Politecnico di Torino

Politecnico di Torino

Cybersecurity

Academic Year 2025/2026

Cybersecurity Aspects of Permissioned Blockchain Systems: Theory and a Practical Proof of Concept

Supervisors:

Danilo Bazzanella

Piergiorgio Rettaroli

Candidate:

Jacopo Coniglio

Abstract

Construction projects depend heavily on coordination, yet trust among stakeholders is often missing. While software is common in the industry, reliance on a single central database creates a single point of failure. An administrator can silently edit or delete records, leaving no evidence of the change. This work investigates how to secure data integrity in such environments without slowing down daily operations. The proposed system uses Hyperledger Fabric but adapts it for practical performance. Storing massive BIM files directly on a blockchain is not feasible. Instead, a hybrid approach keeps the heavy files in a standard database while locking the security on the ledger. A cryptographic hash and timestamp record the state of every document. Even if the storage is external, any change to the file breaks the link to the blockchain, immediately revealing the tampering. A working prototype, developed with Accenture, validates this method. It features a web interface where engineers can view 3D models and automatically check their authenticity. During testing, a database attack was simulated. The system flagged the mismatch instantly because the file no longer matched the blockchain record. Governance tests also confirmed that no single party can force updates unilaterally. This shift to cryptographic verification removes the threat of silent corruption and creates an audit trail that does not depend on blind trust.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Objectives of the Thesis

This work explores how blockchain can be used to support the management of large industrial projects, with particular attention to trust, traceability, and control of shared information. The project was developed in collaboration with Accenture's Industry X in Milan, a global consulting and technology company. This collaboration made it possible to study the problem in concrete operational terms, rather than only at a theoretical level, and to assess how these ideas could be introduced into real project workflows.

The result of the work, conducted as part of an internship at the company and complemented by individual study, is a Proof of Concept (PoC) platform that links blockchain with digital project management processes. The platform is intended to act as a common environment where different administrators involved in a project can store and access documentation and operational records. The objective is straightforward: different companies and stakeholders should be able to work against the same version of the information, and it should always be possible to verify who introduced a change, when that change happened, and whether the data has been modified afterwards.

The need for this kind of solution becomes clear when looking at how complex industrial, construction and engineering projects are normally organized. These projects usually involve multiple actors such as clients, suppliers, contractors, project managers or engineers, each responsible for their own work packages, technical documents, approvals, and materials. In practice, this often leads to very simple but very expensive problems: outdated versions of the same document being shared in parallel, missing sign-offs on critical changes, unclear ownership of

certain files, and delays caused by having to reconcile conflicting information. A recurring difficulty is the lack of a single, reliable source of truth that all parties accept.

The proposed PoC addresses this issue by storing project information in a distributed ledger, using Hyperledger Fabric as the permissioned framework because it suits enterprise contexts and is well-suited to a PoC. Examples include giving a contractor an assignment, approving an event, and uploading an updated model. This makes it easier to check what happened and in which order, and it reduces the amount of time spent disagreeing over document versions or responsibilities.

1.2 Overview of Accenture

Accenture is a global consulting and technology company that works with organizations of all sizes to help them face digital and operational change. What makes the company stand out is its ability to combine business strategy with technological innovation in a very concrete way. It operates in more than 120 countries and employs around 779,000 people worldwide, with offices in locations such as Italy, Germany, the United States, India, and Singapore, and an annual revenue close to 70 billion USD (Accenture Newsroom, 2025). During the work experience carried out as part of this thesis, the work was conducted with the Industry X blockchain team at Accenture in Milan. The group was engaged in several industrial projects aimed at applying digital technologies to improve coordination and data reliability across different stages of production. The activities provided a practical view of how automation, data sharing, and system integration can support engineering processes and reveal the challenges that emerge when digital methods are introduced into established industrial workflows. Within Industry X, digital transformation is treated as a gradual process, one that connects existing industrial methods with new tools for design, monitoring, and collaboration (Accenture Industry X, 2025). Working with professionals in this environment offered a direct view of how large industrial projects are managed and where innovation can actually make a difference. It also gave this work a very practical dimension: ideas about blockchain and transparency were not only studied, but discussed and tested with people who deal with these challenges every day. This collaboration helped me understand how technology can fit into existing workflows and how solutions like the one developed in the Proof of Concept could add real value in similar industrial contexts[1, 2].

Chapter 2

Introduction to Cryptography

Cryptography is a key component of information security today and is necessary for understanding how data can be protected in digital systems. It ensures the confidentiality of data, allows its integrity to be checked, and its origin to be verified. The primary cryptographic methods outlined in this chapter are essential to understanding the security-related topics covered later in the thesis. Since the purpose is to explain the key concepts and how they are used rather than the theory behind them, complex math details have been intentionally omitted. The above concepts have a direct effect on the PoC, where cryptography provides user authentication, secure data handling, and document integrity maintenance in a permissioned blockchain scenario.

2.1 Symmetric Encryption

Symmetric encryption represents one of the two main types of encryption applied in cybersecurity. It protects confidential information with just one private key for both encryption and decryption. The same key must be shared and kept up to date by all authorized parties. Since symmetric encryption requires only one key, it is efficient and ideal for large data volumes, database encryption, and communication over networks. After discussing its primary goal, it is useful to see how symmetric encryption behaves in real-world scenarios and how the secret key is involved in it.

A randomized secret key is generated at the very beginning of the process. The system creates ciphertext that appears random and hard to understand through the use of the secret key and the plaintext as an input for the algorithm used for encryption. The same key is used to decrypt the ciphertext and recover the original

data after it has been sent to the intended recipient. The information is hard to understand without that key.

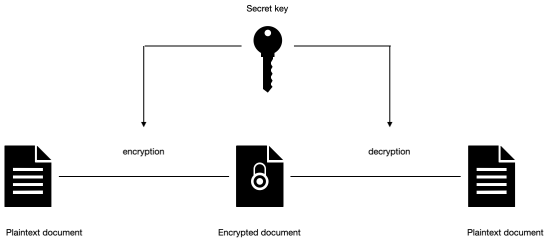

Figure 2.1: Symmetric Encryption

There are two main types of symmetric encryption algorithms:

1. Block ciphers, which encrypt data in fixed-size blocks. They are often used to secure files, databases, and network traffic. Each block of plaintext is transformed into ciphertext using the same key.

2. Stream ciphers, which encrypt data one bit or byte at a time. They are often applied to real-time communications, where continuous data flow requires speed and low computational cost.

Symmetric encryption continues to be one of the more widely used techniques for safeguarding data. Its effectiveness is its biggest benefit. In comparison with asymmetric algorithms, operations use less processing power because the same private key is used both for encryption and decryption. In addition to this, symmetric encryption is optimal for secure network communication, file systems, and large databases where scalability and speed are important. Another important benefit is its simplicity. Implementing and integrating the algorithms into existing structures is quite easy. When used with the correct key lengths and modes of operation, modern standards like AES, as defined by NIST FIPS 197, have been carefully examined and demonstrated to provide strong security. Because of these advantages, symmetric encryption is a trusted choice for applications that must find an agreement between high performance and strong protection. However, symmetric encryption also introduces significant challenges. The most critical one is key management. Since the same secret key is used for both parties, it must be exchanged and stored securely. All encrypted data can be decrypted if this key is intercepted or made public. It becomes very difficult to manage distinct keys for lots of users or services in big systems [3, 4].

The most widely known symmetric encryption algorithms are summarized in

Table 2.1. Each algorithm differs in structure, key length, and current level of security, which determines whether it is still recommended or considered obsolete.

Table 2.1: Common symmetric encryption algorithms [3, 4].

