, this primitive employs a second - ephemeral - trapdoor, specified during the hashing process and needed to compute a collision. This requirement allows providing a separate second trapdoor for each hash, instead of a single trapdoor for every hash calculated with a unique public hashing key [24].

Every participant obtains a secret key for the computation of the hash function and a second secret key associated with a list of attributes used to perform CP-ABE. To hash a message with respect to an access policy, a user computes the chameleon-hash function with the ephemeral trapdoor and encrypts the trapdoor using the encryption algorithm of CP-ABE. To modify an approved transaction, a user that has a private key satisfying the access policy can reconstruct the ephemeral trapdoor and compute a collision for the transaction's hash.

Even though the proposed solution is elegant and allows for modifications on a transaction level, a significant limitation of the approach is the absence of public evidence that a transaction has been modified. As observed before, the use of chameleon-hashes allows an invisible change. However, we argue that such an imperceptible change sharply weakens the tamper-evident feature of a blockchain. As a result, the integrity and immutability of the ledger may suffer severe consequences.

The work proposed in [25] differs from most of the approaches as it does not deal with hash functions. Rather, Cai et al. introduce a deletable blockchain based on the proof-of-space consensus mechanism in which the three components of a transaction, i.e., the identity of the sender, the identity of the receiver, and the content of the transaction, do not need to be public.

During the deletion process, the system uses a traceable ring signature or a Pedersen commitment scheme to disclose the sender's identity or the transaction content, respectively, depending on the privacy requirements. The deletion request can be submitted only by the sender of the transaction and signed by the participants of the network that agree on removing the transaction using a linkable multi-signature scheme. If the multi-signature is valid, i. e., it is not generated by a single malicious user, the rest of the users in the network accept the deletion operation.

Upon transaction submission, a traceable ring signature is created by the sender, attached to the transaction, and broadcasted to the network. If the sender chooses to reveal its identity to delete the block of transactions, it generates another traceable ring signature. In case both signatures are valid, the other users in the network compute the various public keys used to sign the transactions in the block (which are assumed to be created by a single sender). If the process results in the identification of a unique public key, the key is revealed and the deletion process proceeds. If the sender decides to disclose the content of the transaction, instead, it generates a Pedersen commitment scheme. Similar to the previous procedure, the other users of the network check whether the committed value holds within the same block and eventually proceeds in the deletion process. If the request to delete a block is valid, the network generates a so-called linkable digital multi-signature used to replace the transaction block. Such substitution does not create clashes in the hash chain due to a particular block design implemented by the authors.

In their protocol, Cai et al. assume that all transactions in a block are originated from the same sender. This assumption might be a limitation as most of the blockchain architectures do not have similar restrictions. Another limitation we identified in the proposed scheme is the need to disclose part of the content of a transaction to propose a deletion. In highly sensitive domains, such as healthcare, this is likely to be unacceptable because a patient might not be willing to reveal its identity to the whole network to ask for the deletion of its personal information. Further, the creation of a specialised block structure makes the scheme incompatible with existing blockchain implementations. A careful reader may notice that a modification of the block creation procedure happens with the use of chameleon-hash functions as well. However, we argue that such change is minimal compared to the ones proposed in this scheme. Last, Cai et al. claim to provide a deletion mechanism able to work on a transaction level. However, the fact that the same sender needs to generate all transaction in a block prevents the scheme from being used at a real transaction level. When a redaction is performed, all transactions inside a block are invalidated. As a result, the whole block becomes invalid.

The use of an additional cryptographic technique is proposed by Lee et al. The authors adopt truncated hash values to compute the hash value of a block and employ a multichain structure [26]. Upon transaction submission, the transaction owner sets a difficulty level that determines the amount of work needed to modify the transaction. Noteworthy, the owner can propose an immutable transaction by adequately setting the difficulty level. Approved transaction are bundled into blocks and attached to a single main chain after being mined. Depending on the difficulty level, a transaction owner can submit a modification request to a sidechain network [26]. The choice of the sidechain on which to propose the modification depends on the difficulty level of the transaction. Sidechains are controlled from the main chain with the tail bits of the block hash in the main chain [26]. When a sidechain is reconnected to the main chain, the modification requests are executed.

An explicit limitation of this approach is the ability of the sender to decide the difficulty level of transaction modification. A malicious user can set the difficulty level high enough to create an immutable transaction. Hence, the solution assumes that no malicious users submit unlawful immutable transactions. Similarly, drawing a parallel with the GDPR, a malicious peer can set the difficulty level of a transaction high enough to render the transaction immutable. Therefore, there may be situations in which data cannot be deleted or modified upon the data subject's request.

Another structural approach is suggested by Deuber et al. The authors developed a consensus-based voting mechanism that does not rely on trust assumptions or heavy cryptographic primitives [27]. Consensus voting is combined with policies to dictate which modifications are allowed. Additionally, the protocol provides public verifiability and accountability [27]. Challenging the proposals that suggest the use of heavy cryptographic primitives, the authors provide a decentralised mechanism for a permissionless network in which anyone can propose modifications. Miners vote on the modification proposals and check if the request satisfies the policy. To account for modified blocks, both the block validation algorithm and the chain validation algorithm are modified. Intuitively, a modified block clashes with the previous and the following blocks. Hence, the modification of the validation strategy is supported by the inclusion of an additional Merkle tree in the block structure to store the old state information of the redacted block.

Similarly to the work of Ateniese et al., the redaction happens at a block level. Moreover, the work of Deuber et al. requires to modify the validation process. In particular, the old state of a block, represented by the Merkle tree, needs to be kept so that the new validation algorithm works.

An analogous approach is presented by Xu et al.: a redactable proof-of-stake-based blockchain featuring fast confirmation and public verifiability. Their proposal is driven by the absence of algorithms that allow a rapid approval of a redaction proposal in a permissionless setting [28]. Any user in the network can propose a redaction by submitting a candidate edited block. The members of the voting committee, i.e., users that have the rights to approve the proposed candidate block, are decided through the use of a verifiable random function. Each stakeholder privately executes the test to verify whether it is part of the voting committee. Being a PoS permissionless blockchain, users' likelihood to be engaged in the voting process depends on the stakes they own. Any edited block in the chain is publicly verified, and multiple redactions are allowed in a single block. Similarly to [27], Xu et al. propose the addition of a redaction policy and the modification of the block structure, block validation and chain validation algorithms. The block structure is modified to include the original state of a block before a redaction happens. The original state is then used in the block validation and chain validation algorithm. In the block validation algorithm, the procedure validates the data included in the block and the signature of the leader who proposed the block, but, for an edited block, the validation of the signature happens against the original state. In the chain validation algorithm, the procedure checks the relationship with the block with the neighbouring ones. In case of a redacted block, the validation process verifies whether the redaction policy is satisfied.

The work of Xu et al. presents limitations that are similar to the work of Deuber et al. except for block-level modifiability. The old state of the block still needs to be saved such that the new validation procedure succeeds. Moreover, their strategy introduces modifications to the block structure, to the block validation algorithm, and to the chain validation algorithm.

An original solution is presented in [29]. Puddu et al. present the possibility of mutable proof-of-work blockchain that provides the same tamper-resistance and guarantees of immutable blockchain [29]. The proposed model achieves mutability thanks to its ability to maintain alternative versions of data records, agree on a valid version, and hide the invalid ones. Alternative versions are structured as a set of different transaction versions, where only one transaction is specified as active. A specific transaction version included in the collection is the *nope*, used to delete content. All mutation requests are subject to a policy check to ensure modifications are triggered by authorised entities as specified by the transaction sender [29]. At a certain point in time, the active transaction is decrypted, whereas all the other transactions in the set are encrypted. Decryption keys are distributed among miners using secret sharing. Upon modification request, miners engage in a multi-party computation protocol to recover the appropriate decryption key and decrypt the required version of the transaction. To address the problem of transaction consistency that happens when a transaction depends on a modified one, Puddu et al. propose the use of a cascading effect in which all affected transactions are modified.

Using the approach of Puddu et al., the sender of the transaction can encrypt alternative versions of data. While this approach could be used to support deletions by providing an encrypted *empty* transaction, all modifications might not be known a-priori. If data needs to be modified with a mutation that was not initially envisioned, this solution will not help.

3.2 LOCAL APPROACH

In contrast to most of the proposals, the work of Florian et al. focuses on a local erasure of data without requiring significant modifications to the transaction protocol. According to the authors, a local modification is more practical and more natural to adopt compared to a global change of the blockchain protocol. The proposed work is a functionality-preserving local erasure (FPLE) deployed as an extension to node software. FPLE allows participants to select a portion of transaction data and to erase that content without requiring major protocol modifications or coordination procedures [30]. Whenever a node deems an erasure necessary, it stores the transaction in the erasure database with the deleted part overwritten. The original transaction is then removed or overwritten from its original location. To avoid that after a deletion a node cannot correctly recognize new blocks that depend on the erased one, the authors enforce two rules: unconfirmed transactions with reference to deleted data are considered invalid, whereas confirmed transactions that cannot be verified are accepted trusting that miners are behaving correctly and have sufficient incentives to mine valid blocks.

3.3 LAYERED APPROACH

The layered approach proposes to redact a blockchain through the use of an additional layer built on top of the blockchain itself. An example of such an approach is the work of Thyagarajan et al. They present a publicly-verifiable layer that allows modifying blockchain content [31]. Any user in the network can propose a modification, and a deliberation process performs a decision on the request to check whether it complies with the repair policy of the chain. Network participants agree on the policy to be checked upon repair request that specifies the requirements a proposal needs to satisfy. After a proposal is submitted, a group of deciders (a subset of participants) propose a decision and post it into the chain. Once the decision is taken, miners (i) verify that the proposal satisfies the policy requirements, (ii) check that deciders have accepted the proposal, and (iii) update the block according to the proposal. To support multiple repairs and allow the validation of the chain after a modification, Thyagarajan et al. propose the use of two additional data structures: a repair layer database and an approved repairs database. There is a one-to-one relationship between a block in the chain and an entry in each database. The repair layer database is used when the block contents need to be deleted. Precisely, the repair database stores the hash of the transaction to be removed from the original block. Instead, the approved repairs database stores the approved repair proposals. The proposed work achieves a partial level of integration with existing blockchain implementation thanks to the modification of the validation algorithm. Instead of validating the modified chain, the chain is verified using the original content on each block. Hence, the hash chain appears not to be broken.

3.4 ACCOUNT-BASED APPROACH

Unlike the previous approaches, account-based focuses on the modification of a user's account status. The work of Gates is the only one to date that approaches the problem of blockchain modifiability from this perspective. The author proposes GateChain, an architecture that supports revocable transaction models (RTM). To achieve the desired revocation, the blockchain leverages an augmented account model that supports additional transaction types. In this approach, the revocation of a blockchain transaction does not require its deletion. Instead, the model supports the user in withdrawing the account status changed due to an incorrect transaction [32]. To achieve so, the RTM of GateChain introduces two account types: the standard account and the vault account. The standard account follows the transaction model of Bitcoin (UTXO), and it is used for conventional transactions that do not support revocable states. Instead, the vault account is specifically designed to support revocable states [32]. Since the revocation of account status can produce a change only after a predefined delay period, a user willing to revoke a transaction can initiate the procedure before the delay period expires. The proposed model does not entirely solve the problem of modifying or deleting transactions from a blockchain. Instead, [32] can be better classified as a safety mechanism that allows users to revert a transaction in case they realize that an error occurred. Notably, this revocation mechanisms can be applied only of a transaction is issued from a vault account, meaning that erroneous transactions created from a standard account cannot be reverted.

3.5 SUMMARY

In this section, we summarize the various approaches designed for blockchain mutability. Table 1 gives a graphical overview of the discussion. In the table, category reflects the different types of approaches we identified in the literature. Settings define the blockchain architecture for which the solution is presented, being it permissioned, permissionless, PoS-based, PoW-based, PoSP-based, etc. Note that in this particular representation, PoSP refers to the proof-of-space consensus mechanism introduced in one of the proposals. Integration refers to the ability of the proposed scheme to work with existing blockchain architectures. Public verifiability specifies whether the modification can be seen and verified by the participants in the network. Accuracy defines the scope of the change and/or deletion, which can be either block or transaction.

The structural approach proposes a modification of the block structure to support the modification of the blockchain. The proposed solutions that fall under this category are usually not compatible with existing blockchain architecture due to the structure modification they require. Under the local approach, proposals suggest erasing data on a node-level rather than on a global level. By doing so, a node is free to decide whether removing or not the unwanted content when deletion has been approved. The layered approach we reported is designed to achieve a partial level of integration with existing blockchain. As a result, the client that interacts with the network, the block structure, or the blockchain architecture is partially (or not at all) modified. However, the process of validating the chain is changed to avoid that modifications result in an invalid blockchain. This approach is made possible through the creation of an additional layer on top of an existing blockchain that supports modifications. The account-based approach does not address the problem of modifying the blockchain entirely. Instead, it can be seen as a preventive measure that allows participants to revert an erroneously issued transaction before a specific time. After the predefined revocation period is expired, it is not possible to modify the transactions that have been accepted and added to the chain.

3.6 PRESENTATION OF OUR WORK

This section introduces our work starting from an analysis of what is missing in the related work we analysed. As can be seen from Table 1,

	Category	Settings	Integration	Public Verifiability	Accuracy
[23]	ST	Not specified	No	No	Block
[24]	ST	Not specified	No	No	Transaction
[25]	ST	PoSP based	No	Yes	Block
[26]	ST	Permissionless	Partial	Not specified	Transaction
[27]	ST	Permissionless	No	Yes	Block
[28]	ST	Permissionless	Partial	Yes	Transaction
[29]	ST	PoW based	No	No	Transactions
[30]	LO	Not specified	Yes	No	Transaction
[31]	LA	Not specified	Partial	Yes	Transaction
[32]	AC	Not specified	No	No	Transaction

Table 1: Taxonomy of proposed mutable blockchain schemes

most of the work proposes to redact content on a transaction level. We agree with such a decision, and we believe redactions should happen on a transaction-level to offer fine-grained modification abilities. Therefore, our work will strive to achieve transaction-level modifiability minimising the impact on the whole block.

We have already noted that the account-based approach does not solve the problem of modifying content on a blockchain. Instead, it is more a preventive measure. The local approach, instead, relies on the fact that every node in the network performs a redaction when needed. Last, the layered approach modifies the process used to validate a blockchain. Instead of validating the modified chain, this approach validates the original one. To do so, the original data still needs to be available for the validation to happen correctly. Even though the structural approach usually modifies the block structure to allow modifications, and it might be hard to integrate into an existing architecture, we believe it is the most viable approach among the ones we discussed. If wisely used, a structural approach could introduce little modification to the block structure or the block creation process and could render integration with an existing blockchain quite straightforward.

Another component we believe should be present in a modifiable blockchain is the ability to see that modification or deletion of data has been done at a certain point in time. While in some of the discussed work a change is visible, in others every action on past data is hidden. However, we think that the integrity of the ledger could be severely weakened and the tamper-evident property would be lost. We therefore suggest introducing a mechanism through which the network agrees on a modification. Such agreement results in a transaction published on the ledger so that every participant tracks the redaction in its history.

Given the fact that one of our goals is to allow redactions to ease the frictions between the immutability of the blockchain and the data subject's rights stated in the GDPR, we believe we should begin our discussion with an analysis on whether data on a blockchain falls within the scope of the regulation. Moreover, we think that the redaction process should involve the relevant parties, i. e., data subject, data controller, and data processor and that the request of redaction should be started by the data subject. We stress that all related work is lacking a proper discussion on the legal aspect of the problem, which should be the starting point towards the design of compliant blockchain applications. Therefore, in the next chapter, we will address the problem from a legal perspective with the goal of understanding if and which data on a blockchain should be redacted.

4 MAPPING THE GDPR ON THE BLOCKCHAIN

This chapter focuses on understanding which data on a blockchain could be subject to the Regulation and if modification of that data is needed to produce a compliant use of blockchain. We stress that, given the multidisciplinary nature of the problem, every technical effort should be based on a solid understanding of the legal aspect, which is lacking on the related work we discussed.

Before striving for a solution to the contrast between Article 16 and Article 17 of the GDPR and the immutability property of a blockchain, we will provide an introduction to the GDPR. GDPR applies only to personal data, and therefore we will analyse which data stored on a blockchain might classify as such and thus will be caught under the scope of the Regulation. We anticipate that, due to an unclear legal framework and conflicting opinions of different statutory bodies, it is very challenging to provide an answer with certainty.

Following the discussion on what type of personal data might be at stake in a blockchain, we proceed into the mapping of the core principle of the Regulation with the blockchain. As a result of this mapping, we identify the major conflicts between the two, namely the assignment of data controller and data processor roles, the exercise of data subjects' rights, and the transfer of personal data to third countries.

To conclude, we present the legal and technical requirements to build a compliance blockchain application without weakening blockchain core properties in such a way that would render its application useless.

4.1 INTRODUCTION TO THE GENERAL DATA PROTECTION REGULATION

The General Data Protection Regulation (GDPR) [33] is the new European regulation on data protection and privacy for all citizens of the European Union (EU) and the European Economic Area (EEA) that supersedes the Data Protection Directive 95/46/EC. Adopted on April 2016 and enforceable since May 2018, the aims of the GDPR are mainly to give individuals more control over their data and to simplify the regulatory environment within the European Union to provide clear visibility, understanding, and control over data that is processed. Unlike the Data Protection Directive, the GDPR is a regulation that directly applies to the European Union member states without additional local implementations.

Before diving into the details of the requirements imposed by the regulation and its goals, it is worth reflecting on the cases in which the GDPR does apply. Art. 2 GDPR and Art.3 GDPR identify the material and territorial scope of the regulation respectively. The material scope defines that the Regulation applies to the processing of *personal data*, both manually and partially or entirely automated. In simple terms, the regulation applies to any processing of personal data, and its material scope is defined broadly to ensure a high level of protection [34]. The territorial scope defines that the regulation applies to the processing of personal data belonging to EU citizens regardless of where the processing takes place.

The GDPR identifies two major actors that play a role in the processing of data subjects' data, namely the data controller and data processor. Following, we provide their definition that will be later used to assign their responsibilities to the actors that can be identified in a blockchain:

DATA CONTROLLER *"the natural or legal person, public authority, agency or other body which, alone or jointly with others, determines the purposes and means of the processing of personal data;"*. The controller can be then identified as the one entitled to decide the purpose of processing, which data should be processed, for how long, who can access, and what security measures need to be taken. If there is the case that several entities take the decision, there may be "joint controllers". The data controller must exercise control over the data processor, and it has the responsibilities for that process, including legal liability [34].

DATA PROCESSOR *"means a natural or legal person, public authority, agency or other body which processes personal data on behalf of the controller;"*. The existence of the processor depends on a decision taken by the controller that can decide to process data within the organization or to delegate the activity to an external person, namely the processor. To be qualified as a processor, one has to be a separate legal entity with respect to the controller and to carry out the processing of personal data on behalf of the controller. Processors are obliged to maintain a record of all processing activities to demonstrate their compliance with the regulation, to implement organizational and technical measures to secure the processing mechanism and to notify data breaches to the controller [34].

4.1.1 *Personal Data*

This section discusses whether data stored on a blockchain constitute personal data and will be caught within the scope of the Regulation. As we anticipated, there is a surrounding uncertainty on whether data processed by a blockchain should be considered as personal. This legal unclarity caused by missing guidelines and by conflicting opinions expressed by legal authorities hinders the development of GDPR compliant blockchain applications that process personal data [35]. GDPR defines personal data as "*any information relating to an identified or identifiable natural person*", and an identifiable person - usually referred to as data subject - as a natural person that

> "can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, psychological, genetic, mental, economic, cultural or social identity of that natural person".