Algorithm	Cipher Type	Key Length (bits)	Status
AES (Advanced Encryption Standard)	Block cipher	128 / 192 / 256	**Recommended**
3DES (Triple DES)	Block cipher	112 / 168	Obsolete
Blowfish	Block cipher	32–448	Acceptable but outdated
Twofish	Block cipher	Up to 256	Recommended alternative
ChaCha20-Poly1305	Stream cipher (AEAD)	256	**Recommended**
RC4	Stream cipher	Variable	Insecure

2.2 Asymmetric Encryption

Nowadays, encryption is essential for keeping digital communication safe. Among the different methods, asymmetric encryption, also known as public-key cryptography, is one of the most important and widely adopted techniques in cybersecurity. It is essential to many systems that people use on a daily basis, such as HTTPS connections or digital signatures. In contrast to traditional encryption, which relies on a single shared key, this method depends on two independent keys that work together. Instead of using just one shared key, it works with two: a public one, which anyone can see, and a private one, which must stay secret and stored locally. The two keys are connected by a mathematical relationship so that a message locked with one key can only be opened with the other [5]. This technique fixes a common issue with symmetric encryption, which is that two parties need to agree on the same secret key in order to communicate securely. Asymmetric encryption eliminates the need for that initial exchange. Once the public key is shared, anyone can send encrypted data that only the owner of the private key can read. It's a simple idea but very effective for communication across open networks [5]. That said, asymmetric algorithms are not as fast as symmetric ones. They need more computing resources and time to run. Because of this, most systems use them mainly at the start of a secure session, just to share a secret symmetric key. After that, symmetric encryption takes care of the rest since it's faster for large amounts of data [5]. Finally, the strength of asymmetric encryption depends on how long the keys are. Longer keys offer better protection but also make the system slower. Practically, computer engineers have to find a compromise between security and performance depending on the context [5].

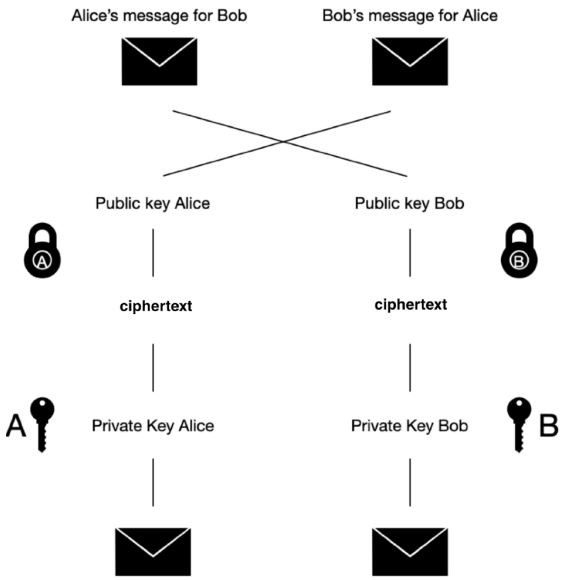

Figure 2.2: Asymmetric Encryption

To better understand the use of asymmetric encryption, it is useful to highlight the main algorithms that rely on this mechanism and their corresponding minimum key lengths, as suggested by IBM and NIST. The following table provides a summary of the most common asymmetric encryption algorithms, along with the minimum recommended key size to ensure adequate security levels in modern systems.

Table 2.2: Asymmetric encryption algorithms and their minimum recommended key lengths [5] [6] [7] .

Algorithm	**Min. Key Length (bits)**
RSA	2048
Elliptic Curve Cryptography (ECC)	256
DSA	2048
Diffie–Hellman (DH)	2048
ECDH (Elliptic Curve Diffie–Hellman)	256

2.3 Hash Functions

A **hash function** is a mathematical algorithm that transforms an input of arbitrary length into a fixed-length value, known as a *digest* or *hash value* [8]. Cryptographic hash functions are fundamental components of information security, ensuring data integrity, authentication, and non-repudiation in digital systems [8, 9].

These functions are characterized by the following essential properties [8]:

- **Deterministic:** the same input always produces the same output.
- **Avalanche effect:** even a small change in the input leads to a completely different hash.
- **Efficiency:** hash functions are designed to be computationally fast.
- **Preimage resistance:** it is infeasible to reconstruct the original message from the hash.
- **Collision resistance:** it is infeasible to find two distinct inputs that produce the same hash value.

The most widely adopted family of hash functions is the **Secure Hash Algorithm (SHA)** series, standardized by the *National Institute of Standards and Technology (NIST)*. The first versions of SHA were defined in the *Secure Hash Standard (SHS)* [8], while the most recent member of the family, SHA-3, was standardized later in *FIPS PUB 202* [9].

- **SHA-0** — introduced in 1993, now obsolete.
- **SHA-1** — 160-bit output; considered insecure due to proven collision vulnerabilities.
- **SHA-2** — includes SHA-224, SHA-256, SHA-384, and SHA-512, still considered secure and defined in [8].
- **SHA-3** — based on the Keccak algorithm and standardized in 2015 as defined in [9].

Table 2.3: Main SHA Versions and Output Sizes [8]

Version	Output Size	Status
SHA-0	160 bit	Obsolete
SHA-1	160 bit	Insecure
SHA-224	224 bit	Secure
SHA-256	256 bit	Secure
SHA-384	384 bit	Secure
SHA-512	512 bit	Secure
SHA3-224	224 bit	Secure
SHA3-256	256 bit	Secure
SHA3-384	384 bit	Secure
SHA3-512	512 bit	Secure

Hash functions are particularly relevant in distributed systems such as blockchains, where they ensure the immutability and integrity of data. For instance, **SHA-256** is employed in Bitcoin for block hashing and transaction verification, while **SHA-3** has been adopted in newer blockchain platforms [9].

2.4 Digital Signatures and Certificates

Digital signatures are one of the core mechanisms that allow two parties to trust data exchanged over an open network. In practical terms, a digital signature aims to answer two questions at once: Did this data really come from the claimed sender, and has it remained unchanged during transmission [10]? The signing process is conceptually simple. First, the sender computes a cryptographic hash of the message. This hash is a compact fingerprint of the original data. Then, that fingerprint is processed using the sender's private key, producing the digital signature. On the receiver's side, verification is symmetric: the receiver recomputes the hash of the received message and checks, using the sender's public key, that the signature corresponds to that same hash. If the check succeeds, the message is both authentic and intact [10, 11]. From an implementation point of view, the most widely standardized signature algorithms are DSA, RSA, and ECDSA [10, 11]. DSA relies on the discrete logarithm problem in modular arithmetic. RSA relies on the difficulty of factoring large integers. ECDSA uses elliptic curve cryptography to provide the same security level with much shorter keys, which makes it attractive for systems with performance or bandwidth constraints.

Table 2.4: Main digital signature algorithms [10] [12].

Algorithm	**Security Basis**	**Typical Key Size**
DSA	Discrete logarithm problem in modular arithmetic	$\geq$ 2048 bits
RSA	Integer factorization (hardness of factoring large RSA moduli)	2048–4096 bits
ECDSA	Elliptic curve discrete logarithm problem	$\approx$ 256 bits for RSA-equivalent security

However, a signature alone does not solve the problem of identity. A valid signature only proves that whoever signed the message had access to a given private key. It does not prove who that entity is in the real world. This is exactly the role of digital certificates. A digital certificate is a structured document that binds a public key to an asserted identity, such as an organization, a service, or a device. The certificate

is created and signed by a trusted third party called a Certification Authority (CA). The certificate contains the subject's identity, the public key, validity dates, and the CA's signature [13]. Any external party can then verify the CA's signature on the certificate and decide whether to trust the public key it contains. This model is the foundation of Public Key Infrastructure (PKI). In protocols such as HTTPS, the browser accepts the server's certificate only if it is correctly signed by a trusted CA and then uses the associated public key to verify that the server can produce valid signatures during the TLS handshake. In other words, the certificate links the key to an identity, and the signature proves control of that key [13].

2.5 Message Integrity

Confidentiality alone is not enough in a security system. Beyond keeping data secret, we must also ensure that data has not been modified in transit or storage. This property is referred to as message integrity [14]. A cryptographic hash function can detect accidental or malicious modifications because any change in the input produces a completely different digest. However, a standalone hash does not prove who generated it. An attacker could alter a message, recompute the hash, and forward both to the recipient. The recipient would see a consistent hash but would have no guarantee that it originated from the legitimate sender. For this reason, practical systems rely on Message Authentication Codes (MACs). A MAC is a short tag computed from the message and a secret key shared between the communicating parties. If the message changes, the MAC no longer validates; if an attacker does not know the secret key, they cannot forge a valid MAC. As a result, MACs provide both integrity and source authentication [14, 11].

HMAC

HMAC (Hash-based Message Authentication Code) is defined in NIST SP 800-107 and is built by combining a secure hash function such as SHA-256 with a shared secret key [14]. The construction is specifically designed to resist common attacks on naive "keyed hashing," and it is widely deployed in TLS, IPsec, SSH, and API signing schemes. Formally,

$$\mathrm{HMAC}(K, m) = H\Big((K \oplus \mathrm{opad}) \,\|\, H((K \oplus \mathrm{ipad}) \,\|\, m)\Big)$$

where K is the secret key, and *ipad* and *opad* are fixed padding constants.

CBC-MAC and CMAC

Some integrity mechanisms are built on symmetric block ciphers instead of hash functions. CBC-MAC authenticates a message by processing it with a block cipher

(e.g., AES) in Cipher Block Chaining (CBC) mode and using the final block as the tag. CMAC, standardized in NIST SP 800-38B, improves on CBC-MAC by handling variable-length messages securely and is typically preferred in modern designs [15].

Table 2.5: Common Message Authentication Code (MAC) constructions [14] [15].
.

Algorithm	**Built from**
HMAC	Hash function (e.g., SHA-256)
CBC-MAC	Block cipher in CBC mode
CMAC	AES-based block cipher

In summary, MACs solve a problem that plain hashing cannot: they prove that the data is intact and that it originated from a party that knows the shared secret. This is why MACs are embedded in essentially every secure communication protocol in use today [15].

2.6 Key Management and Public Key Infrastructure

A Public Key Infrastructure is a framework that allows users, devices, and applications to confirm who they are and to share information without risk [16]. It depends on public-key cryptography and digital certificates to ensure confidence in data transmission. In the Public Key Infrastructure, each participant is provided with a digital certificate, an authenticated digital record that serves to cryptographically bind the user's identity to a public key [16]. This makes it possible to exchange data with confidence, even between parties that have never met before. PKI is a core part of modern cybersecurity because it supports four key goals: confidentiality means that a message can be read only by the person or designated system; integrity certifies that the information received is exactly what was sent—this is typically achieved by allowing the recipient to detect unauthorized changes; authenticity means the person who sends a message is really the one they claim to be; non-repudiation ensures that the sender cannot later deny having sent that same message [16]. Public-key cryptography by itself provides mathematics to protect data. PKI adds an organizational and procedural layer so that public keys are tied to real identities and can be trusted in practical systems. PKI involves more than just keys or digital files. It combines software tools, secure hardware, and clear procedures managed by people. Its main building blocks include the following: Certificate Authority (CA): a trusted organization that issues and signs

digital certificates. The CA uses its private key to sign certificates so others can verify them. Registration Authority (RA): the RA checks the identity of an applicant before the CA issues a certificate. An organization can use the same entity as both CA and RA. Certificate Database: a storage system that keeps issued certificates and related metadata (for example, validity times). Central Directory: a secure index or repository for cryptographic keys and certificates. Certificate Management System: tools and protocols to create, issue, distribute, renew, and revoke certificates. Certificate Policy (CP): a public document that explains how the PKI operates, how identities are checked, and what rules are followed [16]. Digital certificates and their content. A digital certificate proves that a specific public key belongs to a named owner. Typical fields in a certificate include: the owner's Distinguished Name (DN); the owner's public key; the certificate issue date and expiration date; the issuing CA's DN; and the issuing CA's digital signature. Good certificates are tamper-resistant and include enough data to verify identity and trust. All certificates should have an expiration date so old keys are retired [16]. How certificate issuance works (simple flow). The user or device generates a private key and a matching public key. The requester creates a Certificate Signing Request (CSR) that contains the public key and identity attributes. The requester signs the CSR to prove possession of the private key. The RA (or CA) verifies the requester's identity and the CSR. After verifying the request, the CA signs the new certificate using its private key and then gives it to the requester. Anyone who receives the certificate can later confirm its authenticity by checking the CA's digital signature with the CA's public key [16]. PKI and common attacks. PKI reduces many risks but is not immune to attacks. One typical threat is a man-in-the-middle attack: an attacker intercepts or replaces public keys so that two parties unknowingly talk through the attacker. PKI defends against this by binding keys to verified identities via certificates. A more severe risk is the compromise of a CA's private key. If a CA key is stolen, an attacker could create valid-looking certificates for any name. That is why CA private keys—especially root CA keys—are protected with extreme care [16]. Root CAs and hierarchy. PKI commonly uses a hierarchy of CAs. The top level is the root CA. Root CAs often self-sign their certificates and then sign subordinate CAs. Because a root CA is so powerful, its private key is usually kept offline and highly protected. Root certificates often expire after 15–20 years. If a root CA or subordinate CA is compromised, the CA must publish the breach details and revoke affected certificates. Systems use revocation lists or online checks to avoid trusting compromised certificates [16].

Chapter 3

The History and Evolution of Blockchain Technology

3.1 1992 - The Crypto Anarchist Manifesto

Timothy C. May presented a clear picture of how strong cryptography could change personal communication and economic exchange. He argued that when cryptographic tools become widely available, people will be able to send messages, make agreements, and carry out transactions without revealing their legal identities. In such a context the name of a person will no longer be necessary for electronic interaction. Instead, reputation and the history of past behaviour will become the main ways for others to judge trustworthiness. This change shifts the source of social control from formal institutions to informal networks of trust. May also described the technical foundations that would make this development possible. He pointed to public key encryption, zero knowledge interactive proof systems, and new software protocols for authentication and verification. He believed that improvements in computing power and network speed would make these methods practical. He listed devices and platforms that could support encrypted markets and anonymous communication. These included faster personal computers, dedicated encryption hardware, smart cards, and high speed links. Together these technologies reduce the cost of private communication and increase the difficulty of effective surveillance. The manifesto considered the wider social and political impact of these technologies. May predicted that states would try to limit or control cryptographic tools by appealing to security and law enforcement reasons. He accepted that criminal actors might use anonymous networks for harmful purposes and that some illegal markets could appear. At the same time he compared the new cryptographic era to earlier technological shifts that changed power relations. He argued that just as the printing press weakened some old social authorities, cryptography could weaken

centralized control over information and property. For a study of cybersecurity and digital governance this argument highlights a central tension. Cryptographic freedom strengthens individual privacy and autonomy while at the same time posing difficult problems for regulation, law and the prevention of abuse.

3.2 1997 - Hashcash

Hashcash was proposed by Adam Back in 1997 as a practical response to a very specific problem on the early internet: large-scale abuse of open systems, especially email spam [17]. At that time, sending unsolicited email in massive volume was basically free for the sender. The cost (bandwidth, storage, attention, server load) was pushed entirely onto everyone else. Legal or policy-based solutions were slow and not globally enforceable. Back's approach was different. Instead of trying to block bad actors by identity or by filtering content, he tried to make abusive behavior economically unattractive at the technical level [17].

The core mechanism he introduced is what he called proof of work. The basic idea is that, before a message is accepted, the sender has to attach a small piece of evidence showing that they spent a measurable amount of computation to generate it. Concretely, the sender is required to find a value, sometimes described as a "stamp," such that when this value is combined with some metadata and hashed, the resulting hash output satisfies a public difficulty condition (for example, beginning with a certain number of zero bits) [17].

Two details are important here. First, finding such a value is intentionally costly. There is no shortcut. The only way to get a valid stamp is to try different nonces again and again until the hash output matches the difficulty target. That means the sender is literally burning CPU cycles and electricity. Second, verification is extremely cheap. The recipient, or the server, or anyone else, can check the stamp in essentially constant time. So the system is asymmetric by design: expensive to produce and cheap to verify [17].

This asymmetry is exactly what gives Hashcash its defensive power. A normal user sending a handful of legitimate emails per day can afford that small computational work without noticing. A spammer trying to send millions of messages cannot, because the total work scales with volume. In other words, Hashcash does not try to make messaging "secure" in the traditional sense. It tries to make abuse uneconomic.