Where data is stored completely anonymous, it is not considered personal data, and it does not fall within the applicability of the Regulation. Instead, pseudonymous data still qualifies as personal data and, as such, its processing is regulated. According to the GDPR, pseudonymous data is the one obtained with

> "the processing of personal data in such a manner that the personal data can no longer be attributed to a specific data subject without the use of additional information, provided that such additional information is kept separately and is subject to technical and organizational measures to ensure that the personal data are not attributed to an identified or identifiable person." [33]

According to [36], two categories of data on a blockchain might fall under the scope of the GDPR: transactional data stored inside a block and users' public keys. Transactional data include all data stored in a block except for public keys and might include personal data depending on the application the blockchain is designed to support. For instance, let us consider a financial transaction. The transaction content includes the sender's public key, the receiver's public key, and the amount to be transferred. Since we consider transactional data as all data except for public keys, the amount to be transferred constitutes our transactional data, which, alone, does not constitute for personal data as it does not refer to an identified or identifiable natural person. It is a number that, alone, does not allow any direct or indirect identification of a data subject. Let us now discuss the second example of a blockchain that manages medical data. A transaction might include the doctor's public key, the patient's public key, and a set of health-related data, such as heart rate, blood pressure, cholesterol levels, etc. Unlike the previous number, these can be used to identify a data subject either directly or by combining them with an additional set of data.

4.1.1.1 *Transactional data*

Transactional data can be stored in three alternative formats: plain text, encrypted, or hashed. In the following, we will evaluate whether one of these alternative storing formats is sufficient to classify data as anonymous and thus relieves us from the need to provide appropriate additional mechanisms to comply with the Regulation. The standard of anonymity imposed by the GDPR is high [36]. Precisely, anonymisation of personal data comes "*from processing personal data in order to irreversibly prevent identification*". Personal data stored in plain text format remains personal data and thus falls within the scope of the Regulation. Encrypted data is subject to the requirements of the Regulation as well because encrypted data can still be accessed by using the correct decryption key.

Under the EU data protection regime established by the GDPR, encryption is a pseudonymisation technique because the data subject can be indirectly identified [36, 35]. Transactional data that has been hashed constitutes personal data as well [35]. Even though hash functions are built in such a way that it is not possible to recover the original data without brute-forcing the input space, the Article 29 Working Party declared that hashing represents a form of pseudonymisation technique because it might still be possible to link the dataset with the data subject [36].

Arguably, from a technical point of view, it is very unlikely that hashed data can be used to recover the original data. However, from a legal point of view, none of the hashing techniques currently used has been declared as capable of anonymisation by the European Data Protection Supervisor [36]. To further clarify the matter, hashing algorithms are not considered able to achieve the standard of anonymisation if they are used alone. The use of salted has or peppered hash, however, increases the strength of hashing algorithms due to the presence of additional data added to the original one during the hashing process. With regard to salted hash functions, Article 29 Working Party have already presented its opinion stressing the fact that they are not capable of achieving anonymisation [37]. A slightly different opinion has been expressed with regard to peppered hash functions. Whilst Article 29 Working Party recognised that this technique offers higher privacy guarantees, it does not specify whether it is suitable to achieve anonymisation for GDPR purposes [35].

A naive solution to achieve compliance for transactional data is the one of storing personal data off-chain, meaning that data is not directly stored on the ledger. Instead, hash pointers of personal data are stored on the ledger so that it is possible to prove the integrity of the data without exposing it in a shared environment. It is worth noticing that this solution does not solve all the problems connected with personal data management. As observed by Finck, meta-data stored on the blockchain might reveal personal information even though personal data is not directly stored on-chain. Moreover, storing data off-chain might require the introduction of a trusted third party that handles the storage of data, removing part of the motivation for introducing a blockchain. However, storing personal data off-chain seems to be one of the most important steps to produce a GDPR-compliant blockchain application [36]. According to Finck et al., unless peppered hash functions are used in combination with additional privacy guarantees, their application will not render data anonymous because the criteria identified by Article 29 Working Party to determine whether the identification is still possible, namely singling out, linkability, and inference, are not likely to be satisfied [35].

4.1.1.2 *Public Keys*

Public keys are pseudonyms used to represent a user in a blockchain without the need to use other identification mechanisms, such as the US social security number or the Dutch BSN. As we have already seen in the case of transactional data, to avoid public keys being considered personal data, they should be anonymous. Referencing the definition of pseudonymisation we cited before, public keys are data that cannot "be attributed to a data subject without the use of additional information". However, when additional information is available and combined with a public key, identification can be considered plausible [36]. Hence, public keys are not regarded as anonymous data and, according to Finck, it is possible to conclude that public keys are pseudonymous data and fall within the scope of the regulation. Public keys are considered personal data in the case they are used to relate to a natural person [35]. Instead, if public keys are not used to identify a natural person, for instance, an organisation, they are not personal data.

Being an essential component of a transaction, it is not possible to move public keys off-chain, as doing so will remove any connection between the transaction and its participants. Moreover, public keys are required to submit and approve transactions and avoiding their use will prevent the use of the system. Given that it is not possible to store them off-chain only, compliance with the GDPR in this situation is more difficult to obtain [36].

Nonetheless, various mechanisms have been proposed to reduce the possibility of linking public keys with an individual. A first proposition is the use of stealth address, i.e., transactions that are associated with hashed one-time keys. Such a mechanism requires the creation of a new address and a new pair of keys for each transaction the user wants to submit. However, such an arrangement have been proved to be far from achieving high guarantees of privacy protection [36, 35]. Another possible mechanism includes the use of zero-knowledge proofs (ZKP) that provide the evidence that a transaction has occurred without disclosing the actual content of the transaction, such as transactional data and public keys. With regard to ZKP, a European Parliament issued report seems to consider zk-SNARKs - a particular design of ZKP - as a possible mean to comply with data protection by design requirements [35]. Ring signatures are another possibility to hide the public key of a user behind a public key associated with a group. Using this technique, it is possible to prove that a user in a group signed a transaction with a private key associated with the set of public keys connected to the group, without revealing the exact user. Homomorphic encryption allows performing computation on encrypted data to produce the same result of the calculation performed on the original data. Whilst it could make possible to store only encrypted data on the blockchain, it is doubtful that such data could be considered anonymous for GDPR purposes [35].

Another alternative is the addition of noise to the data such that it becomes impossible to define the identities of the senders and receivers of multiple transactions bundled together. Provided that noise techniques are combined with more robust privacy techniques, and necessary safeguards are put in place, Article 29 Working Party recognised that the addition of noise to data might be considered as an actual anonymisation technique [36]. However, we should stress that the decision is not universal, and additional safeguards and privacy technique might be evaluated on a case-by-case basis. According to Finck, it is not possible to conclude whether one of the mentioned mechanisms constitutes a suitable anonymisation technique that satisfies the standard of the GDPR. Hence, in many cases where personal data managed through a blockchain, both public keys and transactional data can constitute personal data for GDPR purposes.

4.2 MAPPING THE BLOCKCHAIN WITH THE GDPR

In this section, we introduce the principles set by the GDPR on the protection of personal data, and we draw a parallel between those principles and blockchain technology. According to the literature we reviewed [36], it is possible to argue that the principles of *Accuracy* and *Storage Limitation* are, at first sight, in contrast with how a blockchain is generally designed. On the contrary, it is much easier to find a gentle association between the other principles and such a distributed technology.

4.2.1 *GDPR Six Core Principles*

In this section, we present the six core principles on which data protection is based on. Being a particular type of distributed ledger, a blockchain requires a copy of the data is shared and replicated among nodes. At first sight, this replication feature might seem in contrast with many of the core principles of the Regulation [38]. In the following, we discuss how those principles reflect on blockchain and which are the pain points when someone tries to interface the technology and the Regulation.

4.2.1.1 *Lawfulness, Fairness, and Transparency*

Article 5 Sec. 1 lit. a GDPR introduces the principle of lawfulness, fairness, and transparency in the processing of personal data. Precisely, it states that "*personal data shall be processed lawfully, fairly and in a transparent manner in relation to the data subject*" [33]. In other words, personal data can be processed if and only if there is legal permission or the data subject's consent [34]. To give informed consent on the processing of their data, the data subjects need to understand how their data is being processed. Hence, the transparency principle requires that (i) the identity of the data controller is known to the data subject, (ii) there is enough information for the data subject to understand the purpose of processing, (iii) data subjects have the right to obtain confirmation and communication of processing activities, and (iv) data subjects are aware of risks, rules, safeguards, and rights related to personal data processing [33].

According to Ibáñez et al., transparency on the intended use of data and the presence of legitimate reasons to collect personal data are reasonably easy to achieve on a blockchain. Indeed, validation rules on a blockchain are usually public and can be communicated to data subjects. More problematic is the lawfulness of data processing activities. In particular, the issue resides in the control on data processor to make them accountable whenever something unlawful is done with data [38]. In public and permissionless blockchain, accountability on the data processor is hard to obtain because identities are not clearly established. However, the authors believe that under the assumption that validators are considered joint data controllers, it is easy to map a distributed data controllership to the centralised case covered by the GDPR [38].

4.2.1.2 *Purpose Limitation*

Article 5 Sec. 1 lit. b GDPR introduces the principle of purpose limitation in the processing of personal data. Precisely, it states that "*personal data shall be collected for specific, explicit and legitimate purposes and not further processed in a manner that is incompatible with those purposes;*" [33]. However, some purposes such as archiving for the public interest, statistical purposes, or scientific and historical research purposes should not be considered as incompatible. The purpose plays a central role in the lawfulness of processing activities as it specifies why personal data is collected and for what reasons it is processed. It allows to determines whether other principles, such as data minimisation, accuracy, and storage limitation are respected [34].

Similarly to fair and lawful processing of personal data, Ibáñez et al. considers this principle easy to respect because it is easy to inform users on the purposes of personal data processing. However, there is an issue in the case of public and permissionless blockchain because it is hard to control if someone takes advantages of the open availability of data to execute some form of processing that goes beyond the initial purposes [38].

Finck et al. identifies an additional problem concerning the purpose limitation principle whenever a blockchain processes personal data. Precisely, the question resides on whether the continuous processing of blocks after the execution of the transaction that inserted that data into the ledger can be compatible with the purpose limitation principle [35]. An additional and broader discussion on the purpose limitation principle can be found in [35].

4.2.1.3 *Data Minimisation*

Article 5 Sec. 1 lit. c GDPR introduces the principle of data minimisation in the processing of personal data. Precisely, it states that "*personal data shall be adequate, relevant and limited to what is necessary in relation to the purpose for which they are processed;*" [33]. Data minimisation principle should not be considered as a prescription to reduce data collection to an absolute minimum. Instead, its goal is to reduce the amount of data collected to the lowest possible to realise the processing purposes [34].

Ibáñez et al. considers this principle as independent from the use of a blockchain, meaning that data minimisation principle in the centralised setting considered by the GDPR can be easily applied to a distributed environment. Similarly to a centralised data controller, data controllers jointly running a consortium permissioned blockchain are required to assess that collected data does not exceed the required to perform processing purposes [38].

Finck et al. supports a different opinion on the data minimisation principle and expresses concerns about two characteristics of blockchain technology. First, the append-only and continuously growing structure of blockchain renders obsolete data very hard to deal with. Second, the fact that the ledger is replicated among nodes creates many replications of personal data [35].

4.2.1.4 *Accuracy & Storage Limitation*

We discuss the mapping of accuracy and storage limitation with a blockchain in the same section due to their close relationship when mapped with the technology. Both principles require the ability to modify or delete personal data subject to a data subject request. This ability seems at odds with the way blockchain is designed.

Article 5 Sec. 1 lit. d GDPR introduces the principle of accuracy in the processing of personal data. Precisely, it states that "*personal data shall be accurate and, where necessary, kept up to date; every reasonable step must be taken to ensure that personal data that are inaccurate, having regard to the purposes for which they are processed, are erased or rectified without delay;*" [33]. Connected with this principle is the right to rectification (Art. 16 GDPR).

Article 5 Sec. 1 lit. e GDPR introduces the principle of storage limitation in the processing of personal data. Precisely, it states that "*kept in a form which permits identification of data subjects for no longer than is necessary for the purposes for which the personal data are processed;*" [33]. Similarly to the data minimisation principle, storage limitation suggests that data should be stored as less as possible subject to the processing purposes for which it has been collected. This principle is connected with the right to erasure (Art. 17 GDPR).

Blockchain is designed to be tamper-proof and tamper-evident, and the possibility to make modification or deletion is hard on purpose. As we discussed in previous sections, the two properties, usually grouped and referred to as immutability, are highly praised and desirable in some scenarios. They are however, in direct conflict with the right to rectification and the right to erasure [38].

4.2.1.5 *Integrity and Confidentiality*

Article 5 Sec. 1 lit. f GDPR introduces the principle of integrity and confidentiality in the processing of personal data. Precisely, it states that "*processed in a manner that ensures appropriate security of the personal data, including protection against unauthorised or unlawful processing and against accidental loss, destruction or damage, using appropriate technical or organisational measures;*" [33]. Being a very general principle, this obligation does not seem to generate any particular in relation to blockchain technology [35].

4.3 TENSIONS BETWEEN BLOCKCHAIN AND THE GDPR

In this section, we review the major conflicts that arise whenever we try to map a blockchain with the Regulation. First, we present various alternatives to identify the roles of the data controller and data processor in the different types of blockchain. Second, we review the tensions regarding the exercise of rights and freedoms of a data subject. Third, for completeness, we briefly describe the possible problem of personal data transfer to third countries or in international organisations. Even though we present all the mentioned tensions, we shall specify that the third one falls outside the scope of this research, and it is presented for completeness purposes. Instead, the main focus of our discussion will be on the second conflict mentioned above, namely the exercise of data subject's rights and freedoms. In particular, we will focus on the right to rectification and the right to erasure. Indeed, as we discussed in the previous section, they appear as being at odds with the way a blockchain is structured [38]. Since the data controller and the data processor are in charge of satisfying the requests of a data subject, their role in the blockchain is in close relationship with the ability of a data subject's to exercise its rights. Moreover, the allocation of responsibilities of the data controller and data processor is fundamental to allow data subjects to exercise their rights [39]. Therefore, we shall discuss their mapping with the blockchain as well.

It has bee argued that the tensions between the GDPR and blockchain technology are due to two main factors [35]. First, GDPR has based its principles on the assumption that it is always possible to identify an entity - the data controller - to which a data subject can refer when exercising its rights [35]. While in a centralised environment it is most of the time possible to identify a data controller unequivocally - or a set of them under the definition of joint controllers - the identification of such figure is much more complicated in a distributed scenario. Second, GDPR prescribes that data can be modified or erased when necessary. However, modification and deletion on a blockchain are designed to be as hard as possible with the goal of preserving the integrity of the ledger and creating trust in the network [35].

4.3.1 *The roles of Data Controller and Data Processor*

Data controller and data processor are two fundamental roles identified by the GDPR when it comes to the accountability of the processing of personal data. Controllers (and processors as well) are obliged to implement appropriate security measures and to demonstrate that processing operations are compliant with the principles of the Regulation [39]. In a centralised scenario, where a central entity establishes the means and purposes for processing personal data, it is possible to identify both roles clearly and objectively. Similarly, one or multiple entities (still objectively identifiable) are responsible for the actual processing of personal data. However, the definition becomes inadequate in those scenarios in which the processing of personal data happens in a distributed fashion [39].

According to the opinion of the Article 29 Working Party on the concept of controller and processor [40], determining the purposes and means of data processing is equivalent to determine *why* and *how* of certain processing activities [35, 39]. Precisely, the concept of controller should be considered as a functional concept, meaning that it is "*intended to allocate responsibilities where the factual influence is, and thus based on a factual rather than a formal analysis*" [40]. Therefore, it is not possible to provide a formal decision on who is the data controller because (i) a formal definition of a controller laid down by law might be lacking or (ii) because it might be that the formal appointment does not reflect the factual responsibilities. Any identification of the controller through a contract can be overturned in case the factual responsibility is found to be on a different party than the one initially identified [35]. Although the *why* and *how* of data processing (*purposes* and *means* respectively) seem to have an equivalent importance, [40] appears to give a higher importance to the purposes over the means. In fact, the determination of the purposes is a responsibility of the data controller, whereas the determination of the means of processing might be delegated to the data processor [39, 40]. Additionally, according to [40], the determination of the purposes of personal data processing automatically triggers the qualification of an entity as the data controller.

Following the discussion presented in [39], the opinions of [40], and the work of Finck [35, 36], we dive into mapping the roles identified in the GDPR in the participants of a blockchain network. As stressed by Duarte, to understand who determines the purposes and means of data processing, it is necessary to consider both how personal data is processed and the structure and governance of the blockchain [39].

4.3.1.1 *The role of the Data Controller*

GDPR prescribes that a data controller must be identified. The data controller is the one entitled to decide the purpose of processing, which data should be processed, for how long, who can access, and what security measures need to be taken. The role of the data controller is thought and designed to fit a centralised scenario [38, 39]. However, in a distributed environment such as a blockchain where several parties participate, it is challenging to identify responsibilities and assign roles [39]. Intuitively, the mapping with a permissionless blockchain is more laborious compared to the mapping with a permissioned one because there are no limitations on who can join the network, and validation rules are often decided by developers when the technology is built. Instead, when a permissioned blockchain is used, the assignment of the role to participants simpler due to the more centralised governance of the blockchain. It has been proposed that developers might be identified as data controllers. However, developers often do not participate in the consensus protocol, and validation rules might be changed by validators provided that they reach enough consensus. Therefore, developers seem to exercise a minimal influence on the means of processing and no influence over the purposes of processing [35].

DATA CONTROLLER IN PUBLIC - PERMISSIONLESS BLOCKCHAIN
In public and permissionless blockchain, the control on the platform is distributed, and there are no centralised authorities that supervise

the transaction process and the inclusion of additional nodes, users, or miners. Hence, it is not easy to map a centralised definition of the data controller to the highly distributed setting of such a blockchain. Moreover, there is no comprehensive agreement in the literature regarding the assignment of this particular role in distributed ledgers. According to Duarte, the difficulty in determining the data controller comes primarily from two factors: (i) there is a broad and unsupervised number of actors that influence the management of personal data, and (ii) the purposes of the processing might be different in the same platform [39].

Bacon et al. present a method to overcome these difficulties. The authors propose to assess the problem from a micro-level perspective. Precisely, they sustain that the assignment of the role of the controller should be based on a transaction-level analysis. According to both [41] and [39], this micro-level perspective suits better compared to the macro-level one that suggests identifying the controller based on the study of the blockchain as a whole infrastructure. They argue that, in the case of a micro-level perspective, it is the users itself that determines the purposes and means of processing through its decision to join and use the blockchain platform. A recent European Parliament report expresses similar opinions. It qualifies users as both data controllers and data processor because they upload data on the ledger and store a copy of it. The view of users as the data controller is, however, far from being straightforward and unproblematic. For our research, we will not dive into the consequences of users as data controllers. The interested reader can find a structured and comprehensive discussion of this situation in [35].

In a scenario where users are considered data controllers, miners exercise control over the means of processing. However, they do not exercise control over the purposes, which is the main criteria to establish the role of the controller [39]. On the contrary, it might be that nodes - those who store the ledger and participate in the validation of blocks - can be seen as joint controllers.

The assignment of roles, however, might not be accurate in all platforms. Indeed, there are cases in which nodes and miners can set up additional purposes and modify the means of processing. In those cases, both should qualify as joint controllers [39].

DATA CONTROLLER IN PRIVATE - PERMISSIONED BLOCKCHAIN

Unlike public and permissionless blockchain, a central entity - or a group of well-defined entities - that decide the purposes and the means of the processing of personal data governs private and permissioned blockchain. In such a situation, two contrasting opinions arise. The first is due to Finck who proposes to identify as data controller the platform itself and to assign responsibilities to the organisation that governs the platform. On the contrary, Bacon et al. sustain the micro-level perspective we discussed above and argue that assigning duties to the platform cannot be done when the problem is analysed from a macro-level perspective only. In their opinion, the data controller should be the users, and the centralised authority should be assigned the role of the data processor.

The mapping proposed by Duarte summarises the two previous opinions. In his view, it is crucial to follow the suggestions of the Article 29 Working Party stating that the allocation of responsibilities should be based on the factual influence the various parties exercise in the process [40]. Hence, we can identify two cases. In the first case, users have limited or no control over the platform they are allowed to use to perform their activities. Therefore, we can argue they do not exercise sufficient control on the choice of the platform to have a factual influence over the purposes and means of processing. Clearly, in this situation, the influence is exercised by the central authority governing the blockchain that should be assigned with the role of the data controller. In the second case, instead, users have the possibility of choosing the platform on which to run their business. Therefore, they have factual influence over purposes and means of processing, and they should be assigned with the role of the data controller.

DATA CONTROLLER IN CONSORTIUM BLOCKCHAIN Similarly to a private blockchain, a consortium blockchain is permissioned and governed by a set of nodes belonging to different companies. The set of nodes might be interpreted as a centralised entity that exercises factual influence by determining purposes and means of processing. Therefore, they should be identified as data processors. Ibáñez et al. proposes that, in case a community jointly decide the validation and the processing rules, it is possible to fit the entities in the community under the definition of joint data controllers [38]. However, quoting the work of Finck, "*it is far from clear what degree of involvement is necessary to be qualified as a joint controller*" as different cases law expressed contrasting opinions on the subject Finck et al.

4.3.1.2 *The role of the Data Processor*

Similar to the role of the data controller, a data processor should be identified as well. However, unlike the previous role, a data processor might not exist at all. Indeed, its existence depends on the decision of the data controller on outsourcing the processing of personal data. To be qualified as a processor, one has to be a separate legal entity with respect to the controller and to carry out the processing of personal data on behalf of the controller. As the identification of the data controller, the assignment of the data processor to one entity in a blockchain is far from being clear.

DATA PROCESSOR IN PUBLIC - PERMISSIONLESS BLOCKCHAIN We previously discussed the micro-level perspective proposed by Bacon et al. to assign roles in public and permissionless blockchain. Following the reasoning, the user itself decides the purposes of the processing. Hence, it should be qualified as the data controller. In this situation, nodes and miners exercise control over the means of processing as they control the validation process. Therefore, they should be identified as the data processor. Instead, in case nodes and miners voluntarily exercise control over the purpose of processing by proposing new ones, there is no data processor in the network as they qualify as joint data controllers.

DATA PROCESSOR IN PRIVATE - PERMISSIONED BLOCKCHAIN We presented before the mapping proposed by Duarte to identify the data controller in private and permissioned blockchain. In the first case he identifies, users have limited or no control over the platform they are allowed to use to perform their activities. Instead, the central authority governing the blockchain exercises the influence and that should take on the role of the data controller. In this scenario, no processor should be identified. In the second case, instead, the users have the possibility of choosing the platform to run their business. Therefore, they have factual influence over purposes and means of processing, and they should take on the role of the data controller. Accordingly, the nodes belonging to the private organisation offering the service to the users should qualify as data processor [39].

DATA PROCESSOR IN CONSORTIUM BLOCKCHAIN According to the above discussion, the set of nodes in a consortium should be regarded as the data controller. In case additional nodes and miners that do not belong to the initial set of nodes are added to the blockchain, they should represent the data processor. They do not exercise control over the purposes of the processing, but they do exercise control over the means by participating in the validation process and the consensus mechanism.

4.3.2 *The exercise of Data Subject's Rights*

In Section 4.2, we presented the six core principles of the GDPR, and we discussed their mapping on a blockchain architecture. Some of these principles do not present particular concerns when the supporting technology is a blockchain. However, at first sight, it seems that some other provisions trigger both technical and legal challenges. In particular, limitations appear to happen concerning the right to access, the right to rectification, and the right to erasure. Additionally, when using public and permissionless blockchain, the challenges may be higher to overcome.

Although the right to access is not our primary focus, we note that there might be limitations regarding the exercise of this right from data subjects. Often, personal data stored on a blockchain is encrypted or hashed. Therefore, it might be hard to provide the data subject with the exact knowledge of what personal data is stored and processed [39]. Similarly, it may be hard to ensure the data subject has a copy of its data because a node can provide its local copy of the blockchain, which is not guaranteed to be equivalent to the ones stored in all the other nodes [39].

Stressing the fact that it might be hard for data subjects to exercise their rights in a blockchain, we should acknowledge that data controllers in a private or consortium blockchain are in a better position to facilitate data subjects' exercise of their rights [35]. As we discussed above, in private blockchain users or nodes should be regarded as data controllers, whereas in consortium blockchain nodes are most likely to be assigned with this role. Since both users and nodes possess more control over the data that is processed and how it is processed, the exercise of the rights above should be more straightforward.

Even though data subject's themselves can be qualified as data controllers in many applications, challenges remain on how to exercise the right to rectification in the so-called immutable structure of a blockchain [39]. In public and permissionless blockchain, there are three main obstacles to the exercise of this right. First, a node might be able to modify its local copy of the blockchain but, since the adopted blockchain is the one accepted by the majority of the participants, such modification is irrelevant [39]. Second, it may be impossible to identify enough the nodes in the network to create a fork. Third, the process of creating a fork requires a high level of coordination. We argue that a fork should be considered as the last possible solution to apply a modification due to the necessity of reaching an agreement off-chain before the forking process. In a more closed environment, such as a private or consortium blockchain, the set of nodes can be identified in a more straightforward way and agreement on a modification is easier to obtain.

Similar challenges to those identified in the case of the exercise of the right to rectification can be found in the process of exercising the right to erasure. Indeed, it is tough - and almost impossible after a sufficient set of blocks have been added to the chain - to change or delete data on a blockchain. As before, one may argue that it is possible to use the forking process to remove data from a blockchain. In agreement with [39], we stress once again that a hard fork should be an exceptional event and should not be considered as the preferred method to allow data subjects to exercise the right of erasure.

Before diving into the presentation of technical and legal approaches to address the problems connected with the rights to rectification and erasure, it is worth noticing that the meaning of *erasure* is open to interpretation [35, 39]. To further complicate the matter, we should highlight the fact that the definition of erasure remains unsettled. According to the European Court of Justice (ECJ), erasure of personal data might include its simple removal from a search index without requiring data to be deleted. Accordingly, someone might argue that the deletion of the encryption key used to encrypt data or the suppression of the link that enables a party to access personal data can be regarded as sufficient measures to claim compliance with the Regulation. However, there are no official decisions on whether the steps above could be considered enough. The Court of Justice of the European Union (CJEU) seems to suggest that the erasure should be equivalent to the destruction of personal data. However, this opinion is in contrast with the one expressed by the ECJ.

Last, we stress that, as discussed in Section 4.1.1.1 and Section 4.1.1.2, both transactional data and public keys stored on a blockchain are most likely to be considered as personal data even when encrypted or hashed. Further, the anonymisation threshold imposed by the GDPR is currently unlikely to be satisfied by the current state-of-the-art algorithms.

4.3.2.1 *Technical approaches to the problem*

Chapter 3 focuses on solutions that involve the modification of the blockchain. However, it is worth noticing that it is not the only proposed approach. Most of the work done so far focuses on the preservation of transaction privacy, whereas the general problem of data protection received increasing attention only recently [38].

Some naive approaches to address transaction privacy suggest to create a fresh key pair for each transaction to reduce the likelihood of detecting pattern by analysing transactions. Others suggest the use of TOR as a mean to hide the real identity of the transactor. We argue that their implementation might be used to achieve a higher level of privacy. However, they are far from attaining data protection and from being compliant with the Regulation. More sophisticated methods aiming at complying with the GDPR propose the integration of cryptographic techniques into validation protocols to hide potentially private information [38]. For instance, ring signatures allow a party to sign a transaction with its private key and another party to verify that the transaction has been signed with a key associated with a group of keys, without revealing its exact identity. Other solutions take a step further and try to hide the content of a transaction while permitting validators to check the regularity and authenticity of it. Cryptographic techniques used to achieve the mentioned goal are ring-confidential transactions and zero-knowledge proofs. According to [38], there are very few technical proposals tailored to allow data subjects to exercise their right of erasure. Ibáñez et al. argue that this lack of technological solutions is driven by the belief that, once a sufficient anonymisation level of personal data is obtain on a blockchain, there is no need to check compliance because data is not personal anymore once it has been anonymised.

4.3.2.2 *Legal approaches to the problem*

One of the most straightforward and most advocated solutions coming from the legal environment proposes to store personal data off-chain and to save on the blockchain a hash used as a link to an encrypted entry in a database where the data is stored [36, 38]. In agreement with [38], we argue that such a solution renders the blockchain useless as it brings back some of the problems that might justify the use of a blockchain as a mean to ensure integrity and availability.

The proposal to store data off-chain is also seen as a solution to implement the right to erasure. If a data subject requests the erasure of its data, it might be enough to delete the data in the database and leaving the hash link as pointing to an empty location. Similarly, the right of rectification can be satisfied by creating a new transaction that invalidates the hash link to the incorrect data and issues a new link to the correct version. However, in both cases, the hashes of personal data will remain on the blockchain. At the time of writing, many pieces of research clarify that hashing should be regarded as a pseudonymisation technique that does not prevent the Regulation from applying [36].

We previously discussed the fact that the term *erasure* might be open to interpretation, and the right to erasure is not an absolute right. However, as noted in [38], the possibility that a semantic invalidation of data being stored on the blockchain can be considered enough depends on the sensitivity of the information. Indeed, there is an example in which a court ruled in favour of a complete deletion asked by a data subject. This means that it is not always possible to rely on a semantic invalidation of data (similar to the one obtained by issuing a new transaction that obfuscates the old one) and that in some situations a hard deletion is required [38].

Another possibility for erasure might be the use of strong encryption so that data becomes anonymous. Unfortunately, a precise legal definition of a sufficient level of encryption is lacking, and the threshold imposed by the GDPR to achieve anonymity is notoriously very high. Moreover, according to [36], none of the encryption algorithms developed so far has been recognised as able to achieve such level. As argued by Ibáñez et al., the deletion of both the key and the original data will likely put the controller in a safer position compared to the one obtained by deleting the key only [38].

4.3.3 *The Transfer of Personal Data to a Third Country*

Personal data can be moved to third countries, i. e., countries outside the European Union, only after an accurate evaluation of the safeguards and security protections provided by the state under investigation. In particular, when deciding on whether a third country possesses enough safeguards, one should investigate if the country "*offer[s] guarantees ensuring an adequate level of protection essentially equivalent to that ensured within the Union*" [33]. In case there are sufficient safeguards, transfer of personal data to a third country does not require any additional process. In case safeguards are not enough, transfer of personal data might still be allowed if either the controller or the processor can provide those safeguards. As the location of the nodes in a blockchain might be hard - or even impossible - to control, the transfer of personal data to third countries in a blockchain can generate conflict with the GDPR [39].

Similarly to the other conflicting situation we discussed above, the situation is harder when the blockchain is public and permissionless. On the contrary, when the blockchain is private and centrally managed or permissioned and operated by a consortium, it is simpler to determine the location of the participants as they are controlled before joining the network. Nevertheless, participants may be located outside the European Union. In those cases, a legal ground has to be found to process personal data in the blockchain and to move data to nodes located outside the Union [39].

4.4 REQUIREMENTS

In this section, we present the legal and technical requirements that we collected through our review process and a set of interviews we conducted with legal and technical experts. First, we present the elements that will enable our prototype to comply with the Regulation. Second, we describe the technical requirements of a modifiable blockchain that respect the legal requirements.

4.4.1 *Compliance Requirements*

In this chapter, we introduced the GDPR as well as its core principles and how they map with a blockchain. Due to the fact that the Regulation applies to personal data only, we discussed what type of data stored on a blockchain might be considered as such. We discovered that, although encrypted or hashed, both transactional data and public keys stored on a blockchain constitute personal data in most of the situations. Therefore, data subjects have the right to request the rectification of incorrect data and the erasure of data according to Art. 5 GDPR. However, we have already noticed that the right to erasure is not absolute. GDPR prescribes six different grounds on which data subjects can base their exercise of the right to erasure. However, according to [38], at least two of them are challengeable. The first regards the fact that personal data is no longer necessary for the purposes for which it has been collected. Ibáñez et al. argue that on a blockchain it is always required to process data. Therefore perpetual processing is needed for the purposes for which data has been obtained [38]. The second ground that can be challenged regards the withdrawal of consent. The authors propose that, under the assumption that the data subject gave the approval for processing on a blockchain, consent must be perpetual as well and cannot be revoked. Without claiming our support for or against this interpretation, we argue that there is a lack of clarity around the possibility to challenge the grounds identified by the Regulation and, to the best of our knowledge, [38] are the only ones to support this argument.

We also noticed how the concept of *erasure* is open to interpretation, and many sources point to encryption and hashing as potentially viable solutions to achieve compliance. However, the same sources stress the lack of legal definitions regarding what could be considered as a *sufficient* level of encryption. Additionally, most of them agree on the classification of encrypted and hashed data as personal data, acknowledging that the presence of a *hard-delete* functionality might help the data controller in being compliant and demonstrating its compliance with the Regulation.

To conclude this section, we summarise our discussion, and we provide the answer to *SQ3* of *RQ1*. Although the fact that hashed data is deemed to be personal data despite the very low likelihood of recovering the initial data could be surprising, we acknowledge the absence of legal definition or court decisions stating that hashing can be considered as a sufficient measure for deletion. Moreover, we recognise that encrypted and hashed personal data are still being considered personal by many authors in the literature. Therefore, we argue that, in case personal data, in plain text, encrypted form, or hashed is not stored on the blockchain there is no need to propose modifications to a blockchain to comply because the ledger does not contain personal data. However, we find it hard to think of the usefulness of a blockchain in which there is no link between the data stored off-chain and the data stored on the ledger. Therefore, to justify the use of a blockchain, at least hashes of personal data might be stored on the ledger to enhance the data integrity. Moreover, the public keys of users are likely considered personal data as well. In those cases, introducing a modification feature to the blockchain is a possible way to comply with the Regulation.

4.4.2 *Technical Requirements*

This section presents the technical requirements that we identified through the literature review and the interviews. The first requirement that we discuss is the ability to modify a single transaction - and we refer to as transaction-level modifiability - in contrast with the ability to alter a block - which we refer to as block-level modifiability. As we will present, the possibility to modify a block presents limitations due to the unrealistic assumptions the approach supposes. Second, we present the evidence of modification. Arguably, the ability to alter a blockchain ledger poses the risks of weakening its integrity and immutability. Therefore, we thought that the public evidence of modification could avoid the loss of *tamper-evidence* as any modification will still be visible to all participants. Third, we introduce what we call proof-of-redaction. Although it is not possible to have complete certainty about the fact that a node deleted the information, we argue that the presence of such a proof could render each node accountable for its actions and could serve as evidence of the fact that the request was processed by the node.

4.4.2.1 *Transaction-level modifiability*

From the analysis of the existing solutions, we discovered that a few solutions provide transaction-level modifiability. We define *transaction-level modifiability* as the ability to modify a or delete a single transaction without impacting a whole block. The inability to edit or delete a single transaction is an explicit limitation of most of the proposed scheme. Some of them assume either that a block includes a single transaction or that a block contains transaction issued by an individual user. We argue that these assumptions are far from being real in many of the current blockchain implementations. Moreover, these are not realistically feasible as they will introduce additional overhead in terms of storage and scalability.

4.4.2.2 *Evidence of modification*

Immutability is a highly-praised feature of blockchain. Although referring to a blockchain as tamper-proof and tamper-evident is more correct, we group those properties and refer to those as *immutability* for simplicity. The immutability of a blockchain is undoubtedly one of its strongest features and likely the one that makes a blockchain unique compared to other types of distributed ledgers. The robust hash chain structure perfectly achieves the goal of maintaining an *immutable* log of transactions where parties do not trust each other and do not want to engage in a trust relationship with a third-party that supervises and grants the integrity of the transaction process. However, there are some other scenarios in which this feature presents itself in contrast with different requirements. For instance, the presence of unwanted or illegal material on a blockchain might be detrimental for the participants of the network. In those case, where laws or regulations are infringed, it may be possible to remove part of the content from the blockchain so that the system can be operated a compliant fashion. However, modifying the blockchain inevitably weakens the tamper-evident property. Precisely, in case a modification happens, it can happen without leaving any traces. We stress that to reduce the impact of a modification, it should be evident that an alteration has occurred. We believe a modification should be justified so that auditability and the trustworthiness of the blockchain can be maintained to a sufficient level to justify the use of a blockchain even when its modification is permitted.

4.4.2.3 *Proof of Redaction*

The Proof-of-Redaction (PoR) is a mechanism that allows a party to agree on the proposed redaction and to show to the network that a redaction happened. Before a modification or deletion is executed, the network agrees on a redaction submitted by a peer of the data controller - or one of the data controllers in case there are many. After the computation of the collision, the peer in charge of computing it proposes a transaction specifying how to redact the ledger and which random value could be used to compute the collision. If the proposal is validated by the other peers, it is published as a way to show that the network agreed on a precise way to modify the ledger.

5 HOW TO CHANGE THE IMMUTABLE

In the previous chapter, we focused on building a solid understanding of the legal aspect of the problem. We concluded that in many situations, both encryption and hashing are not sufficient to anonymise personal data. Therefore, if a data subject request for modification or deletion is approved on a legal ground, that decision needs to be applied on the blockchain. We identified three requirements that we believe are sufficient to design a redactable blockchain that does not weaken an existing architecture, namely transaction-level modifiability, evidence of modification, and proof of redaction. Using a structural approach, we tackle the problem of blockchain immutability with a small adjustment to the block creation process, and we strive for maintaining the tamper-evidence property through the introduction of the proof of redaction.

This chapter describes the architectural design of our proposal of redactable blockchain that leverages chameleon-hash functions with ephemeral trapdoor. We begin with the introduction of the design of our redactable blockchain, and we proceed with the presentation of its technical building blocks. We continue by representing and commenting on the flow of a data subject's request and analyse the two possible cases that such a request creates in the network. After the presentation of the building blocks and the flow of transactions, we combine the two elements, we demonstrate where the building blocks fit in the architecture and how we incorporate them to produce our final design. Moreover, we provide an implementation in Go of the chameleon-hash function with ephemeral trapdoor. The evaluation of whether the design meets the declared objectives and the results of applying the modifications in terms of performance and core properties is discussed in Chapter 6.

5.1 DESIGN

In this section, we present the architectural design of our solution that provides the ability to modify or delete a transaction on a blockchain, based on the request of a data subject. Depending on the request, the blockchain will update the wrong transaction data and produce evidence of modification. Furthermore, each node of the network that was identified as either the data controller or the data processor publishes a proof-of-redaction. We begin by introducing the various building blocks of our solution, and we proceed by combining them into the blockchain. We continue by presenting the flow of a data subject's request into our architecture. Last, we show how to incorporate the building blocks into an existing blockchain architecture to achieve the desired capabilities.

5.1.1 *Building Blocks*

This section provides the details of the building blocks of the solution. First, we introduce the concept of chameleon-hash functions. We proceed by showing why early designs of such functions would not be suitable for our architecture due to a security flaw called key exposure, and we present their strengthened concept with ephemeral trapdoors. Following the presentation of chameleon-hash functions, we present commitment schemes and secret sharing schemes that will be used during to manage the ephemeral trapdoors and to execute the redaction procedure.

5.1.1.1 *Chameleon-Hash Functions*

A chameleon-hash is a particular type of hash function parametrised by the presence of a public key [24]. Firstly introduced by Krawczyk and Rabin in [42], chameleon hashes are designed to be collision resistant for any party except for those that possess the trapdoor, i. e., the secret key related to the public key. Chameleon-hash functions are constructed so that the knowledge of the public key allows a party to compute the hash function. In contrast, the knowledge of the trapdoor makes the process of finding a collision for every given input much more straightforward.

Let us now introduce a more formal definition. A chameleon-hash function $\mathsf{CH}()$ is a non-standard hash function associated with a user U which holds a pair of keys: a public key pk and a secret key sk generated from a given key generation algorithm [42]. Given a message m and a random string r as inputs, the function $\mathsf{CH}_U(m, r)$ generates a hash value that satisfies the following properties:

1. **Collision resistance**. The chameleon-hash function $\mathsf{CH}_U()$ is said to be *collision resistant* if it is computationally infeasible to find two input pairs m_1, r_1 and m_2, r_2, $m_1 \neq m_2$ such that $\mathsf{CH}_U(m_1, r_1) = \mathsf{CH}_U(m_2, r_2)$.

2. **Trapdoor collision**. A chameleon-hash function $\mathsf{CH}_U()$ holds the *trapdoor collision property* if there is an efficient algorithm, given the secret key sk, an input pair m_1, r_1 and an additional message m_2, computes a value r_2 such that $\mathsf{CH}_U(m_1, r_1) = \mathsf{CH}_U(m_2, r_2)$.

3. **Uniformity**. A chameleon-hash function $\mathsf{CH}_U()$ is said to be *uniform* if all messages m induce the same probability distribution on $\mathsf{CH}_U(m, r)$ where r is chosen uniformly at random.