Another point that matters historically is that Hashcash does not rely on any central authority. There is no issuer, no whitelist, and no gatekeeper that decides

who is allowed to participate. Anyone can generate a proof-of-work stamp locally. Anyone else can verify it. In Hashcash, trust is not assigned to a central authority. It is derived from the fact that the sender has actually performed the required work and can prove it. This idea is one of the elements that later appears again in cryptocurrency designs [17].

More than a decade later, when Satoshi Nakamoto published the Bitcoin white paper in 2008, he explicitly cited Hashcash as the model for Bitcoin's proof-of-work mechanism [18]. In Bitcoin, the process looks very similar on a mechanical level. Nodes, the so-called miners, take a block of pending transactions and repeatedly hash the block header while varying a nonce. They continue until the resulting hash falls below a publicly defined difficulty target. This is the same fundamental pattern described by Back: do measurable work, prove it with a hash, and let everyone else verify that proof almost instantly [17, 18].

The purpose, however, is different. In Hashcash, proof of work mainly serves as a rate-limiting tool. It is there to make spam, denial-of-service floods, and other high-volume attacks expensive. In Bitcoin, proof of work is repurposed to solve a deeper coordination problem: agreement on the state of a shared ledger in an open and adversarial environment, without trusting a central authority to act as referee [18]. Miners who successfully produce valid proof of work gain the right to append the next block of transactions. Because this costs real resources, rewriting history becomes economically difficult. The network ends up with a ledger that is both globally visible and, in practice, extremely hard to fake.

Seen from this perspective, Hashcash is a turning point. It is one of the first systems to treat computation itself as a scarce resource that can be used to regulate behavior online [17]. Bitcoin takes that same idea and applies it to digital money, showing that proof of work can be used not only to slow down abuse, but to establish consensus and protect economic value in a decentralized network [18].

3.3 1997 - The Idea of Smart Contracts

Nick Szabo introduced the concept of smart contracts in the late 1990s as a way to bring traditional contract logic into digital systems [19]. His main observation was that many kinds of economic relationships, such as payments, leases, and escrow, depend on clear conditions and predictable enforcement. In the physical world, these conditions are often handled by institutions such as banks, notaries, and courts. Szabo argued that some of this enforcement could instead be carried out directly by software, provided that the rules of the agreement were expressed in a

precise and verifiable form [20].

In Szabo's description, a smart contract is not only a legal document. It is also a piece of code that can automatically observe inputs, apply rules, and execute outcomes. The goal is to reduce uncertainty between the parties. Instead of trusting that the counterparty will behave correctly, both parties rely on a system that enforces the agreed terms by design [19]. For example, a simple digital payment escrow can be modeled so that funds are locked under predefined conditions and are released only when those conditions are met. The process does not require a human arbitrator in the middle, because the logic is embedded in the contract itself.

This approach has two direct consequences. The first is lower transaction risk. If the contract is self-enforcing, then it becomes harder for one party to take advantage of the other or to later change the conditions. The second is lower transaction cost. Automating enforcement can remove or shrink the role of intermediaries such as brokers and clearing houses [20]. Szabo repeatedly emphasized cost reduction as one of the main motivations for smart contracts. He described them as a way to make economic activity more efficient by minimizing the number of separate actors that must be involved in each step.

Szabo also linked smart contracts to security. He noted that once valuable assets move online, digital systems need a way to apply access control, timing, and conditional release rules with a high level of reliability [20]. In other words, it is not enough to store assets in a database. The system must also enforce the business logic that governs who can move those assets, under what circumstances, and in what order. Smart contracts were presented as a structured method for doing exactly that.

Although Szabo wrote before public blockchains existed, his work anticipates how smart contracts are later used on blockchain platforms. Modern blockchain systems execute contract code directly on a decentralized network instead of on a single server. This gives the contract two properties that Szabo considered important: predictable execution of terms, and reduced reliance on a central authority [19, 20]. In this sense, his work provided the conceptual foundation for programmable value transfer, long before platforms such as Ethereum made it widely accessible.

3.4 1998 - B-money

Wei Dai's "b-money" is a concise but remarkably prescient proposal for a permissionless digital cash system operated by pseudonymous participants over an open

network [21]. The design is presented in two variants.

In the first variant ("Protocol 1"), every participant maintains a replicated ledger of accounts and balances. Money is created by performing publicly verifiable computational work and broadcasting the result; once the network accepts the proof, the worker's account is credited. Ownership and transfers are expressed as signed messages from the current owner to the next, and all nodes update balances deterministically from those messages. This makes value transfer independent of any central issuer or clearinghouse: authority derives from verification rules and collective replication, not from institutions [21].

Dai also sketches contract enforcement without trusted intermediaries. Parties commit to contracts by publishing terms and stakes under their digital pseudonyms; if one side fails to perform, an agreed procedure debits a pre-specified penalty (for example, via escrow-style arrangements). The aim is the same one later pursued by smart-contract platforms: encode obligations as conditions that the network can observe and enforce, reducing reliance on courts or brokers [21].

Recognizing the operational burden of universal state replication, Dai proposes a second variant ("Protocol 2") in which a subset of authenticated "servers" keep the ledger, while ordinary users retain the ability to audit and challenge server misbehavior using cryptographic receipts. This foreshadows later separations of roles (full nodes vs. lightweight clients) and explores the scalability–decentralization trade-off explicitly [21].

Technically, b-money anticipates several pillars of blockchain systems: pseudonymous identities backed by digital signatures rather than legal names; proof-of-work issuance that turns computation into scarce money, with cheap verification; a replicated, append-only account record maintained by a peer network; and rule-based contract/escrow mechanisms that allow the network to enforce outcomes. Equally important are the open problems Dai flags: coordinating global agreement on state, deterring Sybil attacks, calibrating issuance so that computational progress does not cause runaway inflation, and making the system efficient enough for real use. In this sense, b-money articulates both the architectural blueprint and the research agenda that subsequent cryptocurrencies would try to complete [21].

3.5 2004 - Reusable Proof of Work

Hal Finney extends the Hashcash idea with Reusable Proofs of Work (RPoW): a system where a proof-of-work token can be reused and transferred between

pseudonyms without duplication. The key insight is that a proof of work, costly to produce and trivial to verify, can function as a scarce digital object. The central obstacle is double spending. RPoW addresses it by recording uniqueness and ownership on an attested server that cryptographically proves the code it runs and its issuance policy [22].

In the initial implementation, the server ran inside an IBM 4758 cryptographic coprocessor (FIPS 140-1 Level 4) and guaranteed that it would issue a new RPoW only in exchange for an input of equal value, such as a Hashcash stamp or another RPoW. Each transfer destroys the previous token and creates a new one, preserving scarcity and preventing duplication [22].

From a cybersecurity perspective, RPoW highlights three points: (i) trusted hardware and remote attestation as the foundation of a digital mint; (ii) role separation between clients, which produce proof of work and sign transfers, and the server, which prevents double spending and maintains ownership; and (iii) an explicit trade-off: it is not fully trustless, but it gains reusability and throughput. Bitcoin later reuses proof of work for distributed consensus, removing the central server, while RPoW remains the first working prototype of reusable digital scarcity [22].

3.6 2005 - Bit gold

Bit gold outlines a way to treat proofs of work as scarce, ownable units recorded in a public registry. A participant generates a proof of work over a challenge string, publishes the solution together with a timestamp and other metadata, and receives a title to that string. Titles are bound to public keys, and each new challenge incorporates the previous solution, forming a sequence that is costly to forge and easy to verify. The public registry provides order and auditability: anyone can check that a title was created from valid work, that it is unique, and that subsequent transfers are properly signed. The design aims to minimize reliance on trusted third parties by using time-stamping, digital signatures, and replicated logs, while recognizing that some coordination is still needed to prevent conflicting histories. For a security engineer, two points stand out. First, bit gold shifts the trust boundary from an issuer to verification rules that anyone can execute: proof-of-work costliness, timestamped publication, and signed ownership updates. Second, it surfaces the open problems that Bitcoin later addresses with a global consensus process: resistance to Sybil attacks, convergence on a single history without a central mint, and incentives for participants who maintain the registry. In this sense, bit gold is a near-miss prototype: it assembles most of the necessary

components for decentralized digital money and frames the remaining consensus problem explicitly. [23]

3.7 2008 - Bitcoin

Bitcoin was proposed as a method for transferring value on the Internet without delegating trust to a central intermediary. The mechanism relies on well-understood building blocks: users sign transactions with public-key cryptography to demonstrate control of funds; a peer-to-peer network propagates these signed messages; and miners construct a chain of proof-of-work blocks that establishes a public ordering of events. In practice, nodes validate incoming transactions, assemble a candidate block, and then iterate over the block header by varying a nonce until the resulting hash falls below the current difficulty target. This search is computationally costly, whereas verification by other nodes is fast. When a valid block is found, it is broadcast to the network; peers verify both the proof of work and each transaction before extending the same chain. The system's first block (the "genesis block") was created on 3 January 2009. From there, security emerges from incentives and verification rather than institutional authority: miners are rewarded for following the consensus rules, and any honest participant can check the entire history independently. [18]

3.8 2015 - Ethereum

Ethereum generalizes the idea of programmable money into a programmable state machine. Instead of a narrow script language tied to specific transaction templates, it offers a virtual machine that executes small programs called smart contracts. The network maintains a single global state of accounts and contract storage. Users submit transactions that either transfer ether or call a contract; contracts can in turn call other contracts, update storage, and emit logs. Execution is deterministic and metered by gas: each operation has a cost, the sender specifies a gas limit and price, and the transaction halts if it runs out of gas. This mechanism prevents unbounded computation and ties resource use to an explicit fee market. The account model replaces Bitcoin's UTXOs with two account types—externally owned accounts controlled by private keys and contract accounts controlled by bytecode—making it straightforward to express stateful applications such as token ledgers, automated market makers, auctions, or access-control registries.

The system launched on 30 July 2015 and initially used proof of work to secure the chain; over time, clients, tooling, and development patterns converged around contract interfaces and events that make applications easier to compose. From a

security perspective, Ethereum trades simplicity for expressiveness: contracts are powerful but introduce new classes of bugs, so safe deployment depends on careful design, restricted interfaces, and economic limits enforced by gas. Conceptually, Ethereum reframes the blockchain as a general-purpose execution environment where agreement is not only on the order of transactions but also on the state transitions produced by running contract code. This is the step that turns blockchains from a payments network into a platform for decentralized applications. [24]

Chapter 4

Blockchain Technology

The core ideas behind blockchain technology emerged in the late 1980s and early 1990s. In 1989, Leslie Lamport developed the Paxos protocol, and in 1990 submitted the paper The Part- Time Parliament [2] to ACM Transactions on Computer Systems; the paper was finally published in a 1998 issue. The paper describes a consensus model for reaching agreement on a result in a network of computers where the computers or network itself may be unreliable. In 1991, a signed chain of information was used as an electronic ledger for digitally signing documents in a way that could easily show none of the signed documents in the collection had been changed [3]. These concepts were combined and applied to electronic cash in 2008 and described in the paper, Bitcoin: A Peer to Peer Electronic Cash System [4], which was published pseudonymously by Satoshi Nakamoto, and then later in 2009 with the establishment of the Bitcoin cryptocurrency blockchain network. Nakamoto's paper contained the blueprint that most modern cryptocurrency schemes follow (although with variations and modifications). Bitcoin was just the first of many blockchain applications.

4.1 Blockchain and Distributed Ledgers: Architecture and Security

Blockchain technology, despite being increasingly widespread and used in different sectors, needs a clear and precise definition. According to the National Institute of Standards and Technology (NIST), blockchains are defined as "tamper-proof and alteration-resistant digital ledgers, implemented in a distributed way (that is, without a central repository) and usually without a central authority." [25]. This definition allows us to isolate three different concepts. The first concept is resistance to tampering, which highlights a key property of blockchains: immutability, guaranteed by the use of cryptography, for example hash functions and the

digital signature. The second concept is distributed implementation, meaning a peer-to-peer architecture as opposed to the classic client–server models. Finally, the concept of the absence of a central authority in whom all trust is placed. Trust is instead transferred to a shared protocol. While a traditional centralized database entrusts custody and the recording of transactions to a single entity, such as a bank, blockchains introduce a radically different paradigm. The NIST describes it as a system that allows a community of users to record transactions in a shared ledger [25]. This statement provides two points for reflection. The first is the transfer of the responsibility for keeping a ledger from a single authority to a community. Responsibility is shared among all participants, so trust shifts from a single authority to the network itself. The second is that in a traditional system those who run the system have the power to confirm, cancel, modify, or delete a transaction. In a blockchain this does not happen, because transactions are recorded when they are published, making the ledger a permanent and verifiable history at any time. This shift from a centralized system to a distributed community represents a break with established practices.

To understand its limitations and characteristics, it is first necessary to analyze the model used for comparison. According to the NIST, in the centralized model "ledgers are owned and managed by a centralized trusted third party (i.e., the ledger owner) on behalf of a community of users" [25]. This architecture, whether implemented on a single server or on a coordinated cluster, essentially concentrates authority and control. It creates a system of delegated trust, where users must rely on the integrity, competence, and security of a single entity—whether a bank, a company, or a government. The centralized owner becomes a single point of failure, not only for security, but also for the availability and integrity of the ledger itself. The trust required by the centralized system gives rise to vulnerabilities. The NIST framework systematically contrasts these weaknesses with the resilient design of blockchain. The first critical vulnerability concerns data resilience and availability. A centralized ledger represents a single point of failure, since it "can be lost or destroyed; a user must trust that the owner is correctly backing up the system" [25]. This issue is removed in blockchain through distributed redundancy. A key advantage is that "each user can maintain their own copy of the ledger," making its loss or destruction much harder. Moreover, centralized systems often operate on homogeneous networks, where uniform software and hardware can make the entire system vulnerable to a single successful attack. In contrast, a "blockchain network is heterogeneous: software, hardware, and network infrastructure differ across nodes. Therefore, an attack on one node does not guarantee that it will work on the others" [25]. Resilience also extends to geographical network threats. In fact, a blockchain network "can consist of nodes distributed geographically around the world. Thanks to this and to peer-to-peer functioning, the network is resilient

to the loss of a node or even an entire region" [25].

While these architectural advantages ensure the persistence and availability of the ledger, the true revolution of blockchain lies in its cryptographic mechanisms, in addition to ensuring the availability of the ledger, one of the most significant features of blockchain is its ability to establish a reliable and immutable chronology without relying on a central authority. In the centralized model, the validity and completeness of the ledger ultimately depend on trust. "Transactions in a centralized ledger may not be transparent or valid; a user must trust that the owner is validating each incoming transaction" [25]. Similarly, "the list of transactions in a centralized ledger may not be complete; a user must trust that the owner is including all valid transactions received" [25]. Blockchain replaces this reliance on a single entity with reliance on a verifiable process. Its network "ensures that all transactions are valid; if a malicious node were to broadcast invalid transactions, other nodes would detect and ignore them, preventing their propagation" [25]. Moreover, for a new block to be accepted by the network, it must cryptographically reference the previous one, thereby forming a tamper-evident chain in which any attempt to exclude valid transactions or alter the historical record would be immediately rejected by honest nodes.

The centralized model presents a concentrated and attractive target for attackers. "A centralized system may be insecure; a user must trust that the systems and networks are receiving critical security updates and following best practices. The system could be compromised and personal information stolen" [25]. This creates a risk of a single breach in which a vast repository of sensitive data may be exfiltrated. By contrast, a blockchain network, due to its distributed nature, "does not present central points of attack." To compromise network participants, "an adversary would need to target them individually; attacking the blockchain itself would encounter resistance from the honest nodes" [25]. This architecture also localizes risk: if "a single node were not updated, the impact would be limited to that node, not the entire system" [25]. Security, is a matter of the collective strength and honesty of a global, redundant network.