To summarise, a chameleon-hash function offers the same collision resistance guarantees that a cryptographic hash function offers unless the party computing the collision knows the trapdoor - the secret key sk - associated with the public key *pk* used to compute the hash. Many of the early designs of chameleon-hash functions, however, suffer from a problem called *key exposure*. The problem, described in both [43] and [44] by Chen et al. and Ateniese and de Medeiros respectively, consists in the ability of every party to acquire the knowledge of the trapdoor when the first collision is computed by the holder of the secret key. Therefore, when the party knowing the trapdoor computes the first collision, it becomes possible for any other party to recover the secret key and to compute arbitrary collisions on all the hashes derived from the same public key. While in other scenarios problem could be only marginal, it is not acceptable for our application. Every peer in the network will gain knowledge of the trapdoor associated with all the transactions issued by the same user. Therefore, it will be able to compute collision for all the mentioned transactions.

Chen et al. and Ateniese and de Medeiros independently provided an implementation of a chameleon-hash function that does not suffer from key exposure based on different hardness assumptions. A recent development on key-exposure free chameleon-hash function is due to Camenisch et al. who envisioned the addition of a second - ephemeral - trapdoor [45]. The presence of a freshly-chosen second trapdoor limits the ability of the first trapdoor's holder because it renders not possible to compute arbitrary collision with the knowledge of a single secret. Moreover, the second trapdoor is different for every hash computed with the same public key - and therefore associated with the same secret key. Such construction avoids the problem of giving every party the ability to compute arbitrary collisions once the first collision has been computed. Indeed, the computation of a collision by an authorised party does not leak any information on the second trapdoor used to find the collision.

5.1.1.2 *Commitment Scheme*

A commitment scheme is a cryptographic primitive that allows a party to commit to a value without revealing the actual value while maintaining the ability to disclose the value later. Commitment schemes are designed as a two-step process:

1. during the *commit* phase, a party chooses a value and produces a commitment to that value;
2. during the *reveal* phase, the committed value is disclosed, and the commitment can be checked.

Commitment schemes posses two security properties, namely **hiding** and **binding**. The *hiding* property ensures that, at the end of the commitment phase, no receiver can learn information on the committed value. The *binding* property, instead, ensures that at the end of the reveal phase, the party computing the commitment cannot disclose a value different from the one it committed to.

5.1.1.3 *Weighted and Verifiable Secret Sharing*

Once the key-exposure issue of early chameleon-hash designed is solved, we ran into the problem of distributing the ephemeral trapdoor so that collisions can be found only when the party holding one first trapdoor is allowed to do so. Secret sharing schemes enable us to distribute the ephemeral trapdoor among different parties so that each party cannot compute the secret unless it combines its share with others until a threshold is reached.

Secret sharing, introduced independently by Shamir and Blakley, refers to a method used to divide a secret S into shares s_i and to distribute the shares among a set of participants. In secret sharing schemes, a party - the dealer - divides the original secret S into shares s_i and distributes the shares among the participants. Each participant cannot reconstruct the entire secret if not by joining its share with others' shares. Moreover, the secret can be reconstructed only when a sufficient number of parties combine their shares. Formally, a (t, n)-threshold secret sharing scheme [46] is a scheme where

- a secret S is shared among
- n players
- which receive a share s_i of the secret S, with $i \in \{1, .., n\}$
- such that the secret S can be reconstructed from at least t shares.

Verifiable secret sharing was introduced as a modified version of secret sharing, where every participant can detect malicious actions either by the dealer or by other participants. For instance, using verifiable secret sharing a party can discover whether the share it received from the dealer is correct and belongs to the set of shares that allow the secret to be reconstructed. Similarly, every party can verify that another participant is disclosing a valid share during the reconstruction of the secret, and it is not lying on its share to gain knowledge of the others' share. To construct a verifiable secret sharing, each share should contain a public commitment through which it is possible to prove the correctness of the received share.

In weighted secret sharing schemes, each share s_i of the secret S is associated with a weight w_i. Instead of requiring at least t shares to reconstruct the secret, weighted secret sharing schemes require that the sum of the weights of the parties participating in the reconstruction is greater than or equal to the threshold.

5.1.2 *Request Flow*

In this section, we show how the flow of a data subject's request happens. We begin by presenting the trivial case in which the request is directly deemed acceptable by the data controller(s) after the analysis of its Data Protection Officer (DPO). We proceed by showing the case in which the data subject needs to appoint the Data Protection Authority (DPA) due to the data controller refusing to accept the request without sufficient justification. We stress that the implementation of the system covers the case in which a request has been granted. Precisely, a request to delete or rectify data should be performed by a data subject to the data controller directly without the use of the blockchain. The DPO should carefully evaluate a request and, in case there is a dispute, the DPA might play a significant role as well. While it should be possible to encode decision rules in smart contracts, the current imprecise and uncertain situation reduces the ability to execute automated processing of a data subject request.

5.1.2.1 *Case 1: Data Controller Accepts*

The first and most trivial case is the one in which the data controllers accepts the initial request of the data subject. In such a scenario - represented in Figure 6 - there is no need to include the DPA as the request is deemed correct since the beginning. Therefore, once the data controller accepts the request, it performs its action (modification or deletion), it publishes a proof of action, and it forwards the request to the data processor(s). Since a data processor may or may not exist, the arrows in the figure are dotted.

In many cases, the request performed by the data subject will be directed to the Data Protection Officer. The Data Protection Officer (DPO) is a role that has to be identified by the data controller, and it is required by the Regulation to oversee the data protection strategies of a company to ensure compliance with the GDPR. Since every organization does not need to identify a DPO, there might be cases in which the entity is not present. However, in those cases in which sensitive personal data is managed, the DPO is most likely to be present. Therefore, we assume that the data controller will process a data subject's request through the DPO.

A similarly trivial situation is the one in which the DPO objects the request of the data subject and provides grounds to justify its rejection. In case the data subject considers the decision correct under the offered justifications, the process terminates. The situation in which the data subject decides to challenge the decision is described in the following section.

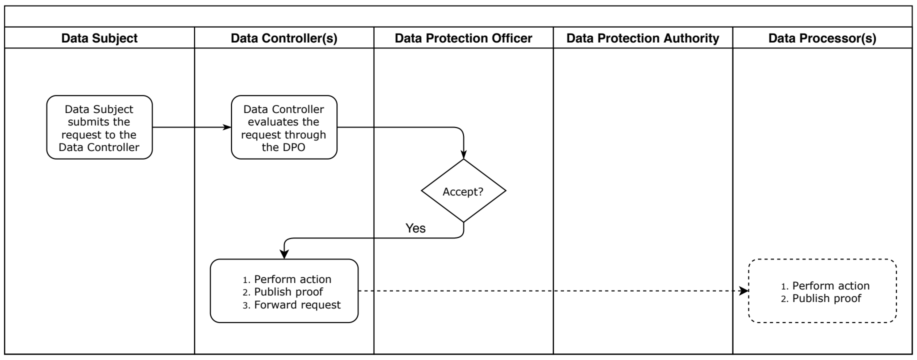


Figure 6: Data controller accepts the request of the data subject

5.1.2.2 *Case 2: Data Subject appoints the Data Protection Authority*

The Data Protection Authority (DPA) - or Supervisory Authority - is an entity that each Member State should identify to monitor the application of the GDPR and to protect the rights and freedoms of data subjects. Among the various tasks of which the DPA is responsible for, Art. 57, Sec. 1, lit. e and Art. 57 Sec.1 lit. f identify two crucial responsibilities in relation to the figure of the data subject. Precisely, the first prescribes the responsibility of providing data subjects with the information they need concerning the exercise of their rights [33], whereas the second states that the DPA should handle complaints filed by the data subject - or any other organization or association - and update the applicant regarding the progress and outcomes of the claim [33].

Being in charge of handling data subject's complaints, the DPA could receive a data subject's request to evaluate the protection of its rights. The flow is similar to the once where there is a DPO. However, instead of submitting a complaint to the DPO, the data subject files a complaint to the DPA. Similarly, the DPA evaluates the complaint to understand whether the data subject's rights are being respected or violated by the data controller. If the DPA deems the request acceptable, the authority informs the data controller which proceeds as in the case above: the action of the request is executed, a proof is published, and all the data processors - if any - are informed of the decision and are instructed with the actions to take to ensure compliance. Once a data processor receives a notification from the data controller, it takes steps to execute the action, and it publishes its proof of deletion. The flow is depicted in Figure 7.

5.2 INTEGRATION

In this section we focus on how we integrate the building blocks and the cryptographic primitives presented in Section 5.1.1 to support

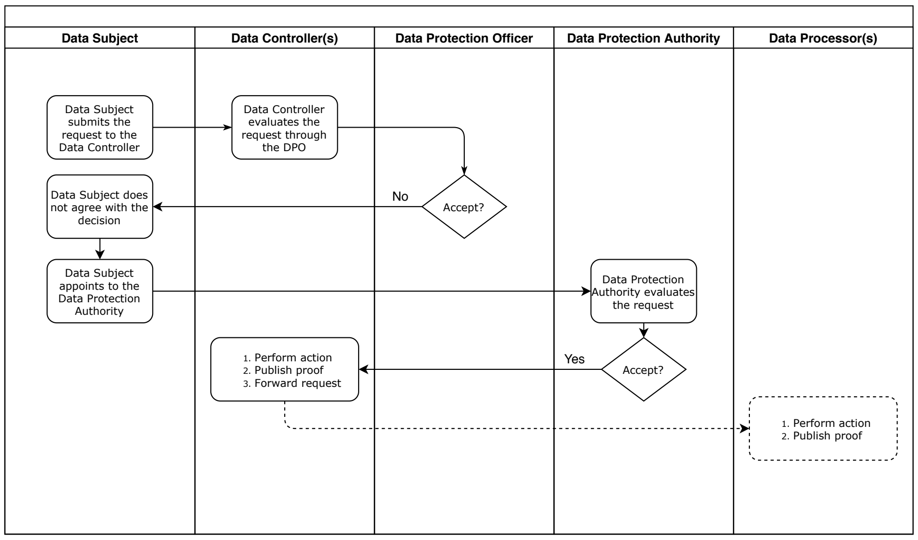

Data Subject
Data Controller(s)
Data Protection Officer
Data Protection Authority
Data Processor(s)
Data Subject submits the request to the Data Controller
Data Controller evaluates the request through the DPO
Accept?
No
Data Subject does not agree with the decision
Data Subject appoints to the Data Protection Authority
Data Protection Authority evaluates the request
Accept?
Yes
1. Perform action
2. Publish proof
3. Forward request
1. Perform action
2. Publish proof

Figure 7: Data subject appoints the Data Protection Authority

the flow depicted in Section 5.1.2 into an existing blockchain, namely Hyperledger Fabric.

For simplicity, we will start by addressing the case in which there are no multiple transactions linked together. This assumption will reflect on the ability to modify or delete a single transaction without the need for implementing a cascade effect on all the others that depend on the modified/deleted one. We will briefly discuss the more complex case later in the chapter.

5.2.1 *Chameleon-Hash Function into the Block Creation Process*

To make an existing blockchain modifiable - or redactable - the first change we should implement is the modification of the hash function. We have already observed in the review of the related work - Chapter 3 - that different methods have been proposed. We will briefly recap some of them and discuss which is, in our opinion, the best choice.

5.2.1.1 *Chameleon-Hash Integration in Related Work*

The work of Ateniese et al. is perhaps the most representative one, as well as the first one, and received the attention of the consulting company Accenture which submitted the first patent of a modifiable blockchain. In their work, the authors proposed the use of chameleon-hash functions as a substitute for the hash function used to chain the blocks in a blockchain [23]. A finer-grained application of the same concept is proposed by Derler et al. in [24] addressing the limitation of a block-level modification. Similarly to the work of Ateniese et al., the authors substitute the existing hash function with a new primitive they dubbed chameleon-hash with ephemeral trapdoors. However, instead of targeting the hash function used to connect blocks, Derler et al. focused on the hash function used to compute hashes of transactions. Note that, while the hash function used to chain the blocks and to compute hashes of transactions might be the same, the scope of the modification differs a lot. Therefore, we will base our design on the work presented in [24].

5.2.1.2 *Chameleon-Hash Functions in Hyperledger Fabric*

Unlike many other blockchain architectures, Hyperledger Fabric uses a particular design of transaction flow. The first phase of the transaction flow is represented by a user that submits a transaction proposal to various peers in the network through a chaincode function call. Each peer that received the transaction executes the chaincode function and returns the result of the execution as a transaction proposal response. When the application has collected enough consistent and signed transaction proposal responses, it forwards the transaction containing the endorsed responses to the ordering service. The ordering service has the responsibility of ordering transactions and bundling them into blocks. Once a block is created, it is broadcasted to all peers that independently validate the block and commit it to their ledger. Note that Hyperledger Fabric supports the use of channels which allow for partitioning the network, thus creating multiple subnetworks. However, for simplicity, we will assume that all peers are in the same channel. Therefore all will receive the broadcasted blocks and will eventually have the same ledger. The case in which there exist multiple channels does not differ except for the fact that only those peers in a channel will receive the newly created block.

To understand where the hash function comes into play, it is worth analysing the structure of a block in Hyperledger Fabric. A block is composed of three main parts presented in Listing 1

Listing 1: Description of the Block structure from Hyperledger

```
type Block struct {
Header      *BlockHeader
Data        *BlockData
Metadata    *BlockMetadata
}
```

The chaining of blocks in the blockchain happens through the hash of the previous block that is inserted into the hash of the current block. Both hashes are stored in the header of the block, as can be seen in Listing 2:

Listing 2: Description of the BlockHeader structure from Hyperledger

```
type BlockHeader struct {
Number          uint64
PreviousHash    []byte
DataHash        []byte
}
```

Listing 3: Hashing of block data from Hyperledger

```
func BlockDataHash(b *cb.BlockData) []byte {
sum := sha256.Sum256(bytes.Join(b.Data, nil))
return sum[:]
}
```

As the name might suggest, the hash of the list of transactions is contained in the `DataHash` field. The `DataHash` field is computed by the ordering service during the creation of a new block. By looking at the code in Listing 3, it is possible to notice how Hyperledger Fabric does not apparently make use of the widespread concept of Merkle tree to compute the hash of the transactions in a given block. Instead, transactions' data is converted into a byte array, and the hash is computed, by default, using SHA-256 checksum algorithm. To be precise, Hyperledger Fabric is designed to support the Merkle tree structure. However, the default width of the Merkle tree's leaves is specified as the maximum dimension of a block and, as a result, the Merkle tree is a simple flat hash over the concatenation of bytes of a block data. Therefore, the Merkle tree results in a single leaf - which is also the root - computed as the hash over the concatenation of all transactions.

THE HASHING PROCESS At this point, there are two options to integrate the ability of modification into this architecture:

1. The first alternative would be to substitute the default hash function used to compute `DataHash` and employ the chameleon-hash function instead. To compute a collision, one could use a sequence of byte equal to the previous one except for the portion representing the transaction to be deleted. While this approach is straightforward and requires very few modifications to the architecture, it poses a major issue. Indeed, to compute a collision, a party will need to know the whole string and will be able to extract meaningful information on other transactions. Moreover, the hash function will be unique, and a single public key has to be used as input. This does not allow for hashing each transaction with a different hashing key as in the design of Derler et al.

2. The second alternative would be to substitute the whole procedure of hashing. Precisely, it could be possible to compute the `DataHash` field as the root of a Merkle tree. Using this approach, a hash will be computed for each transaction and hashes will be combined in pairs and hashed further, until the root of the tree

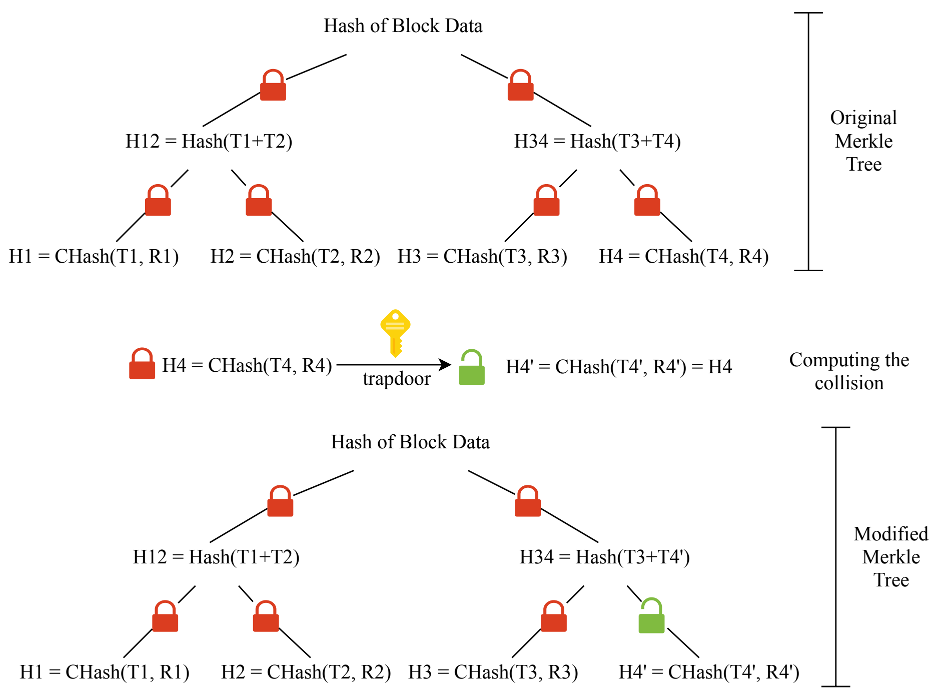


Figure 8: Chameleon-Hash Function into the Merkle Tree

is reached and the final `DataHash` is computed. To compute a collision, one could substitute only the data of the transaction to be deleted and perform the computation for the leaf of the Merkle tree corresponding to that transaction. In such a way, each leaf will be hashed with a different public hashing key, and the additional trapdoor can be managed appropriately. Even though it requires more modifications to the hashing process, this second approach allows for a targeted and proper disclosure of the ephemeral trapdoor. Therefore, we will modify the hashing process as just mentioned.

It is worth noticing that the behaviour of the second approach can be achieved by the first as well by modifying the configuration parameter that specifies the width of the Merkle tree. However, changes to the hashing procedure are still needed to account for different hashing keys. To decide which of the two alternatives for this approach to use should rely on performance reasons, given the fact that they should produce an equivalent result. A visual representation of where the chameleon-hash function should be used to obtain a redactable transaction is presented in Figure 8.

5.2.1.3 *Trapdoor Distribution in Related Work*

Until this point, we identified where the hash function comes into play when creating a new block. This is done by the ordering service which orders and bundles transactions into a block before broadcasting it to the other peers in the network. Recalling the fact that we are introducing chameleon-hash functions with ephemeral trapdoors, we should discuss how to distribute the keys in such a way that the ability to modify a transaction does not belong exclusively to the ordering service. Before diving into the various approaches that we propose, it is worth looking at how the problem is addressed in some of the recent related work.

The work of Ateniese et al. proposes three approaches to manage the trapdoor keys of the chameleon-hash function. Since in their work the trapdoor key is unique, they discuss key management suitable for a single key. Nonetheless, the discussion gives an initial idea of possible approaches. The three key management approaches depend on the type of blockchain used in a particular application, where the types of blockchain are considered as being those presented in Section 2.2.10 due to Buterin [19]. In the case of a private blockchain, one could envision the presence of a central authority. Therefore, the straightforward key management solution would be to give the trapdoor key to a central authority which has the power to compute collisions [23]. In the case of a consortium blockchain, instead, the trapdoor key could be shared among all the parties in the consortium through a Multi-Party Computation (MPC) protocol [23]. To reconstruct the trapdoor key, all the parties - or at least a sufficient number of them - have to engage in the protocol. Last, in a public blockchain, the trapdoor key could be either shared among all the full miners or shared among a carefully chosen set of full miners [23].

A different proposal is the one presented in [24] by Derler et al. In their work, they envision the role of the attribute authority, which is responsible for issuing the public key used to compute the hash function to the various address owners in the network. Further, the attribute authority issues the secret keys containing a list of attributes to all the other users. When an address owner wants to submit a modifiable transaction, it computes the chameleon-hash of the transaction, and it encrypts the ephemeral trapdoor using an access policy. If - and when - a transaction needs to be modified, only the parties possessing a secret key that satisfy the access policy can decrypt the ephemeral trapdoor and to compute a collision.