4.2 Blocks and Linking in Blockchain

As mentioned before, in blockchain there is no central authority, since everything is managed through a shared and permanent record. The system works thanks to the way blocks are formed and connected using cryptographic methods.The process starts when someone sends a transaction to the network. This can be done through a client app, like a browser extension or a digital wallet. Once it's sent, the transaction is distributed over the peer-to-peer network, but at this stage it's

not yet written into the blockchain. In most systems, transactions are kept waiting in a temporary area called mempool until a trusted or authorized node picks them up and puts them inside a new block. A block is basically a data unit with two main sections [25]:

Block Header: it holds the cryptographic and administrative details of the block.

Block Data: this part stores a list of confirmed and valid transactions.

Before being added to a block, every transaction has to be checked. The network makes sure the format is correct and that all the cryptographic signatures are valid. This proves that the sender truly has the right to use those digital assets [25]. This step is essential for keeping the ledger honest and reliable, since "the network ensures that all transactions are valid; if a malicious node spreads invalid ones, other nodes detect and ignore them, stopping their propagation"[25]. Even if the way it's implemented can differ, the general block structure usually follows this pattern[25]:

Block Header:

- Block Number: shows the block's position in the chain.
- Previous Block Hash: connects it to the block before.
- Merkle Root Hash: a compact and secure summary of all the transactions in the block.
- Timestamp: shows when the block was created.
- Nonce: a number used in Proof-of-Work systems to solve a specific computational challenge.

Block Data:

- The verified list of transactions and sometimes other registry information.

The fact that the ledger cannot be changed depends on how the blocks are connected to each other. Every block includes the cryptographic hash of the previous one, and this creates the actual chain [25]. This simple structure has an important consequence. If someone tries to change a transaction that is already inside a block, the hash of that block would change immediately. Since the next block keeps the original hash, the link between them would instantly break. To hide the modification, the attacker would have to recalculate all the hashes of the following blocks. Doing that would need an enormous amount of computing

power[25]. Because of this design, “any altered blocks can be easily detected and rejected” [25], keeping the whole history of transactions reliable and protected from manipulation[25].

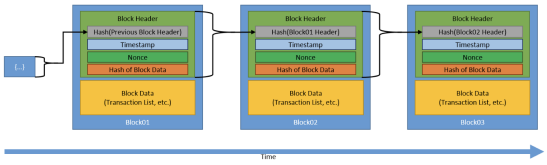

Figure 4.1: Generic Chain of Blocks

4.3 Types of Blockchain Architectures

To proceed with the study of blockchain technology, it’s important to define the two main authorization models, which determine who can add new blocks and who can check the transactions.
The first method is the permissionless model. The main characteristic of this model is that it is open to anyone who wants to participate so without the need to be approved by a central authority. This type of blockchain network represents in the best way the concept of decentralization and is used in some cryptocurrencies such as Bitcoin. Any user in the network can read the chain, send transactions, and try to publish blocks. This feature, however, makes the network more vulnerable to possible adversary attacks. To reduce this risk and to guarantee security without the need for pre-existing trust between participants, permissionless networks use consensus protocols such as Proof of Work (PoW) or Proof of Stake (PoS). These mechanisms require validators to face a computational or economic cost, making it economically disadvantageous and difficult to attack the network. Correct behavior, on the other hand, is rewarded with an incentive (for example in cryptocurrency) for publishing blocks that follow the protocol.
On the other hand, in permissioned networks, the ability to publish blocks is reserved for a predefined and authorized group of nodes. To become part of the network, a user must be identified and invited. Once inside, not all users have the same privileges. Some can only read transactions, while others can also write and add them. This system is mainly used by groups of firms that want to collaborate. The certified identity of participants is the foundation of the permissioned model. This makes it possible to create a clear audit trail and also to implement an advanced governance model, where reading and writing rights on the distributed ledger are assigned according to each member’s role and level of trust inside the consortium. A permissioned blockchain works as a way for participants to coordinate without having to fully trust each other, while still keeping control over who can view or

change their own sensitive data. The permissioned model has two main limitations: the first one is lower system resilience, caused by the dependence on a limited number of authorized nodes; the second is the presence of a central point of trust (the so-called "trust anchor") in the entity that manages access, which goes against the ideal of being fully "trustless." However, for enterprise use, these disadvantages become essential functional compromises[25].

4.4 Consensus Mechanisms

One of the central problems a blockchain must solve is deciding who gets to add the next block. Consensus mechanisms are the rules that help different participants reach the same view of the ledger, even if they do not fully trust one another. In permissionless networks many nodes may try to publish a block at the same time, and they usually do so because there is an economic reward at stake, such as newly created coins or transaction fees. Since nodes are often motivated by profit rather than by the network's health, the consensus rules must be designed so that honest behaviour is more attractive than cheating. Every blockchain begins from a single agreed starting point, the genesis block. From there, new blocks are added according to the chosen consensus model, and each block links to the one before it by including the previous header's hash. This chaining plus the ability of any node to check each block independently means that all participants can reach the same view of the current ledger without trusting a third party. When two versions of the chain exist at the same time, most systems use a clear rule—often the chain with the most work or the longest valid history—to decide which chain to accept. Temporary disagreements are allowed but must be resolved so the network converges back to a single history. Proof of Work (PoW) is a common method for choosing who will publish the next block. In PoW a node must solve a hard computational challenge; the solution is easy for others to check, but expensive to find. A typical challenge in PoW makes it costly to create many identities and try to control the network, since influence is tied to computing power and not merely to the number of addresses. Miners sometimes join pools to share work and split rewards, splitting the task of scanning for the correct nonce. The downside of PoW is its hard use of resources such as time, electricity and hardware. That cost motivates miners to locate where power is cheapest and can raise environmental and centralization concerns. Proof of Stake (PoS) uses a different method. Instead of buying influence with computing machines, a participant's chance to propose a block depends on how much cryptocurrency they have locked as stake. The more stake someone has, the more reason they have to keep the system healthy, because misbehaviour could harm their own holdings. PoS avoids the need for expensive challenge-solving and therefore uses much less energy. There are different

ways to implement PoS. In a simple stake-weighted selection, nodes are chosen randomly but with probability proportional to their stake. In voting-based PoS, several stakers propose blocks and then staked participants vote in rounds to pick one block. Coin-age systems add a waiting period so coins must age before they count toward selection, which prevents very large holders from always dominating. Delegate systems let users vote for a smaller set of trusted block producers who then create blocks on behalf of the voters. One challenge in PoS is the "nothing at stake" issue: when multiple competing chains exist, a staker can support all branches at essentially no cost, which can slow convergence. Protocol designers address this with economic penalties, slashing and other rules that make equivocation costly. Proof of Authority, also called Proof of Identity, is used in permissioned networks where validators are known and have some level of trust in this way, validators stake their reputations or verified identities rather than hardware or coins. Because misbehaviour directly harms a validator's real-world standing, these systems can rely on lighter-weight consensus protocols and do not need the same resource intensity as PoW. They fit well where legal or organizational remedies exist to punish dishonest validators. Each consensus model balances different trade-offs: energy and cost, resistance to various attacks, centralisation risks, and the assumptions about how much trust exists between participants[25].

4.5 Smart Contracts

The idea of a smart contract goes back to 1994, when Nick Szabo described it as a computerized protocol that carries out the terms of an agreement. In general, smart contracts are meant to meet usual contract needs — for example payment rules, confidentiality, and enforcement — while reducing both accidental and intentional mistakes and lowering the need for trusted middlemen. A smart contract is basically a set of code and data (often called functions and state) that is placed on a blockchain by a signed transaction. Nodes in the network run the contract, and every node that executes it should get the same result. The outcome of the execution is then recorded on the blockchain so it becomes part of the shared record. Users interact with smart contracts by sending transactions to the contract's public functions. The contract runs the requested function using the data the user provides and delivers the agreed service. Because the contract code is stored on the blockchain, it is hard to alter silently. This makes smart contracts useful as a kind of trusted third party in some situations. A contract can do calculations, save data, show public state, and even move funds automatically. Smart contracts are not only for money; they can be used for many kinds of automated tasks. Not every blockchain supports smart contracts, though. Smart contracts often model multi-party transactions, so they are useful in business processes that involve several

organizations. In those cases a contract can give visible and verifiable data, which helps build trust, speed up decisions, cut reconciliation costs, and shorten the time needed to finish transactions. It is important that smart contracts are deterministic. Given the same input, they must always give the same output. Also every node that runs the contract must agree on the new state after execution. For this reason a contract cannot directly fetch external web data during its run. If it needs outside information, that information must be provided as input. When a contract uses data from outside the blockchain, the service that provides that data is usually called an oracle, and this raises its own design challenges. In many blockchains, the nodes that publish new blocks also run the smart contract code at the same time. Some systems separate these roles: some nodes execute contracts while others only check the results. In public smart contract platforms like Ethereum, users pay for contract execution. The system limits how much computing a contract call can do; if the limit is exceeded, the execution stops and the transaction fails. This payment and limit system rewards nodes that do the work and prevents attacks that try to use up all resources, for example by running infinite loops. Permissioned blockchains that allow smart contracts, such as platforms using Fabric chaincode, often work differently. Because participants are known, these systems may not charge users for execution. They can use other ways to prevent bad actions, like revoking access from misbehaving members[25].

4.6 Security Limitations and Risk Management in Blockchain

Using blockchain does not remove normal cybersecurity risks and many of these dangers come from people and their choices, so a solid cybersecurity program is still necessary. As attackers learn more about blockchain systems and their weaknesses, organisations must plan and act to reduce those risks. Existing cybersecurity frameworks and standards are still useful for systems that use or connect to blockchains; they may need small adjustments to fit blockchain specifics, but they give a good basis for protection. For example, the NIST Cybersecurity Framework does not force a single method; it helps organisations choose practices that match their threats, vulnerabilities and risk tolerance, and it can be applied to blockchain environments as well. Blockchains are often called tamper resistant once transactions are recorded in a published block, but transactions that are not yet included in a block can still be attacked. Networks that rely on timestamps may be exposed if an attacker spoofs time or changes a node's clock, making time itself an attack vector, and denial of service attacks can target the platform or the smart contracts running on it. Attackers can scan the network for weak points and use zero-day exploits when they find them. In the rush to launch blockchain

services, new code such as smart contracts may contain bugs or insecure setups that attackers can exploit in the same way they attack web applications today. A blockchain enforces its transaction rules, but it cannot force users to behave well. This is especially a problem in permissionless networks where users are pseudonymous and identities are not tied to real people. These systems often use rewards to encourage honest actions, but some users may act badly if the expected gain is higher than the risk. To cause real damage, attackers usually need a lot of control, for example a large stake or lots of processing power; if a coalition gains enough influence, they can ignore transactions from certain users or regions, build a secret alternate chain and reveal it when it becomes longer than the public one, or refuse to share published blocks and so disrupt information flow. Networks can respond with measures such as hard forks, but whether losses are reversed depends on developer and community choices. It is also possible for insiders in permissioned networks to act maliciously: an administrator might, depending on system design, take over block production, block some users, alter history, double spend, delete data, or reroute connections. Saying a blockchain has no trusted third party does not mean it needs no trust at all. There is trust in the strength and correct use of cryptographic algorithms, trust that smart contracts work as intended and have no hidden bugs, trust in the developers who build and update the software, trust that most users are not secretly colluding since a group controlling over 50% of block creation power could subvert the chain, and for users who do not run a full node there is trust that the nodes they rely on accept and handle transactions fairly. In short, blockchain shifts some forms of trust away from a central authority, but it does not eliminate trust altogether, and risk management must recognise these assumptions and protect against failures in each area[25].

Chapter 5

Hyperledger Fabric

5.1 Introduction to Hyperledger Fabric

In the previous chapters, we studied how blockchain technology works and its development from early applications like Bitcoin and Ethereum. These public networks showed the big potential of distributed ledgers, but they also have limits when used by companies. Problems like anonymity, slow transaction speed, and no data privacy make them difficult for many business needs where participants must be known and operations must stay secret.

To solve these issues, the Linux Foundation created Hyperledger Fabric. This is an open-source platform designed especially for business use. Fabric uses the main ideas of blockchain, which are transparency, immutability, and decentralization. However, it uses a permissioned model where all participants are known and verified. This makes it possible to build trust between companies that need to work together without sharing sensitive information publicly.

A key difference from public blockchains is that Fabric features a modular architecture. This approach ensures that every essential component, including the consensus mechanism, identity management, and data storage, can be configured on its own. This flexibility helps to make performance and security better based on the specific needs of each project.

Another fundamental aspect involves the adoption of smart contracts, which Fabric calls chaincode. These define the rules of the business application. Unlike other platforms that use specific programming languages, Fabric lets developers write smart contracts in common languages such as Java, Go, or Node.js. This makes adoption easier because most companies already have staff who know these

technologies.

Fabric is also different because it does not use a native cryptocurrency or a mining process. This removes unnecessary costs for computing and security risks, making the system more efficient and simpler to manage. Consensus can be reached using different pluggable mechanisms chosen according to how much trust and decentralization the network needs.

Finally, Fabric offers advanced functions for privacy and confidentiality. Using channels and private data collections, participants can share information only with authorized parties, while the rest of the ledger stays secure and consistent. As a result, this platform works well for sectors like finance, healthcare, and logistics, because keeping data secure is the main goal.

5.2 Fabric Architecture Overview

Having introduced the motivations for a permissioned approach and the main features of Hyperledger Fabric (modularity, identity management, chaincode and privacy primitives), we now move from theory to structure. This short section explains how those ideas are implemented in a running Fabric network: how organisations, channels, peers, the ordering service, CAs/MSPs and chaincode lifecycle fit together. The goal is to show how the platform's modular components realize control, confidentiality and consensus in practice, so the reader can link the design choices previously described to their concrete technical roles. A Fabric network is built around the idea that a group of known organisations cooperate on shared business processes while keeping control of their own systems and data. To make this possible, the platform defines logical spaces called channels. A channel is a private ledger shared only by its members; it contains its own configuration, its own policies and its own history of transactions. Because channels are independent, the same organisation can be a member of several channels at the same time. This lets a single participant join different business workflows without exposing data from one workflow to the others. Peers are the physical hosts that implement most of the network's work. Each peer stores a copy of the ledger for the channels it has joined and runs the chaincode that implements the business logic. When a client wants to change the ledger it sends a transaction proposal to peers; some peers execute the chaincode, check the result and sign it as an endorsement. Later, after the transactions are ordered and packaged into blocks, peers validate these blocks and update their local ledger. In short, peers execute smart contracts and store the channel's data. Keeping all peers in agreement requires a component that decides the global order of transactions. This is the ordering service. The ordering service

receives endorsed transactions, arranges them in a single, deterministic order and groups them into blocks. In practice, peers run the chaincode that implements business logic and keep a local copy of the channel ledger. In production, the ordering service usually runs on several nodes to increase reliability. Conceptually, it is the element that makes the ledger consistent across organisations. Identity and trust are enforced using standard certificates. Each organisation typically operates a Certificate Authority that issues X.509 certificates to its peers, orderers, administrators and applications. The Membership Service Provider links these certificates to an organisation and defines which identities are trusted. Because channel policies reference MSP identities, the combination of CAs and MSPs is central to governance: they determine who can join the network, who can approve changes and who can sign transactions. Chaincode is Fabric's term for the packaged business logic—smart contracts that define how assets move and how rules are enforced. The chaincode lifecycle is deliberate: it must be installed on peers, its definition must be approved by the required organisations, and it must be committed to the channel. An essential part of the chaincode definition is the endorsement policy. This policy states which organisations must endorse a transaction before peers will accept it. By choosing different endorsement rules, a consortium can express its trust model in precise terms: for example, it can require a majority of members, or signatures from specific parties, depending on the business need. Client applications are the entry points for users and external systems. Modern Fabric setups often use the Fabric Gateway, a gRPC-based front end that simplifies how applications submit proposals and gather endorsements. The gateway hides many low-level details of the network, so application code can remain small and focused on business logic. Clients use the gateway to propose transactions, receive endorsement responses, and submit the final transaction for ordering. Finally, Fabric is designed to evolve. Channels are created from a configuration block that lists member organisations and policies, and this configuration can be updated later. Adding a new organisation, changing policies, or deploying new chaincode all involve producing a new configuration block and applying it to the channel. Because these changes are recorded on-chain and controlled by declarative policies, network evolution is orderly and auditable. This flexibility—combined with clear separation of responsibility—makes Fabric suitable for real-world consortia where participants, rules and needs can change over time.