5.2.1.4 *Trapdoor Distribution Schemes*

We will now discuss some of the trapdoor distribution approaches that we identified. To help with the discussion, we introduce the following notational convention:

- u represents a user in the network;
- p represents a peer in the network;
- $\mathbb{U}$ represents the set of users in the network;

- CA is the certificate authority that will acquire the role of attribute authority
- pk_u represents the public key of the chameleon-hash function associated with the user u;
- sk_p represents the secret key of the chameleon-hash function associated with the peer p;
- ek represents the ephemeral trapdoor;
- $ch(t, k)$ is a chameleon-hash function with input the transaction t and a key k
- $enc(m, k)$ is the an encryption procedure with inputs a message m and the encryption key k;
- $dec(c, k)$ is the decryption procedure with inputs a ciphertext c and the decryption key k;
- $enc(m, \mathbb{A})$ is the an encryption procedure with inputs a message m and an access policy $\mathbb{A}$;
- $dec(c, [a])$ is the decryption procedure with inputs a ciphertext c and a list of attributes $[a]$;

In our work based on the architecture of Hyperledger Fabric, the ordering service nodes inevitably play a central role being the ones that compute the block's hash value using the ordered list of transactions as input. However, we do not want to give the ordering service the power of being only one that knows all ephemeral trapdoors. As a consequence, we identified the following - non-exhaustive - list of key management solutions:

1. The first approach is similar to the one proposed by Derler et al. In the beginning, the attribute authority CA sends the public hashing keys pk_u to all users $u \in \mathbb{U}$ and the secret keys sk_p to all peers in the network. Note that sk_p contains a list of attributes. Upon submitting a transaction, a user u sends its public hashing key pk_u to the ordering service node. The ordering service node uses the pk_u to compute the chameleon-hash of a transaction t as $ch(t, pk_u)$. Further, it computes the ephemeral trapdoor ek and it encrypts the trapdoor with an access policy $\mathbb{A}$ as $enc(ek, \mathbb{A})$. Upon receiving a request to modify a transaction, a peer will try to decrypt the ephemeral trapdoor using the attributes $[a]$ attached with its secret key sk_p using the procedure $dec(c, [a])$. If the attribute list satisfies the access policy $\mathbb{A}$, the peer can decrypt the trapdoor and to compute a collision. Once a collision has been found, it is possible to insert the new message (or to delete the old one by replacing it with a placeholder) into the blockchain without breaking the chain. A

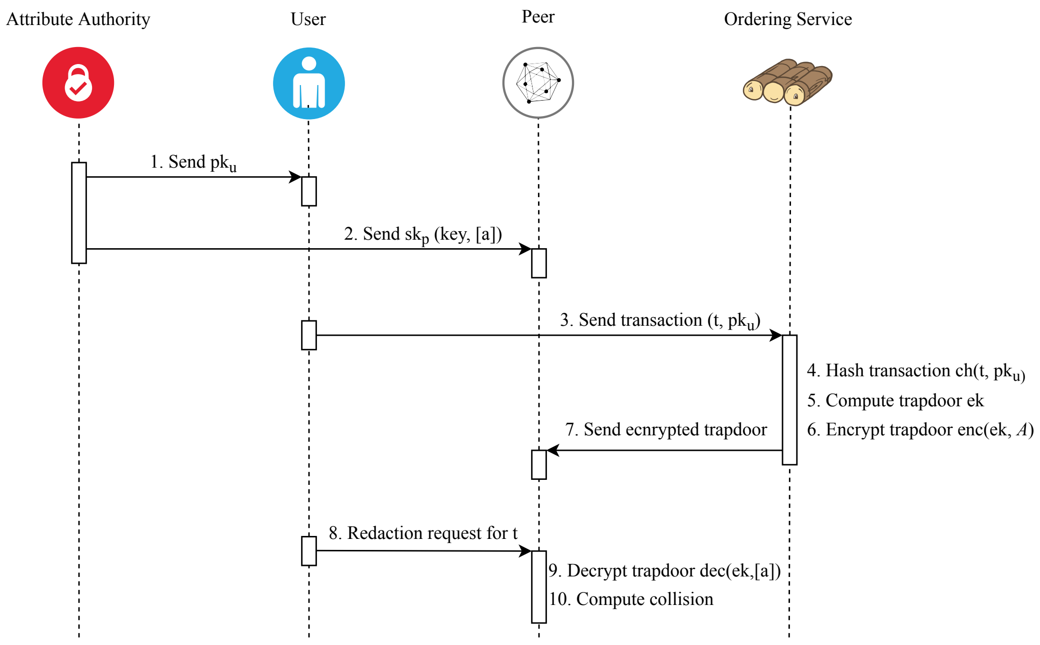

Attribute Authority
User
Peer
Ordering Service
1. Send pk_u
2. Send sk_p (key, [a])
3. Send transaction (t, pk_u)
4. Hash transaction ch(t, pk_u)
5. Compute trapdoor ek
6. Encrypt trapdoor enc(ek, A)
7. Send ecnrypted trapdoor
8. Redaction request for t
9. Decrypt trapdoor dec(ek,[a])
10. Compute collision

Figure 9: First Key Management approach

visual representation of this approach can be found in Figure 9.

2. The second approach, similar to the first one except for the distribution of ephemeral trapdoors, is represented in Figure 10. Instead of encrypting the ephemeral trapdoor ek with respect to an access policy $\mathbb{A}$, the ordering service can send ek to the user u. We note that there are different methods to achieve such a result, and we do not discuss them now. When the user u wants to submit a request, it discloses the ephemeral trapdoor to the peer p representing the data controller. Upon receiving the request and ek, the peer p can compute a collision and redact the transaction as requested by u. Note that using this approach, the secret key of the peers does not necessarily have to contain the list of attributed $[a]$.

3. The third approach combines the presence of the ephemeral trapdoor proposed in [24] with the distribution of the keys as envisioned in [23] in the case of a consortium blockchain. Similarly to the previous one, it differs from the first because of the way the ephemeral trapdoor ek is distributed. Indeed, the ordering service can divide the ephemeral trapdoor ek into shares and send the shares to the peers in the network. To perform a redaction after a data subject request, peers rebuild the trapdoor by engaging in a multi-party computation protocol. Once the secret has been reconstructed, it can be used by the peer identified as the data controller to compute a collision and perform the

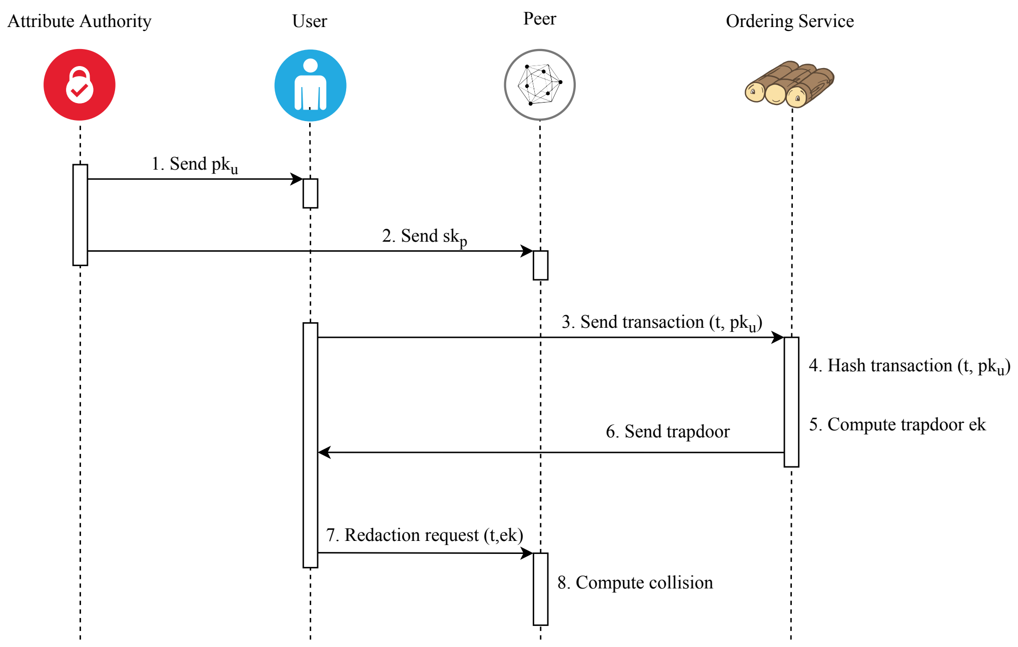


Figure 10: Second Key Management approach

redaction requested by the user. The approach is represented in Figure 11.

4. The fourth approach combines the second and the third. Precisely, the distribution of the ephemeral trapdoor is done similarly to what presented in [23], but a share of the trapdoor is sent to the user who proposed the transaction. Therefore, the ordering service can divide the ephemeral trapdoor ek into shares and send the shares to the peer representing the data controllers(s) and the data processor(s) as well as to the user u. However, instead of providing the same share to all sharers, different parties receive shares of different weight. Through a weighted secret sharing scheme, it is possible to assign more reconstruction power to a party compared to others. We propose assigning the strongest share to the user u so to provide u with the highest level of control on its data. The other parties will receive weaker shares compared to the user. If the other shares alone are enough to reconstruct the trapdoor, the recovery of the secret in case the user u does not want to engage in the protocol or lost its share is still possible. Alternatively, we can require the user's share to be present, resulting in higher security. We stress that the decision depends on the application, and we do not prefer one or the other mechanism. Moreover, to avoid the need to trust the ordering service as a dealer, it is possible to use verifiable weighted secret sharing schemes in which a party can check the correctness of the received share using the public

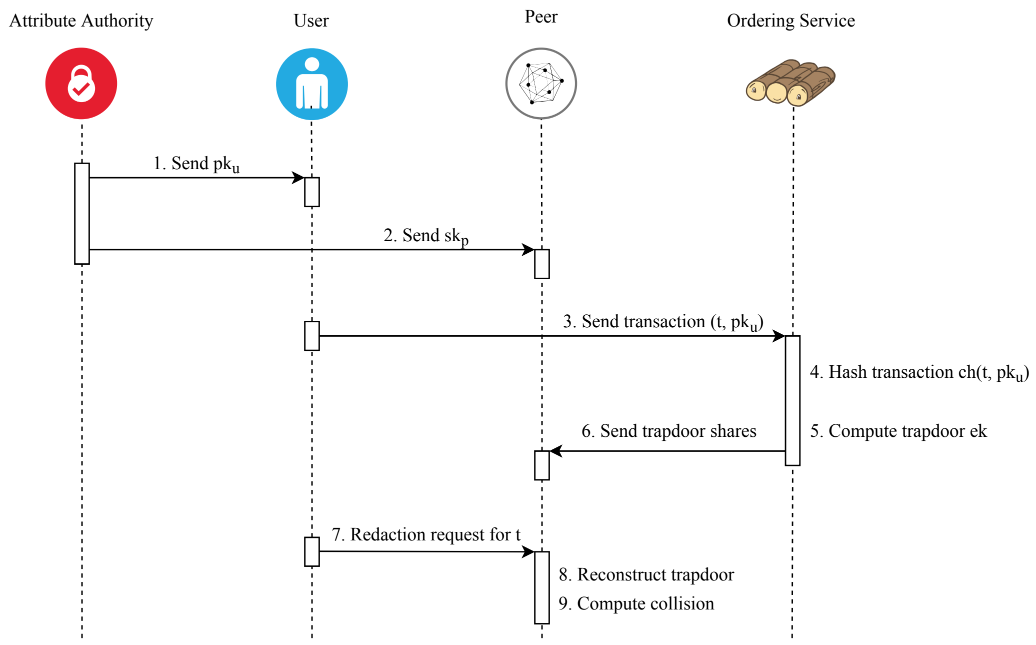


Figure 11: Third Key Management approach

commitment attached to the share. A visual representation of the fourth approach is given in Figure 12.

5.2.2 *Evidence of Modification into the Block of Transactions*

The second building block of our proposal consists of providing evidence of the fact that a modification has happened. Note that this is different from the proof that each peer will publish on the blockchain during the process. Indeed, the evidence will be bundled into the block that contains the modified transaction and not in a newly created block to certify that the process is finished. This evidence of modification highlights the fact that the transaction is not the original one even though its hash is the same as the original one due to the collision found through the use of the chameleon-hash function. The insertion of this evidence into the block is the last step of the redaction process.

A naive approach to provide such evidence is the use of transaction validation codes, a built-in code in Hyperledger Fabric. Due to the particular transaction flow of Fabric, it could happen that a transaction submitted by a user and inserted into a block by the ordering service is not valid at the moment of its commitment to the ledger. This conflict may arise for various reasons. For instance, if a second transaction that modifies the same values has been inserted into the blockchain after the first one was proposed, but before it was committed. If any of the conflicting situations happens, the transaction is inserted into the block, but it is marked as invalid with one of the

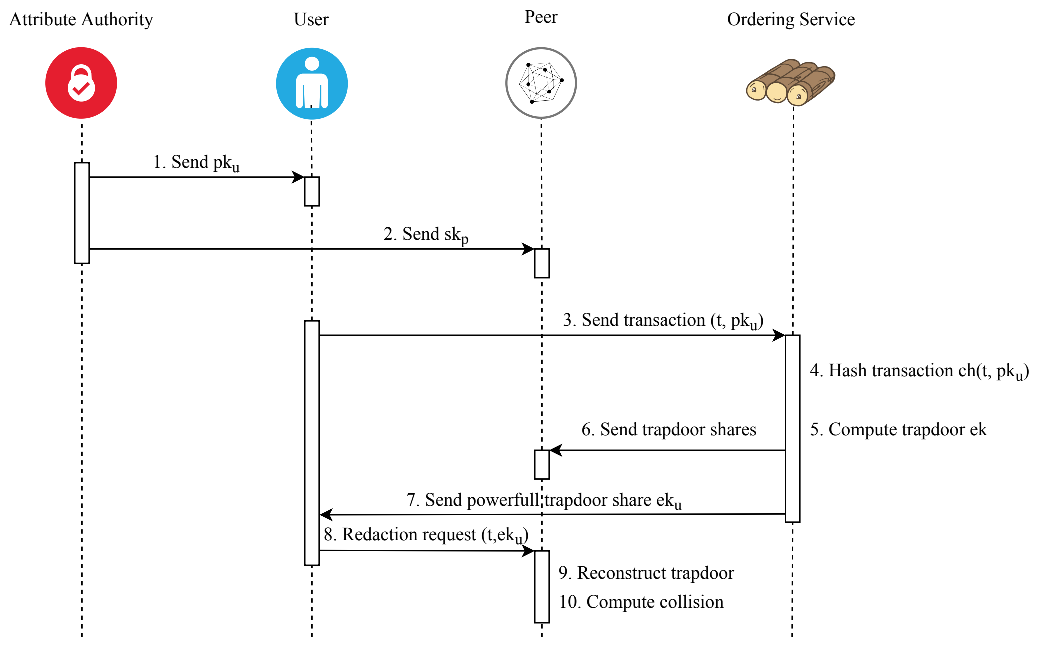


Figure 12: Fourth Key Management approach

available transaction validation codes. By adding a unique code to the list of available ones - as represented in dark red in Listing 4 - each peer should be able to mark the modified transaction with the proper validation code. Whenever a party wants to check whether the transaction has been changed, it is sufficient that it verifies the transaction validation code by making a request to the peers in the network. This code is inserted into the block metadata, the third field of the structure presented in Listing 1. Since block metadata is not considered during the computation of the block hash, a modification of this field does not create a conflict in the block chaining. As one would expect, the evidence in the block is publicly visible.

Listing 4: Transaction validation codes from Hyperledger

```
type TxValidationCode int32

const (
TxValidationCode_VALID                          TxValidationCode = 0
TxValidationCode_NIL_ENVELOPE                   TxValidationCode = 1
TxValidationCode_BAD_PAYLOAD                    TxValidationCode = 2
TxValidationCode_BAD_COMMON_HEADER              TxValidationCode = 3
.
.
.
TxValidationCode_REDACTED_TRANSACTION           TXValidationCode = 253
TxValidationCode_NOT_VALIDATED                  TxValidationCode = 254
TxValidationCode_INVALID_OTHER_REASON           TxValidationCode = 255
)
```

5.2.3 *Proof of Redaction*

The proof of redaction represents the last piece of our redactable blockchain design. As we observed in Chapter 3, the designs that leverage chameleon-hash functions do not present any evidence of the fact that a transaction has been modified. The absence of the evidence comes from the *indistinguishability* property of chameleon-hash functions: it is not possible to tell whether the hash value is computed by the hashing process of the original data or if it comes from the adaptation of an already existing hash. Indistinguishability is a very appealing property in many cases, and the fact that two hashes are identical and indistinguishable helps in avoiding the need to *break the chain* to redact data. However, we have also observed that this absence of evidence could weaken the tamper-evidence of a blockchain.

Tamper-evidence and tamper-proofs are the building blocks of blockchain's immutability. We saw that the tamper-proof property might be hacked through the use of chameleon-hash functions. Similarly, the tamper-evident property could be hacked by the use of chameleon-hashes. Therefore, we need to integrate an additional mechanism to retain the tamper-evident property. We have already introduced the evidence of modification into the transaction block. However, it is as simple as an added value to identify which transaction has been redacted. A question that pops up is what happens if the code is used for a valid transaction. The blockchain will show a redacted transaction; however, the transaction will not be a redacted one, but the original one where the transaction validation code has been misused. How can one discern between an original transaction with a misused validation code from a redacted transaction?

5.2.3.1 *A naïve Proof-of-Redaction*

To answer this question, we introduced the Proof-of-Redaction (PoR). The Proof-of-Redaction is an additional transaction published to the blockchain to show that the network agreed on the modification or deletion of an already published transaction. It is proposed by the party computing the collision, i. e., one of the peers of the data controller to whom the data subject submitted its request. Once the collision has been calculated, the peer proposes a transaction containing a new timestamp for the transaction, the new data used to modify the existing transaction, and the random number that produces the collision. The endorsement policy for such type of transaction should be designed so that each organization can check the validity of the proposed data and randomness through the use of the public chameleon-hash validation function.

Once the collision has been verified, the endorsing peers of all organizations in the network send a proposal response to the data controller's peer which inspects the response and, if all endorsements are correct, submits the transaction to the ordering service. The transaction can be submitted using the standard hashing mechanism or using the chameleon-hash function. If the goal is to avoid a possible redaction of the proof, the hash should be computed using a standard hash function. However, if the proof contains personal data that could be modified or deleted, the chameleon-hash should be preferred. Once the Proof-of-Redaction has been published to the ledger, each peer can modify data and use the proposed randomness to compute the collision. Moreover, the transaction validation code can be modified to show the presence of a redacted transaction.

5.2.3.2 *Committing the Randomness*

At first sight, the naive Proof-of-Redaction we have just proposed seems to work just fine. However, after a more in-depth look, it is possible to notice that it presents some weaknesses.

As a redaction proposal, the party computing the collision presents the new data, a new timestamp, and the random number that produces the collision. At this point, each peer should wait for the transaction to be approved before changing the original transaction. However, all parties are already aware of all the components needed to generate a valid collision. Therefore, a malicious party could modify the original transaction before the other parties validate the redaction proposal. To avoid the situation in which a party changes a transaction before others approve it and the risk of ending up in an inconsistent state, we decided to substitute the random number with a commitment to it.

As a result, the redaction procedure becomes a two-step process depicted in Figure 13. During the first step, the party computing a collision publishes the new data, a new timestamp, and a commitment to the random value that allows producing a collision. This set of data is bundled into a transaction proposal which, as before, needs to be signed by the other parties in the network. However, at this point, no party except the one computing the collision is aware of all the components needed to compute it. If the other peers sign the transaction proposal and all signatures are correctly collected by the party computing the collision, the transaction is published to the ledger as a mean to demonstrate that the network agrees on the proposed modification. When the proposal is published, the party computing the collision can post the random number to the ledger. Upon receiving the random number, all other parties can validate the commitment by verifying that the published randomness satisfies the commitment and confirm the fact that the random number, together with the new data and timestamp, produces the desired collision.

Thanks to the presence of the Proof-of-Redaction, it is possible to distinguish a transaction containing the original data with a misused transaction validation code from one that has been modified follow-

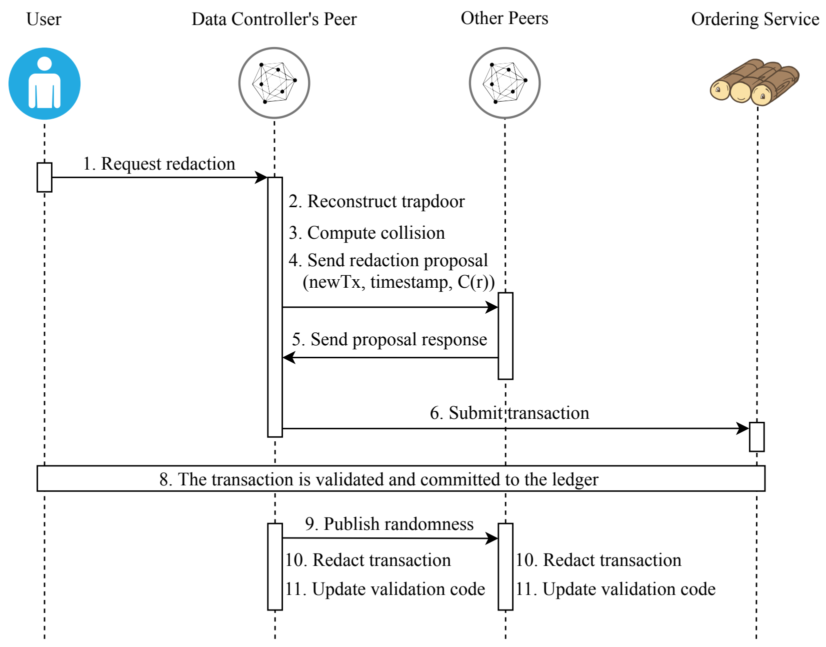


Figure 13: Committing the Randomness for Proof-of-Redaction

ing the procedure. A correctly redacted transaction will have a new timestamp bigger than the other timestamps in the block. The same timestamp and the redacted data can be found on the ledger together with the randomness that allows creating the block. Therefore, if a transaction presents the `TxValidationCode_REDACTED_TRANSACTION` and there exist the connected Proof-of-Redaction on the ledger, the transaction has been correctly redacted. Instead, if the transaction presents only the redaction code, but there is no proof connected with that transaction on the ledger, it is possible to conclude that the code was misused and the transaction the original one. It is worth noticing that a transaction with a misused transaction validation code is considered an invalid transaction. Therefore, it did not produce any change to the ledger when it was inserted into the blockchain.

5.2.4 *The case of Dependent Transactions*

The discussion we had until this point was focused on understanding how it is possible to achieve modifiability of a single transaction through the use of chameleon-hash functions and how the data controller and data processor's peers provide proof of the modification by publishing a transaction to the blockchain. A question that naively pops out is what happens when multiple transactions are linked together, i. e., a transaction on the blockchain depends on one that has been modified or deleted. The fact that transactions dependent on a redacted one should be modified depends on the particular application for which the blockchain has been designed. Let us create an

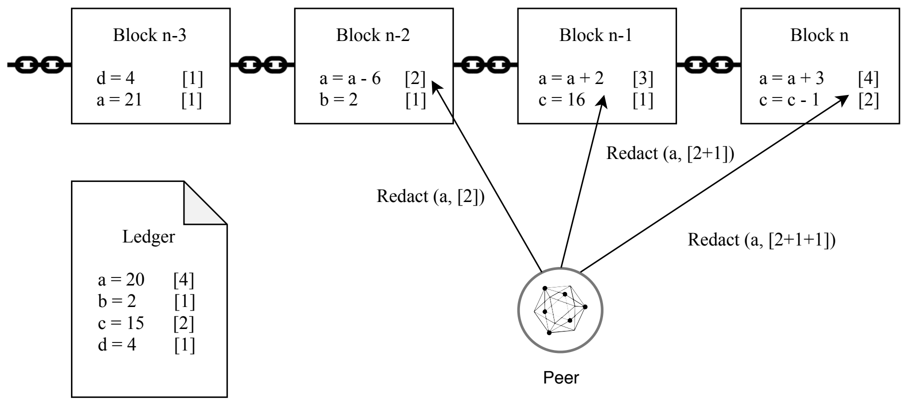


Figure 14: The case of dependent transactions

artificial example of a land registry. The blockchain stores transactions that occur when a user buys or sells a piece of land. We do not store money transactions, but we just store the fact that the owner of a piece of land has changed. If a data subject is granted the right to have its transaction removed, the deletion of that transaction does not interfere with other transactions in the blockchain unless the data subject has been the last owner of the piece of land. However, if the data subject is still the owner of the property, any deletion request is likely to be denied.

In the case in which transactions are dependent, it should be possible to scroll the whole list of blocks to find transactions whose result could be changed as a consequence of redactions. Even though it does not seem to be a native built-in method in Hyperledger Fabric to achieve such a result, we can leverage how the architecture stores assets and process transactions. Assets are stored as key-value pairs and transaction proposals submitted by a user contains the key value of the asset they are going to modify and its version. Version numbers of an asset are used to check that, while a transaction was being endorsed, ordered, and committed, another transaction did not modify the same asset, thus invalidating the initial state of the former transaction. When a transaction is redacted, we could save the value of the representing the asset that has been modified and its version number. To check for a subsequent transaction that depends on the redacted one, we can search for transactions that contain the same key value but a version number higher than the one included in the redacted transaction. By doing so, we should be able to identify all transactions that have been stored on the ledger and depend on the redacted one given the fact that their execution modified the version of the same asset. A representation of a simplified process is given in Figure 14.

To be precise, there is a built-in method if Hyperledger Fabric is configured to work with a specific state database, namely Level-DB. Level-DB provides a technique called `GetHistoryForKey(key)` which allows retrieving all transactions that modified the specified key. In such a situation, it is possible to use the built-in function. If a different state database is used, the procedure we identified above achieves a similar result.

5.3 IMPLEMENTATION

In this section, we present our implementation of the design. We begin with the implementation of the chameleon-hash with ephemeral trapdoor.

5.3.1 *Chameleon-Hash with Ephemeral Trapdoor Implementation*

We present our implementation of a chameleon-hash function with ephemeral trapdoor based on a chameleon-hash function presented in [45]. We provide the first implementation of such a hash function in Go using a black-box approach as described in [45]. Such a black-box approach allows us to implement the presence of the second - ephemeral - trapdoor by combining two instances of chameleon-hash functions instead of implementing the presence of the trapdoor directly. We decide to implement the ephemeral trapdoor via a black-box combination of two chameleon-hash implementations due to the fact that other approaches require the computation of a huge random prime number. We empirically tested that the computation of a random prime $e > n^3$, where n is an RSA 2048-bit modulo and e is the public exponent, introduces a heavy overhead. Instead, the combination of two chameleon-hash functions requires to compute two random primes $e_1 > n_2$ and $e_2 > n_2$ separately. This approach turned out being much faster. Therefore, we decided to opt for the black-box construction. Notably, both the direct approach and the black-box approach present the same security properties that we will discuss in Chapter 6.

We now show our implementation of the chameleon-hash function, and we discuss which parties in the network execute which steps. Since the black box-construction is a combination of two chameleon-hash instances, we present one chameleon-hash instance. The presence of the trapdoor is implemented by using an additional instance of the chameleon-hash function. According to the construction provided in [45], the first step is the generation of e, the RSA public exponent. Connected with the generation of the public exponent is the generation of the RSA modulo n and the private exponent d. As in any standard RSA instance, n is generated as a product of two big random prime numbers, p and q, whereas d is computed as the inverse of e modulo $\phi(n)$. These two algorithms are executed when a user joins the network. The public exponent e is connected with the identity of the user and plays the role of the public hashing key. Instead, the private exponent d works as one of the two trapdoors that allow a peer to compute a collision. Since the data controller's peers are in charge of computing the collisions, the key can be directly made available to them, or it can be sent together with the public hashing key to the user. Note that disclosing the private exponent d to a peer since the bootstrapping of the network does not pose security problems as d alone does not allow a party to compute a collision.

5.3.1.1 *Setup*

The function `CParGen` - presented in Listing 5 - generates the RSA public exponent e as a random prime number with a bit-length λ. In our implementation, λ is set to 2048 bits. This function is executed when a user joins the network and assigns a hashing key to it. This step is executed by the Certificate Authority (CA) that is already in charge of generating an identity for users upon registration to the network.

Listing 5: Generation of RSA public exponent

```
func CParGen (1^λ) {
e := randomPrime(λ)
return e
}
```

The function `CKeyGen` - presented in Listing 6 - generates the RSA modulo n and the private exponent d. Similarly to the previous function, `CKeyGen` is computed by the CA when a user registers to the network. The presence of the second ephemeral trapdoor is provided through the execution of this function by the ordering service during the hashing process. Once the second trapdoor has been generated, it can be divided in shares and sent to the relevant parties by using a verifiable and weighted secret sharing scheme. The verifiability of the sharing scheme allows each party to prove that the share they receive is correct and the fact that the scheme produces shares of different weights allows us to provide a powerful share to the data subject while providing enough less powerful shares to the other parties.

Listing 6: Generation of RSA modulo and private exponent

```
func CKeyGen (λ, e) {
repeat {
p := randomPrime(λ/2)
q := randomPrime(λ/2)
n := p*q
phi(n) := (p-1) * (q-1)
} until e > n AND isCoprime(phi(n), e)
d := inverseModulo(e, phi(n))
return n,d
}
```

5.3.1.2 *Hashing*

Listing 7 presents the function `CHash` which is executed by the ordering service to hash a transaction using a user's public key, i. e., the RSA public exponent. The function `randomNumber()` is designed so to provide a random number of at most λ bits. The function $H^*_n()$ is a random oracle instantiated as a SHAKE256 hash function with 256 bytes of output, i. e., 2048 bits. The function computes a random number r and uses the public exponent e, the randomness r, and the hash function $H^*_n()$ to hash a message. To insert the second ephemeral trapdoor, it is sufficient to execute the same function a second time using the second RSA modulo generated through the execution of `CKeyGen`.

Listing 7: Hashing Process

```
func CHash (λ,e,n,tx) {
r := randomNumber(λ)
h1 := H*_n (tx)
hash := h1 ∗ r^e mod n
return hash,r
}
```

5.3.1.3 *Collision*

The procedure used to find a collision is presented in Listing 8. The private exponent d allows to revert the initial exponentiation as in any standard RSA algorithm and produces as output a new random value $newR$ that, together with the new transaction data $newTx$ allows to find a collision for the original hash value.

Listing 8: Collision Computation

```
func CAdapt (oldHash,n,tx,newTX,d) {
if newTX == tx return r
h1 := H*_n (newTx)
newR := (oldHash ∗ inverseModulo(h1, n))^d mod n
return newR
}
```

5.4 SUMMARY

Before jumping to the evaluation of the proposed design, we should briefly recall how we introduced the ability of modification, who possesses the keys to activate the modification process, and what evidence of modification we added to the architecture. By doing so, we provide the answer to *SQ4* and *SQ5* of research question *RQ1* in Subsection 5.4.1 and Subsection 5.4.2 respectively.

5.4.1 *Prototype Design*

In this section, we provide the answer to *SQ4* of *RQ1* that asks which is a proper design for a prototype. We described the various building blocks of the design and how they are integrated with into an existing blockchain implementation in the previous section. Our design is based on the work of Camenisch et al. which proposed a new primitive called chameleon-hash function with ephemeral trapdoor. By modifying the standard hash function used to compute the hash of the transactions in a block, we provide the ability to modify or delete a transaction without breaking the connection between blocks. However, we felt that a modification or deletion should be publicly visible so that the blockchain will retain the tamper-evident property. Therefore, we introduce a transaction validation code that marks a transaction as redacted as soon as a modification or deletion happens. Last, we envision the presence of a proof published by the data controllers and data processors - if any - into the blockchain.

5.4.2 *Modification Rights*

In this section, we provide a possible answer to *SQ5* of *RQ1* that asks who should have the right to propose and approve redactions. First of all, we note that the redaction request comes from the data subject and needs to be evaluated and accepted by the DPO - or the DPA in case of a dispute. In our work, we focus on the data modification rights on the ledger when a redaction has been previously approved. As we mentioned earlier, it should be possible to automate part of the decision process through smart contracts. However, the current level of uncertainty around the process of granting or denying a data subject's request hinders our ability to design rules for automated enforcement.

We identified four different possibilities to handle the redaction process and avoid to centralise the power on the ordering service. First, secret keys distributed to the peers of the data controller could contain a set of attributes. Upon transaction validation, the ordering service encrypts the ephemeral trapdoor subject to an access policy. When a redaction is requested, only the peers satisfying the access policy will be allowed to decrypt the trapdoor and compute a collision. Second, the ordering service could send the ephemeral trapdoor directly to the data subject. Upon redaction request, the data subject discloses the ephemeral trapdoor to the data controller's peers which compute the collision and perform the redaction. Third, the ephemeral trapdoor could be distributed using a secure secret sharing scheme. Upon redaction request, the trapdoor could be rebuilt through a collaboration between the parties. When the secret is reconstructed, it can be used to compute a collision and perform the redaction. Last, the ephemeral trapdoor could be divided into weighted shares using a weighted and verifiable secret sharing. The strongest share should be sent to the user while the other weaker shares are sent to the data controller and data processor peers. The secret trapdoor can be reconstructed using the strongest share and any of the other share or by combining all other shares. Using this approach, it could be possible to reconstruct a secret even when the user does not want to engage in the protocol or when the user's key is lost.

While the second approach can be seen as the one that reflects the directions of the GDPR by giving more control to the data subject, we acknowledge that the process of managing keys could be a hard task for many people. On the other side, the first approach as-is seems to give high power to the ordering service that decides the access policy to use during the encryption of the ephemeral trapdoor. Lower reliance on the ordering service could be achieved by asking the ordering service to prove the fact that the access policy allows the authorised parties to decrypt the trapdoor. The third approach seems to be the one that requires a fair and distributed effort to many parties. Again, the ordering service should prove the fact that the secret can be reconstructed from the various shares. The fourth and last approach combines the involvement of the data subject with a distributed effort in the key reconstruction. We believe this approach is the one that achieves the goal of giving data subject more control over its data with the possibility to allow the other parties to reconstruct the secret and perform redactions in case the data subject is not willing to engage in the protocol or when the data subject's key is lost.

6 EVALUATION AND DISCUSSION

This chapter describes the process used to evaluate the design of the architecture we provided in Chapter 5. To choose the appropriate approach for our evaluation, we will follow the framework of Venable et al. In particular, we begin by choosing the evaluation approach based on the four-step process presented in the authors' work [48].

The first step revolves around the formalisation of the evaluation's goal. The goal of our evaluation is to demonstrate that the design we are proposing achieves the technical requirements we identified without a disproportionate impact on an existing blockchain. To align our goals with the categories identified in [48], we can formalise the goal as *establishing the efficacy* of our architecture.

The second step of the process requires to decide a strategy for the evaluation. In their work, Venable et al. identify four different strategies. Due to the technical nature of our work, it is sound to reduce our choice between a *Technical Risk & Efficacy* strategy and a *Purely Technical* one. However, the problem we identified is not purely technical as it involves a legal component. Therefore, we decided to follow a *Technical Risk & Efficacy* strategy.

The third step regards the identification of the properties to evaluate. According to Venable et al., the selection of properties is unique to the artefact under evaluation. Therefore, we formalised a set of qualities we wanted to assess. We used an analytical approach [2], and we performed both static analysis and dynamic analysis.

The fourth and last step concerns the design of the evaluation episodes. We designed our evaluation as shown in Table 2. The evaluation of our work is structured in three parts, of which the first two are static analysis and the last part is the dynamic analysis. According to Hevner et al., static analysis examines "*the structure of an artefact for static qualities*" [2], whereas dynamic analysis studies the "*artefact in use for dynamic qualities*" [2]. We will define both the static qualities and the dynamic quality for each evaluation part.

The first part of our evaluation is a security analysis of the redactable blockchain design. Since our work is modifying an existing and established blockchain architecture, there is the risk of unintentionally weakening the system. To check whether our work produces such an undesirable outcome, we inspire the first part of our evaluation on the work of Garay et al. [49] and Kiayias et al. [50, 51]. Their work demonstrate three fundamental requirements of a *robust transactional ledger*, namely *common prefix*, *chain quality*, and *chain growth*, ensure the properties of *Persistence* and *Liveness*. While the three requirements have

Evaluation Method		Artefact	Goal
Static Analysis	Proof	Architecture Design	Demonstrate that an existing architecture is not weakened
Static Analysis	Expert Interviews	Architecture Design	Validate design to ensure that key properties are satisfied and requirements are met
Dynamic Analysis	Performance Test	Chameleon-Hash Function	Test the feasibility of using chameleon-hash in existing architectures

Table 2: Evaluation Methods used in This Study

been initially used to assess the robustness of the consensus protocol, they have been lately employed to validate the design of modifiable blockchain in [27, 52].

The second part of our evaluation is a qualitative evaluation performed through semi-structured interviews with domain experts. Experts were chosen based on their knowledge of the blockchain and the cryptographic primitives. Both the first and the second part of the evaluation constitute our static analysis according to its definition given in [2]. We will carefully define the static qualities - or properties - of the system that we want to evaluate in the dedicated sections.

Last, as the third part of our evaluation, we discuss the performances of the implementation of one of the building blocks, namely chameleon-hash function with ephemeral trapdoor. This third phase is the dynamic analysis that tests the performance of the fundamental building block of our architecture.

6.1 ASSUMPTIONS AND SECURITY OF CHAMELEON-HASHES

This section presents the assumptions and the threat model we will use as a reference attacker model for our evaluation.

6.1.1 *Permissioned Network*

We decided to discuss a possible integration of our architecture in Hyperledger Fabric. The choice of Fabric is mainly due to its permissioned setting and the higher possibility to comply with various GDPR requirements. Fabric offers the ability to create channels, i. e., subnetworks of peers that will share the same ledger and will execute the same chaincode. The sharing of information could, therefore, be designed so that only the relevant parties will get access to the data. Moreover, Fabric recently introduced an additional feature called private data: data deemed as private can be shared only with the relevant stakeholders with or without the presence of a channel. Furthermore, private data can be obfuscated for the ordering service, that may be controlled by an organisation that should not be able to access the content of a transaction.

As a result of our choice of Hyperledger Fabric, we evaluate our design in a permissioned environment. To align the meaning of permissioned with the taxonomy we presented in Chapter 2, we can refer to Fabric as a consortium where a pre-identified set of organisations controls the ordering service and the endorsement policies. Write permissions are restricted to the parties that are part of the network or part of a channel in the more restrictive situation. Similarly, read operations are limited to those parties that are part of the network or channel. However, even though parties do know each other, they do not necessarily need to trust each other.

6.1.2 *Security Properties of Chameleon Hash Functions*

In this section, we present the properties of chameleon-hash with ephemeral trapdoor (CHET) based on [45]. The proofs of the security properties can be found in [45]. The assumptions on which the security of the hash function are based are the same of the ones stated in their paper, namely the RSA assumption and the one-more RSA-assumption. The formalisation of both assumptions can be found in the original article [45], which shows that the construction is secure under those assumptions in the random oracle model [45].

Definition 6.1 (Indistinguishability) *[45] A chameleon-hash function with ephemeral trapdoor CHET is **indistinguishable** if for any efficient adversary $\mathcal{A}$ there exist a negligible function ν such that*

$$|\Pr[\mathsf{Indistinguishability}_{\mathcal{A}}^{\mathsf{CHET}}(\lambda) = 1] - \frac{1}{2}| \leqslant \nu(\lambda)$$

Informally, *indistinguishability* requires that a randomness r does not reveal whether it was obtained through `CHash` or through `Adapt`, that is, it is not possible to understand whether the randomness is produced during the hashing process or the computation of a collision. As a consequence, an outsider cannot decide if the transaction is the original one or if the modified one.

Definition 6.2 (Public Collision Resistance) *[45] A chameleon-hash function with ephemeral trapdoor CHET is **publicly collision resistant** if for any efficient adversary $\mathcal{A}$ there exist a negligible function ν such that*

$$\Pr[\mathsf{PublicColRes}_{\mathcal{A}}^{\mathsf{CHET}}(1^{\lambda}) = 1] \leqslant \nu(\lambda)$$

Informally, *public collision resistance* requires that even if an adversary has access to an `Adapt` oracle it cannot find a new collision, i. e., different from the one generated by the oracle, by itself.

Definition 6.3 (Private Collision Resistance) *[45] A chameleon-hash function with ephemeral trapdoor CHET is* ***privately collision resistant*** *if for any efficient adversary $\mathcal{A}$ there exist a negligible function ν such that*

$$\Pr[\mathsf{PrivateColRes}_{\mathcal{A}}^{\mathsf{CHET}}(1^{\lambda}) = 1] \leqslant \nu(\lambda)$$

Informally, *private collision resistance* requires even the holder of the secret key cannot compute a collision by itself if the second ephemeral trapdoor is unknown even in the presence of an honest hashing oracle which does not return the trapdoor.

Definition 6.4 (Uniqueness) *[45] A chameleon-hash function with ephemeral trapdoor CHET is* ***unique*** *if for any efficient adversary $\mathcal{A}$ there exist a negligible function ν such that*

$$\Pr[\mathsf{Uniqueness}_{\mathcal{A}}^{\mathsf{CHET}}(1^{\lambda}) = 1] \leqslant \nu(\lambda)$$

Informally, *uniqueness* requires that it is hard two find two distinct randomness values r_1 and r_2, $r_1 \neq r_2$ for the same message m and hash value h.

As an additional property, we require a CHET to be correct. Precisely, we require that for every security parameter $\lambda \in \mathbb{N}$, for all public parameters generate from the `Setup`, for every public exponent e, modulo N, message m, for every hash h, randomness r, and trapdoor etd, we have that the check `CHashCheck()= true` [45]. Similarly, we require that for each message m' and randomness r' outputted by `Adapt`, `CHashCheck()= true` [45].

According to Camenisch et al., a chameleon-hash with ephemeral trapdoor is said to be *secure* if is *correct*, *indistinguishable*, *publicly collision resistant*, and *privately collision resistant*.

6.1.3 *Threat Model*

Although the trust model is usually defined when designing a cryptosystem, we felt the need to formalise our assumption regarding the trustworthiness or the different parties in the network. We have already discussed that, given the permissioned nature of Fabric, parties do know each other but do not necessarily need to trust each other. We can formalise this assumption with the honest-but-curious - or semi-trusted/semi-honest - threat model where the different parties follow the protocol but are eager to learn as much as possible regarding the others' secrets. Additionally, we require that the peers in the network do not collude to reconstruct the data subject secret. By doing so, we maintain the ability to rebuild the second trapdoor even when the share of the data subject is lost. However, we observe that it is possible to relax this requirement and allow collusions among network peers if the threshold on the commitment scheme is designed to require the data subject's share.

6.2 SECURITY ANALYSIS

In this section, we define the properties of *Persistence* and *Liveness*. We also define the requirements that ensure the achievement of these properties. The three requirements, namely *common prefix*, *chain quality*, and *chain growth* are the three qualities on which we base our static analysis. Our goal is to show that our design does not preclude the ability to ensure the properties of a *robust transactional ledger*. Therefore, we prove that it does not violate any of the requirements mentioned above.

6.2.1 *Security Requirements and Properties*

We begin by introducing the definition of *Persistence* and *Liveness* due to Garay et al. and Kiayias et al.

Definition 6.5 (Persistence) *The property of persistence (with parameter* $k \in \mathbb{N}$*) states that when a node in the network proclaims a transaction* t *as stable, all other nodes in the network - if queried - will report the transaction* t *at the same position in the ledger and will agree on the entire prefix of the ledger [49, 50], i.e., the ledger up to the transaction* t*. A transaction is said to be stable if at least* k *blocks have been appended to the ledger after the block that contains the transaction* t*.*

Definition 6.6 (Liveness) *The property of liveness (with parameter* $u \in \mathbb{N}$*, referred to as transaction confirmation time) states that if all honest nodes in a network are willing to insert a transaction* t *in the ledger, after a time* u *all nodes - if queried and honestly responding - will report the transaction as stable [50].*

We now introduce the definition of the three requirements, namely *common prefix*, *chain quality*, and *chain growth*. While they have been formalised in [49, 50], we provide a definition closer to the ones reported in [27, 52] as these include the original ones and are formalised in a simpler fashion with a focus on the design of redactable blockchain.

Definition 6.7 (Chain Growth) *The property of* ***chain growth****, with parameters* $\tau \in \mathbb{N}$ *and* $s \in \mathbb{N}$*, states that if two chains* C_1 *and* C_2 *possessed by two honest parties at two time slots* slt_1 *and* slt_2*, with* $slt_2 - slt_1 > s$*, then it holds that* $len(C_2) - len(C_1) \geqslant \tau \cdot s$*, for* $s \in \mathbb{N}$ *and* $0 < \tau \leqslant 1$*, where* τ *is the speed coefficient.*

In simple words, the property of chain growth means that the blockchain stored by any honest party is increasing over time with the addition of blocks to the chain.

Definition 6.8 (Chain Quality) *The property of **chain quality** states that in $l \in \mathbb{N}$ consecutive blocks belonging to a chain C of a honest party there will be at maximum $\mu \cdot l$ adversary blocks, where $0 < \mu \leqslant 1$ is the chain quality coefficient.*

Informally, the chain quality property expresses the fact that the ratio of adversary blocks in any segment of a chain stored by an honest party will be no more than a fraction τ, where τ is the fraction of resources belonging to an adversary [27].

Definition 6.9 (Common Prefix) *The property of **common prefix** states that given two chains C_1 and C_2 belonging to honest parties at time slots slt_1 and slt_2, with $\mathrm{slt}_1 < \mathrm{slt}_2$, then $C_1^{\lceil k} \preceq C_2$, where $C_1^{\lceil k}$ denotes the chain obtained from C_1 by removing the last k blocks, where $k \in \mathbb{N}$ is the common prefix parameter.*

Informally, the common prefix states that considering the chains of two honest nodes at two different time slots, the shortest chain is a prefix of the longest one up to the common prefix parameter k.

6.2.2 *Assessment*

In this section, we assess the security of our work based on the properties of *common prefix*, *chain growth*, and *chain quality*. We show that while the proof of the chain growth is trivial, the proof of chain quality is more elaborated. Moreover, since the common prefix definition does not account for modifications, some modifiable blockchain protocol does not inherently maintain the property of common prefix starting from a blockchain that satisfies it. To account for modifications, Deuber et al. introduced the definition of *editable common prefix* [27]. However, our work does not substitute or delete blocks. Instead, it allows the authorised parties to compute collisions for a targeted rewriting such that the chain of blocks is not modified.

CHAIN GROWTH To show the fact that the chain growth property is maintained we note that our work does not delete blocks. Instead, it is a targeted rewriting of transactions in a blockchain. Even when the request of the data subject is to remove a transaction, we replace it with a placeholder to delete the original information and remove any reference to it. If we suppose that the original blockchain satisfies the chain growth property, our work does not interfere with the ability of the chain to grow. Indeed, the chain will grow as before if new transactions are bundled into blocks by the ordering service.

CHAIN QUALITY Let us suppose the presence of an attacker $\mathcal{A}$ that holds a fraction of computational power up to μ. $\mathcal{A}$ can follow different strategies to increase the fraction of malicious nodes in the chain of an honest node. For instance, the attacker $\mathcal{A}$ could extend the chain of an honest party by proposing additional malicious blocks. However, the creation of blocks is a responsibility of the ordering service in the Fabric architecture. Therefore, the attacker has two strategies to attack the generation of new blocks.

The first strategy is to submit as many malicious transactions as possible to the ordering service to create a malicious block full of its transactions. However, the adversary computational power will be up to μ, where the computational power represents its transaction creation capacity. As a result, the chain will contain a fraction of malicious blocks up to $\mu \cdot l$, where l is the length of the portion of the chain we are inspecting. Moreover, a similar situation could happen in the original non-modifiable architecture. Hence, our work does not allow an attacker to increase its fraction of malicious blocks.

The second strategy can be adopted when an attacker controls a fraction up to μ of the ordering service nodes - which can be now seen as the attacker computational power. In such a case, the malicious ordering service portion will be able to modify transactions before their insertion into a block. However, whenever a block is created and broadcasted to the peers, transactions are not executed by default. Instead, the peers that receive the block check the validity of the transactions and mark them as valid or invalid before executing them and updating the ledger. Even in the case in which the attacker succeeds in modifying transactions so that peers will no be aware of the modification, the fact it holds up to μ power in the ordering service node will stop him from being able to produce more than a fraction of μ malicious blocks. As in the previous case, this attack can be carried out in the original non-modifiable architecture as well when an attacker can take control of part of the ordering service process. Therefore, our work does not give an advantage to the attacker compared to the original architecture.

An alternative attacking strategy could be targeting the rewriting of blocks that may include the ability to delete blocks or to modify them. However, since our work does not allow to remove blocks from the chain, it is not possible to increase the share of the malicious block in a portion of the chain with l blocks by removing honest blocks. Therefore, the attacker has to modify transactions in a block so that the block becomes malicious. Again, we envision two possible strategies.

The first strategy can be used when the creation of the modifiable transaction has been done correctly without malicious intervention. This may happen when an attacker gains control of a share μ of ordering service nodes at a time slot $slt_2 > slt_1$, where slt_1 represents the time slot in which the transaction has been bundled into the block by an honest node. Due to the fact that the adversary $\mathcal{A}$ does not possess the knowledge of the trapdoors that unlock the ability to find collisions, the chameleon-hash behaves as a collision-resistant hash. Therefore, $\mathcal{A}$ needs to compute collisions via brute force attack, which is impractical if the bits of security provided by the hash function are enough and the chameleon-hash function is collision-resistant for all parties unaware of the trapdoors.

The second strategy can be used when the creation of the modifiable transaction has been done maliciously by an attacker controlling the malicious portion of the ordering service. This may happen when the ordering service node uses as the public hashing key its own public key or a public key different from the one belonging to the creator of the transaction. This creates a block that contains a set of malicious transaction. Since the attacker controls a fraction up to μ ordering service nodes, the malicious transactions in a portion of the chain of l blocks will be at most $\mu \cdot l$. If the attacker decides to modify the transaction it controls, the fraction of malicious transactions in the portion of the chain will not increase.

COMMON PREFIX Trivially if no redaction operations have been performed on the chain, the property is satisfied assuming that the original architecture meets the *common prefix*. Even when a transaction modification happens, the hashes of the blocks are not modified thanks to the design of the chameleon-hash function. Therefore, our work does not insert the possibility of creating inconsistencies in the chains of a hones party.

6.3 VALIDATION WITH EXPERT INTERVIEWS

In this section, we present the evaluation performed through expert interviews which constitutes the second static analysis of our reference architecture. During this phase, we analyse the structure of the artefact we developed, and we formalise the set of qualities that will be used to evaluate the design.

6.3.1 *Formalisation of Design Qualities*

This section introduces and formalises the qualities - or properties - of the design that we will assess during our interviews with experts. We will later evaluate whether our design achieves these properties taking into consideration the threat model we defined previously.

6.3.1.1 *Integrity*

Definition 6.10 (Integrity) ***Integrity*** *is the property that data or information have not been altered or destroyed in an unauthorised manner.*

Informally, integrity is the property that ensures data have not been modified or cancelled by an unauthorised party. A similar definition of integrity can be found in [21].

When in relation to our design, we would like the ledger to maintain the *integrity* property. We can, therefore, derive from the integrity property our definition of ledger integrity.

Definition 6.11 (Ledger Integrity) ***Ledger Integrity*** *is the property that data or information contained in a blockchain ledger have not been altered or destroyed in an unauthorised manner.*

For our evaluation, we will specifically refer to the part of the ledger of Hyperledger Fabric called transaction log, i.e., the chain of blocks that stores the cryptographically linked list of blocks.

6.3.1.2 *Tamper-proofness*

Definition 6.12 (Tamper-proofness) *A blockchain ledger is said to be* ***tamper-proof*** *when it is not possible to modify data on the ledger if not by brute-forcing the cryptographic hash or by hijacking the consensus mechanism.*

In Hyperledger Fabric, the consensus mechanism is represented by the endorsement policies that define which parties need to endorse a transaction before it is published. Endorsement policies are defined at the moment in which the chaincode that creates a transaction is deployed in the network.

Definition 6.13 (Weak Tamper-proofness) *A blockchain ledger is said to be* ***weakly tamper-proof*** *if an unauthorised party cannot modify data on the ledger if not by brute-forcing the cryptographic hash or by hijacking the consensus mechanism.*

Compared to the previous definition of a tamper-proof blockchain ledger, we relax the requirements of this property to render it suitable for a redactable blockchain. Any redactable blockchain cannot achieve the original *tamper-proof* property as data modification or deletion are features of such proposals. Therefore, we formalised a weaker notion of tamper-proofness to consider authorised and agreed modifications as valid.

6.3.1.3 *Tamper-evidence*

Definition 6.14 (Tamper-evidence) *A blockchain ledger is said to be* ***tamper-evident*** *if the ledger cannot be tampered with in an unnoticed manner.*

Informally, we say that the *tamper-evidence* quality is present if any modification is evident and can be seen by the other parties in the network.

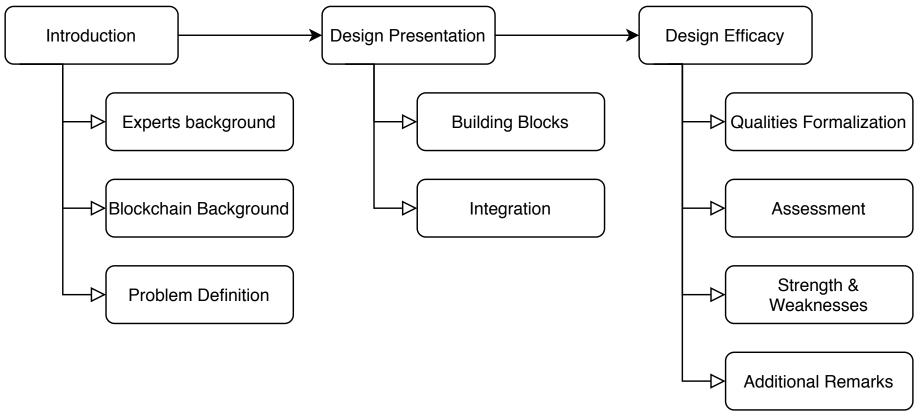


Figure 15: Structure of Experts Interviews

6.3.2 *Immutability*

Definition 6.15 (Immutability) *A blockchain ledger is said to be* ***immutable*** *when it is both tamper-proof and tamper-evident.*

Similarly to what we proposed above in the case of tamper-proofness, we can introduce the notion of weak immutability.

Definition 6.16 (Weak Immutability) *A blockchain ledger is said to be* ***weakly immutable*** *when it is both weakly tamper-proof and tamper-evident.*

6.3.3 *Assessment through Expert Interviews*

This section presents the structure of the validation interviews we held with experts and reports the results of the assessment. We divided each validation interview into three main phases represented in Figure 15. First, we gathered information on the interviewees' background to assess their familiarity with the realm of cryptography and the concept of blockchain. Besides, we evaluated the familiarity of the experts with Hyperledger Fabric. Even though experts' acquaintance with Fabric was not a prerequisite, the assessment gave us insights on the level of background information to provide during our second phase. Second, we presented the reference architecture we designed with a focus on the building blocks and their integration into an existing architecture. Last, we introduced the definition of the qualities we wanted to assess, and we asked experts to express their opinion. Through the use of semi-structured interviews, we allowed the experts to present additional comments on the architecture.

We interviewed a total of 7 experts with cryptography or blockchain background. In particular, experts have 1 to 6 years of experience in cryptography developed both in an academic setting and in a professional environment. In contrast, the prior experience of experts with blockchain was lower, ranging from almost no experience to 5

Reviewers	Ledger Integrity	Tamper-proofnes	Weak tamper-proofness	Tamper-evidence	Immutability	Weak immutability
R1	+	- -	+	++	- -	+
R2	+	- -	+	++	- -	+
R3	=	- -	=	++	- -	=
R4	+	- -	+	++	- -	++
R5	=	- -	+	++	- -	++
R6	++	- -	++	++	- -	++
R7	+	-	++	++	- -	++

Table 3: Summary of Experts Assessment

years. Acquaintance with Hyperledger Fabric was lower than the one with the blockchain, but we did not include previous knowledge of Fabric as a requirement to participate in the evaluation process. The summary of the assessment can be found in Table 3. We summarised a slightly positive opinion with the + symbol and a stronger positive opinion with ++. Similarly, we represent a slightly negative opinion with the − symbol and a stronger negative opinion with −−. To account for reviewers who did not feel comfortable in expressing their opinion on some qualities based on our presentation, we introduce the symbol =.

6.3.4 *Discussion of the Assessment*

This section discusses the opinions and the comments that we collected during the experts' review of the architecture.

LEDGER INTEGRITY As can be observed from Table 3, the opinion of experts on the ability to maintain the integrity of the ledger was positive or slightly positive. The main comments we received on this property regards the threat model and the definition of *unauthorised*: except for one, all interviewees referred to the threat model when assessing the integrity of the ledger and some of them would have preferred to receive a more precise definition of unauthorised.

TAMPER-PROOFNESS Trivially, we validated that the original quality of tamper-proofness cannot be satisfied by a redactable blockchain that employs chameleon-hash functions during the hashing process. This is because chameleon-hash functions are designed so to allow parties that possess the private key(s) to compute a collision for the hash value and to modify data such that the Merkle tree's root is not changed. Therefore, tampering with the ledger is possible. One notable comment on tamper-proofness regards its formalisation: one reviewer pointed out that, although tamper-proofness is ultimately connected with immutability, if the meaning of tampering is considered as *malicious modification* the design is capable of retaining this property.

WEAK TAMPER-PROOFNESS Similarly to ledger integrity, we received positive or slightly positive opinions on weak tamper-proofness. Recalling that the definition of weak tamper-proofness accounts for authorised modifications, experts agree on the fact that the design appears to satisfy this property. Therefore, we validated that, although the original tamper-proofness cannot be met unless we consider tampering as malicious, the property of weak tamper-proofness can be regarded as a satisfiable alternative.

TAMPER EVIDENCE Among the various properties, tamper evidence was the one in which experts showed a strong positive feeling. We, therefore, validated that through the Proof-of-Redaction mechanism we introduced, modifications and deletions are visible to the participants, and the presence of their signature certifies that they agreed on the proposal and took part in the process. Moreover, we validated that hiding the randomness into a commitment is a sound choice to avoid that parties can redact their ledger before reaching a collective agreement.

IMMUTABILITY Since our formalisation of immutability requires that tamper-proofness is satisfied, we can conclude that it is not possible to achieve perfect immutability. However, we also validated that it is not always needed to fulfil such property and its presence strongly depends on the particular application.

WEAK IMMUTABILITY Given the impossibility of maintaining perfect immutability and authorising redactions, we formalised the property of weak immutability as the presence of both tamper-evidence and weak tamper-proofness. Even though the comments we received where positive, we received feedback on the formalisation of such property. On the one side, some experts argued that given the fact that immutability is quite a strong property, a different formalisation of the property that accounts for modifications could have been formulated. On the other side, we received positive comments regarding the ability of weak-immutability to capture the presence of authorised changes. Regardless of the comments, a majority of the reviewers were positive on the ability of the design to reach this level of immutability. Depending on the application, several interviewees also recognised that such *weaker* immutability might be desired.

6.3.5 *Limitations of the Assessment*

In terms of limitations, we identified the following as possible weak points of our evaluation process.

First of all, the low experience of experts with the architecture of Hyperledger Fabric sporadically created some misunderstanding in the overall discussion. This limitation was pointed out directly during the review process. Therefore, further evaluation with experts acquainted with Fabric or longer evaluation sessions with more substantial background could be performed to obtain a more precise overall evaluation.

Second, we assessed the qualities of our design during an interview. Therefore, we based the assessment on the presentation we gave during each 60 to 90 minutes session. Taking into consideration the complexity of the design that combines various cryptographic primitives, the complexity of the architecture in which we proposed the integration, and the limited time available, it was hard to gather a precise assessment. Most of the reviews we collected express a positive or negative comment regarding the ability to reach the declared qualities. Some of them, instead, did not feel comfortable in providing an evaluation due to the short time available to digest a sophisticated design. Therefore, two sessions ended before we were able to assess the properties because interviewees felt uncomfortable in providing feedback. As a result, we found it hard to build an accurate summative assessment. On the other hand, we believe the review process shows a robust formative component which can be used as a base to derive future work and improvements.

Third, the fact that we decided not to specify a precise secret sharing and commitment scheme to maintain certain freedom to target the specific application needs may have limited the ability of experts to evaluate the design. Even though we stress the fact that the precise scheme should be decided based on the application needs, further evaluations may be needed taking into consideration an exact instantiation of secret sharing and commitment schemes.

6.3.6 *Answer to Research Question RQ2*

This section provides the answer to *RQ2* considering the results of the evaluation as a starting point. During the reviews, we asked experts to provide their opinion on the impact that the use of redaction could have on an existing blockchain architecture. While we showed that the requirements for a *robust transactional ledger* are not violated in our design, we felt the need to include expert's opinions regarding the impact of redaction on the integrity and immutability properties. We formalised the definitions of these properties previously in this Chapter.

SQ1: To what extent the integrity of the blockchain suffers from this modification? We formalised the integrity of the blockchain as the integrity of the ledger. To be precise, in this particular context, we refer to the ledger as the hash-chained list of blocks. We assessed ledger integrity during the reviews, and we received slightly positive opinions. Due to the limitations we presented in the previous section, it was not possible to derive a yes or no answer. However, expert's opinion shows that, if assessed with the proper threat model and the correct assumptions in place, the architecture is designed to favour the presence of such quality.

SQ2: To what extent the immutability of the blockchain suffers from this modification? We formalised the property of immutability as the situation in which both tamper-proofness and tamper-evidence are present. The assessment showed that the immutability property does not hold in a redactable blockchain. However, we introduced a slightly weaker definition of tamper-proofness and immutability that considers the fact that a redactable blockchain is not designed to be immutable. Similarly to the previous assessment, expert's opinion shows that considering a proper threat model and a correct set of assumptions, the architecture is designed to favour the presence of weak tamper-proofness. The presence of tamper-evidence through the use of the Proof-of-Redaction received a stronger positive opinion with less dependency on the threat model. Therefore, we can conclude that weak immutability holds as long as both tamper-evidence and weak tamper-proofness hold.

6.4 PERFORMANCE EVALUATION

In this section, we present the evaluation of the performance of our implementation of the chameleon-hash function with ephemeral trapdoor that we described in Section 5.3.1. This last part of the evaluation represents the dynamic analysis of the system to test the performance of its main building block.

We implemented the function using Go 1.14.2 and tested its execution on a virtual machine with an Intel Core i7-8750H CPU @ 2.20GHz running Ubuntu 20.04. We executed the five algorithms 1000 times, and we computed the average, which is reported in Table 4. Additionally, we indicate the number of multiplications in modulo, exponentiations in modulo, and the number of primes generated during each operation.

As can be noted from the performance in Table 4, the `Setup` is the operation that requires the highest amount of time. The reason for the high amount of time needed for the setup phase is the creation of a 2048-bit long prime e. However, the setup is a one-time operation executed when a new user joins the network. Therefore, from a prac-

Operation	#Multiplications	#Powers	#Primes[bit]	Time[ms]
Setup	/	/	1[2048]	785.14
CKGen	/	2	2[1024]	189.64
Hash	2	4	2[1024]	191.57
Check	1	1	/	7.47
Adapt	2	3	/	22.37

Table 4: Performance of Chameleon-Hash with Ephemeral Trapdoors

tical perspective, it does not slow down the hashing process, and it does not impact on the performance of the architecture significantly.

The generation of the RSA modulo and the private exponent d is done in `CKGen`. Since a fresh ephemeral trapdoor needs to be computed for each new hash, the performance of this operation impacts the speed of `Hash` as well because `CKGen` is used to compute the second ephemeral trapdoor. As can be seen from the difference between the execution time of `CKGen` and `CHash`, the actual hashing process requires very little time.

`Adapt` is a very efficient operation as well. The fact that it requires less time to execute compared to `Hash` resides in the absence of prime number generation primitives in the function body.

Last, the most used operation - and the fastest one as well - is `Check` . It is required both during the validation of a new block and the validation of the redaction proposal. The little execution time needed for this operation introduces very little overhead in the validation process.

7 CONCLUSION

In this work, we designed and presented a reference architecture of a redactable blockchain, i. e., a blockchain that supports deletion and modification of approved transactions. We employed chameleon-hash functions with ephemeral trapdoor to introduce transaction-level modifiability to redact single transactions without impacting the chain of blocks.

To ensure that a single party does not have the power to compute collisions through the use of the trapdoor, we divide the secret into shares proposing the use of a weighted and verifiable secret sharing scheme.

We described a redaction process that makes use of the standard transaction model of Hyperledger Fabric, and we introduced the Proof-of-Redaction. This mechanism allows parties in the network to approve and reach consensus on a redaction proposal before the actual redaction. To avoid that parties in the network redact the content of the transaction before participants reach a collective agreement, we hide the randomness that produces the collision into a commitment.

To validate our design, we performed in-depth interviews with cryptography and blockchain experts who expressed positive feelings regarding the ability of the design to retain some fundamental properties of a blockchain.

We believe that our work has the ability to alleviate the frictions between the highly praised immutability of a blockchain and some data subject's rights presented in the GDPR.

7.1 ANSWER TO THE RESEARCH QUESTIONS

This section reports the answers to the two research questions we identified at the beginning of our work. Since the answers to the various subquestions can be found throughout the document, here we focus on building a complete answer to the main research questions. We investigated two main research questions

RQ1: *Should we modify blockchain technology so that it is possible to alter or delete transactions to comply with Art. 16 and Art. 17 of GDPR?*
To understand which requirements are imposed by the Regulation on the management of personal data in a blockchain, we reviewed various related work on the contrasts between blockchain immutability and the GDPR. We validated with a legal expert that in many cases, hashes of personal data, transactional data, and user's public keys are considered personal data when

stored in a blockchain. Therefore, we conclude that, at the moment of writing, a modification or deletion of personal data in a blockchain is a necessary - but not sufficient - requirement to achieve compliance. We designed a reference architecture for a redactable blockchain leveraging the use of chameleon-hash functions with ephemeral trapdoor. We discussed who should have the right to propose modifications, and we presented a process through which parties in the network reach consensus on a proposed change. The redaction agreement is published as proof that history has been modified.

RQ2: How does the modification of the blockchain impact its properties?
To assess the impact that the redaction process introduces into an existing architecture, we performed a static analysis of the fundamental properties of a blockchain. First, we provided a theoretical discussion on the key properties of a *robust transactional ledger*. This concept has been lately used to assess the design of a redactable blockchain in related work. We showed we do not violate any of the requirements to build a robust transactional ledger, and we concluded that our work does not weaken an existing architecture. Second, we discussed and validated our work during interviews with cryptography and blockchain experts to assess the integrity and immutability of the design. While a redactable blockchain cannot meet the standard immutability, we introduced a weaker notion of immutability that considers the presence of parties who are authorised to produce collisions. Experts expressed an average positive feeling on the ability of our work to retain integrity and weak-immutability of the ledger.

7.2 LIMITATIONS

The aims of this thesis were (i) to discuss whether modifications should be permitted on a blockchain to ease compliance with the GDPR, (ii) to analyse the frictions between some data subject's rights and the way a blockchain manages data and to propose an architecture to alleviate the identified conflicts, (iii) to discuss who should have the right to introduce modifications, and (iv) to what extent some critical properties of a blockchain are affected if we allow modifications to happen. We believe that, while the classification of data on a blockchain as personal depends on the particular application and it is not trivial, our architecture contribute to reduce the frictions without a significant impact on the key properties of a blockchain. However, we acknowledge that some components of our work could be susceptible to discussion.

The first limitation of our work concerns with the uncertainty of the legal landscape of personal data management on a blockchain. It is not clear whether personal data on a blockchain will continue to be deemed as such even when more robust hashing mechanisms will be used or when additional privacy guarantees will be put in place. Therefore, we developed our architecture starting from unsettled arguments that could weaken the design if significant changes in the legal landscape develop shortly.

Another limitation regards the actual proof of the fact that the artefact works in practice. While we tested a working implementation of chameleon-hash function with ephemeral trapdoor, and we validated our architecture with experts through semi-structured interviews, a proof-of-concept of the complete architecture should be developed to assess the efficacy of the design in practice.

Concerning the implementation of the hash function, we recognise that a Public-Key Infrastructure (PKI) based on RSA could be considered as obsolete and more recent PKIs based on elliptic curves may be used to increase the security of the scheme and to improve its performance. However, due to the fact that the security properties of chameleon-hash functions have been proven for a limited set of constructions, we decided to use existing work rather than building a construction by ourselves.

To conclude, the last limitation we identified concerns with the threat model in which the design has been evaluated. While we believe that compliance in a permissioned blockchain is easier to achieve compared to a permissionless blockchain and that the honest-but-curious model captures a fair representation of the permissioned environment, we recognise the need to relax some constraints to produce a more thorough evaluation.

7.3 FUTURE WORK

The limitations we described in the previous section could potentially steer future works.

One limitation we identified concerns with the uncertainty of the legal landscape on which we based our analysis of personal data management in a blockchain. While it is not possible for us to clarify the legal matter, future research in the field can take into account the legal changes and adapt the architecture accordingly.

With regards to the design, we propose as future work the formal verification of some security properties. While the security of building blocks have already been proved, it should be possible to formalise our design as a cryptographic protocol and use an automated verification tool, such as ProVerif, to check a subset of the security properties. Additional future work on the design should focus on a more in-depth analysis of the case of dependent transactions. We stress that this analysis should be performed when the use case because there might be no need to modify transactions dependent on a redacted one.

For what it concerns the implementation of the design, we identified to the choice of a precise instantiation of secret sharing and commitment schemes as the first step towards a working proof-of-concept. Similarly, a construction of chameleon-hash functions based on different PKIs might become available, and future work could base new development on those constructions.

Last, we propose as future work the assessment of our design assuming a more permissive threat model. While we have already discussed that it is possible to relax the non-collusion assumption by adequately setting the threshold in the secret sharing scheme, we recognise that many application of blockchain require to work in untrusted settings. Therefore, the assessment of the design in a threat model that reflects an untrusted scenario represents a relevant future work.

BIBLIOGRAPHY

[1] Satoshi Nakamoto. 'Bitcoin: A Peer-to-Peer Electronic Cash System'. In: *Cryptography Mailing list at https://metzdowd.com* (Mar. 2009).

[2] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. 'Design Science in Information Systems Research'. In: *MIS Quarterly* 28.1 (2004), pp. 75–105. ISSN: 02767783. URL: http://www.jstor.org/stable/25148625.

[3] Vijay Vaishnavi and B Kuechler. 'Design Science Research in Information Systems'. In: *Association for Information Systems* (Jan. 2004).

[4] European Union Agency for Network and Information Security. *Distributed Ledger Technology & Cybersecurity*. Dec. 2016. DOI: 10.2824/80997.

[5] Leslie Lamport. 'The Part-Time Parliament'. In: *ACM Transaction on Computer Systems* 16.2 (May 1998), pp. 133–169. ISSN: 0734-2071. DOI: 10.1145/279227.279229. URL: https://doi.org/10.1145/279227.279229.

[6] Stuart Haber and W. Scott Stornetta. 'How to Time-Stamp a Digital Document'. In: *Journal of Cryptology* 3.2 (Jan. 1991), pp. 99–111. ISSN: 0933-2790. DOI: 10.1007/BF00196791. URL: https://doi.org/10.1007/BF00196791.

[7] David Mazières and Dennis Shasha. 'Building Secure File Systems out of Byzantine Storage'. In: *Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing*. PODC '02. Monterey, California: Association for Computing Machinery, July 2002, pp. 108–117. ISBN: 1581134851. DOI: 10.1145/571825.571840. URL: https://doi.org/10.1145/571825.571840.

[8] Nick Szabo. *Bit Gold*. https://nakamotoinstitute.org/bit-gold/. Dec. 2005.

[9] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. 'NISTIR 8202 Blockchain Technology Overview'. In: *National Institute of Standards and Technology* 8202 (Oct. 2018).

[10] Rui Zhang, Rui Xue, and Ling Liu. 'Security and Privacy on Blockchain'. In: *ACM Computing Surveys* 52.3 (July 2019). ISSN: 0360-0300. DOI: 10.1145/3316481. URL: https://doi.org/10.1145/3316481.

[11] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. 'Blockchain challenges and opportunities: A survey'. In: *International Journal of Web and Grid Services* 14.4 (2018), pp. 352–375.

[12] Leslie Lamport, Robert Shostak, and Marshall Pease. 'The Byzantine Generals Problem'. In: *ACM Transactions on Programming Languages and Systems* 4.3 (Juy 1982), pp. 382–401. ISSN: 0164-0925. DOI: 10.1145/357172.357176. URL: https://doi.org/10.1145/357172.357176.

[13] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. 'An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends'. In: June 2017. DOI: 10.1109/BigDataCongress.2017.85.

[14] Miguel Castro, Barbara Liskov, et al. 'Practical Byzantine fault tolerance'. In: *OSDI*. Vol. 99. 1999. 1999, pp. 173–186.

[15] Nick Szabo. *Smart Contracts*. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html. 1994.

[16] UK Government Office for Science. *Distributed Ledger Technology: beyond blockchain*. 2016.

[17] Nick Szabo. 'Formalizing and Securing Relationships on Public Networks'. In: *First Monday* (Sept. 2016). http://journals.uic.edu/ojs/index.php/fm/article/view/548/469.

[18] Vitalik Buterin. 'A next-generation smart contract and decentralized application platform'. In: *Ethereum White Paper* (2014).

[19] Vitalik Buterin. *On Public and Private Blockchains*. https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/. Aug. 2015.

[20] Gideon Greenspan. *Avoiding the pointless blockchain project*. https://www.multichain.com/blog/2015/11/avoiding-pointless-blockchain-project/. 2015.

[21] Karl Wüst and Arthur Gervais. 'Do you Need a Blockchain?' In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. June 2018, pp. 45–54.

[22] Tommy Koens and Erik Poll. 'What blockchain alternative do you need?' In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, Sept. 2018, pp. 113–129.

[23] Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton Andrade. 'Redactable Blockchain–or–Rewriting History in Bitcoin and Friends'. In: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2017, pp. 111–126.

[24] David Derler, Kai Samelin, Daniel Slamanig, and Christoph Striecks. 'Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based'. In: *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[25] Xianji Cai, Yanli Ren, and Xinpeng Zhang. 'Privacy-protected Deletable Blockchain'. In: *IEEE Access* (2019).

[26] N. Lee, J. Yang, M. M. H. Onik, and C. Kim. 'Modifiable Public Blockchains Using Truncated Hashing and Sidechains'. In: *IEEE Access* 7 (Nov. 2019), pp. 173571–173582. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2956628.

[27] Dominic Deuber, Bernardo Magri, and Sri Aravinda Krishnan Thyagarajan. 'Redactable blockchain in the permissionless setting'. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. May 2019, pp. 124–138.

[28] Jing Xu, Xinyu Li, Lingyuan Yin, Bingyong Guo, Han Feng, and Zhenfeng Zhang. 'Redactable Proof-of-Stake Blockchain with Fast Confirmation'. In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 1110.

[29] Ivan Puddu, Alexandra Dmitrienko, and Srdjan Capkun. 'µchain: How to Forget without Hard Forks'. In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 106.

[30] M. Florian, S. Henningsen, S. Beaucamp, and B. Scheuermann. 'Erasing Data from Blockchain Nodes'. In: *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*. June 2019, pp. 367–376. DOI: 10.1109/EuroSPW.2019.00047.

[31] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Bernardo Magri, Daniel Tschudi, and Aniket Kate. 'Reparo: Publicly Verifiable Layer to Repair Blockchains'. In: *ArXiv* abs/2001.00486 (Jan. 2020).

[32] Victor Gates. 'RTM: Blockchain That Support Revocable Transaction Model'. In: *ArXiv* abs/2001.11259 (Jan. 2020).

[33] European Commission. *General Data Protection Regulation*. May 2018. (Visited on Mar. 2, 2019).

[34] Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Springer International Publishing AG 2017, Jan. 2017. ISBN: 978-3-319-57958-0. DOI: 10.1007/978-3-319-57959-7.

[35] Michèle Finck, European Parliament, European Parliamentary Research Service, and Scientific Foresight Unit. *Blockchain and the General Data Protection Regulation: can distributed ledgers be squared with European data protection law? : study*. Scientific Foresight Unit, 2019. URL: https://data.europa.eu/doi/10.2861/535 (visited on Apr. 22, 2020).

[36] Michèle Finck. 'Blockchains and Data Protection in the European Union'. In: *SSRN Electronic Journal* (Jan. 2017). DOI: 10.2139/ssrn.3080322.

[37] Article 29 Data Protection Working Party. *Opinion 05/2014 on Anonymisation Techniques, WP 216*. 2014.

[38] Luis-Daniel Ibáñez, Kieron O'Hara, and Elena Simperl. 'On blockchains and the general data protection regulation'. In: EU Blockchain Forum and Observatory. 2018.

[39] Diogo Duarte. 'An Introduction to Blockchain Technology From a Legal Perspective and Its Tensions With the GDPR'. In: *Cyberlaw Journal of the Cyberlaw Research Centre of the University of Lisbon School of Law-CIJIC* (2019).

[40] Article 29 Data Protection Working Party. *Opinion 1/2010 on the concepts of "controller" and "processor", WP 169*. 2010.

[41] Jean Bacon, Johan David Michels, Christopher Millard, and Jatinder Singh. 'Blockchain Demystified: a Technical and Legal Introduction to Distributed and Centralized Ledgers'. In: *Rich. JL & Tech.* 25 (2018), p. 1.

[42] Hugo Krawczyk and Tal Rabin. 'Chameleon Hashing and Signatures'. In: *Proceedings of the Network and Distributed Systems Symposium*. 2000.

[43] Xiaofeng Chen, Fangguo Zhang, and Kwangjo Kim. 'Chameleon Hashing Without Key Exposure'. In: vol. 3225. 2004, pp. 87–98. DOI: 10.1007/978-3-540-30144-8_8.

[44] Giuseppe Ateniese and Breno de Medeiros. 'On the Key Exposure Problem in Chameleon Hashes'. In: *Proceedings of the 4th International Conference on Security in Communication Networks*. SCN'04. Amalfi, Italy: Springer-Verlag, 2004, pp. 165–179. DOI: 10.1007/978-3-540-30598-9_12.

[45] Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. 'Chameleon-Hashes with Ephemeral Trapdoors - And Applications to Invisible Sanitizable Signatures'. In: *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part II*. Ed. by Serge Fehr. Vol. 10175. Lecture Notes in Computer Science. Springer, 2017, pp. 152–182. DOI: 10.1007/

978-3-662-54388-7_6. URL: https://doi.org/10.1007/978-3-662-54388-7%5C_6.

[46] Adi Shamir. 'How to Share a Secret'. In: *Communications of the ACM* 22.11 (Nov. 1979), pp. 612–613. DOI: 10.1145/359168.359176.

[47] G. R. Blakley. 'Safeguarding cryptographic keys'. In: *Managing Requirements Knowledge, International Workshop on*. Los Alamitos, CA, USA: IEEE Computer Society, June 1979, p. 313. DOI: 10.1109/AFIPS.1979.98.

[48] John Venable, Jan Pries-Heje, and Richard Baskerville. 'FEDS: a Framework for Evaluation in Design Science Research'. In: *European Journal of Information Systems* 25.1 (2016), pp. 77–89.

[49] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 'The Bitcoin Backbone Protocol: Analysis and Applications'. In: *Advances in Cryptology - EUROCRYPT 2015*. Ed. by Elisabeth Oswald and Marc Fischlin. Springer Berlin Heidelberg, 2015, pp. 281–310.

[50] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 'Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol'. In: *Advances in Cryptology – CRYPTO 2017*. Ed. by Jonathan Katz and Hovav Shacham. Springer International Publishing, 2017, pp. 357–388.

[51] Aggelos Kiayias and Giorgos Panagiotakos. 'Speed-Security Tradeoffs in Blockchain Protocols'. In: *IACR Cryptol. ePrint Arch.* 2015 (2015).

[52] A. Marsalek and T. Zefferer. 'A Correctable Public Blockchain'. In: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2019, pp. 554–561.