5.3 Prerequisites and Environment Setup

The experimental environment mirrors the production–style choices adopted for this thesis while remaining intentionally small. The Fabric network runs on a macOS host using Docker containers; the application stack (React frontend, Node.js

backend, SQLite) runs on a separate corporate machine. The client connects over gRPC with TLS and is configured with `asLocalhost=false` because the hosts are distinct. The host must satisfy the official prerequisites for running a Docker–based Fabric test network; the project documentation is the authoritative source for OS–specific notes and compatibility constraints [26].

The network topology derives from the official *fabric-samples* but is adapted to the document–integrity scenario. A single channel is used; two application organisations, Org1 and Org2, join the channel and operate peers (`peer0.org1`, `peer1.org1`, `peer0.org2`); the ordering service is hosted by a separate organisation (Org0) and runs as a single node in this setup. The "Using the Fabric test network" guide and the samples repository provide the scripted commands to start the network, create the channel, join peers, and deploy chaincode; these materials are the recommended entry point and are aligned with Fabric releases [27].

Peers are configured to use CouchDB as the state database in order to support value–based queries over JSON documents that describe metadata for files and tasks. Fabric officially supports LevelDB (embedded) and CouchDB (external); the latter enables rich queries at the cost of operating an external service. For an enterprise–style PoC in which document metadata and task relations benefit from ad–hoc filters, CouchDB is appropriate and consistent with the documented trade–offs [28].

Client integration uses the Fabric Gateway programming model from a Node.js backend. The Gateway, introduced in Fabric v2.4 and recommended in v2.5, allows the peer–side gateway service to gather endorsements on behalf of the client, reducing client–side complexity. In this thesis the backend loads identities from a wallet, connects with TLS using the connection profile, and submits transactions via the Gateway API; this path is specified by the official overview and the Node.js reference [29].

The chaincode lifecycle follows the v2.x process and reflects the governance choices made for this network. The smart contract that records minimal document evidence (identifier, SHA–256 hash, uploader, timestamp) is packaged and installed on Org1 and Org2 peers; each organisation approves the chaincode definition, which includes the version and an endorsement policy that requires both Org1 and Org2. Once the required approvals are present, the definition is committed to the channel. Operational checks ensure readiness before commit, enforcing explicit multi–party consent to the business logic [30].

In practice, setup proceeds as follows and remains reproducible across hosts. First,

install the official prerequisites on the macOS ledger host and prepare Docker Desktop with TLS–enabled peer and orderer endpoints [26]. Second, clone *fabric-samples*, adapt the test network to include the listed peers, start the containers, create the channel, and join Org1 and Org2 [27]. Third, enable CouchDB companions for peers to act as the external state database [28]. Fourth, package, approve, and commit the document–metadata chaincode with the endorsement policy that requires Org1 and Org2 [30]. Finally, configure the backend on the corporate machine with a wallet that holds application identities, a connection profile that references TLS CAs, and `asLocalhost=false`; the backend then evaluates and submits transactions through the Fabric Gateway [29].

5.4 Starting the Test Network

The network is brought online in a controlled sequence so that ordering and peer components start cleanly, a channel is created, and each organisation joins with validated settings. On the macOS host, Docker containers launch the ordering service for Org0 and the peers for Org1 and Org2 with TLS enabled; the client stack runs on a separate corporate machine and connects over gRPC with `asLocalhost=false`. Once containers are up, a channel is created and peers from both organisations join the channel; the environment is considered ready when basic reachability and ledger checks succeed for every peer.

The channel spans two application organisations and one ordering organisation. Each application organisation operates two peers. Org1 runs `peer0.org1.example.com` (port 7051) and `peer1.org1.example.com` (port 18051); Org2 runs `peer0.org2.example.com` (port 9051) and `peer1.org2.example.com` (port 19051). Each peer uses a dedicated CouchDB state database. The ordering service is provided by Org0 as a single node in this proof of concept. Peers belong to the application organisations and endorse/commit transactions; ordering nodes belong to Org0 and sequence transactions into blocks. Org0 does not operate peers.

Anchor peers are configured per organisation to support inter-organisation gossip. In this setup, Org1 advertises `peer0.org1.example.com:7051` as anchor, while Org2 advertises `peer0.org2.example.com:9051`. Anchor peers are standard endorsing/committing peers; they have no extra privileges, but provide the public contact points through which membership and blocks propagate across organisations.

Two concise forms of evidence accompany this section. First, a Docker overview confirms that the expected containers are running on the ledger host: the ordering

service (Org0), the four peers (two for Org1 and two for Org2), their CouchDB companions, and the certificate authorities; chaincode runtime containers (`dev-*`) are also visible (Figure 5.1). Second, a minimal CLI extract shows for each peer that it has joined `mychannel` and reports the same height and current block hash, demonstrating ledger synchronisation before chaincode deployment. The four extracts are obtained with `peer channel list` and `peer channel getinfo -c mychannel` on `peer0.org1.example.com`, `peer1.org1.example.com`, `peer0.org2.example.com` and `peer1.org2.example.com`; identical height/hash across all four peers confirms that the network is ready to proceed.

Figure 5.1 shows the running containers on the ledger host. The caption explicitly distinguishes the ordering node from the peers and notes the presence of CouchDB companions and chaincode runtime containers, to avoid ambiguity between peer roles and the ordering role in Org0.

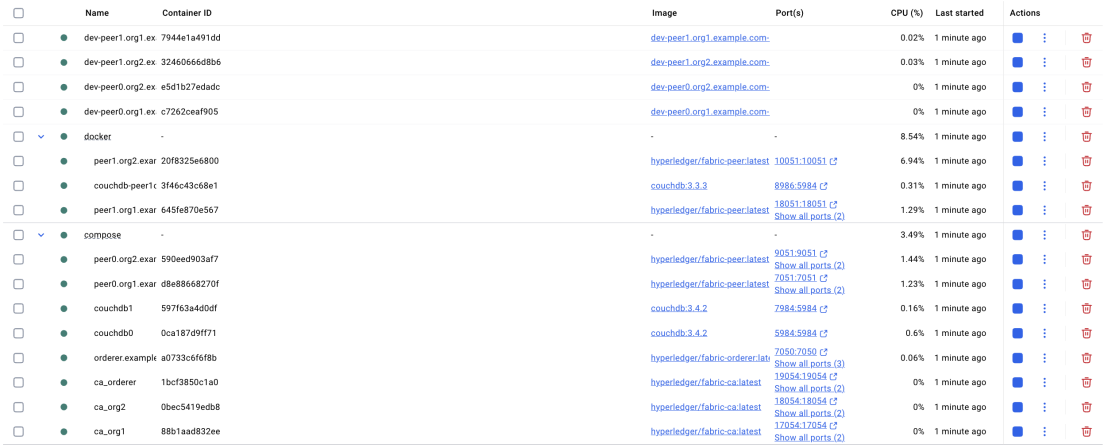

Figure 5.1: Active containers on the ledger host: ordering service (Org0, single node), two peers for Org1 (`peer0.org1`, `peer1.org1`) and two peers for Org2 (`peer0.org2`, `peer1.org2`) with their CouchDB companions, and the certificate authorities. Chaincode runtime containers (dev-*) are also visible.

5.5 Channel Creation and Configuration

In Hyperledger Fabric, a channel is the place where collaboration actually happens. It defines which organizations are allowed to see which data, which peers maintain the ledger, and which policies govern how decisions are made. In the context of this work, channels are not treated as a purely abstract construct, but as the mechanism that turns a generic Fabric deployment into a controlled environment suitable for

managing shared documentation in large industrial projects.

After the test network was started, a dedicated application channel was created to connect the organizations involved in the Proof of Concept. In the PoC scenario, these organizations represent different stakeholders in a construction project (for example, the company coordinating the project and a contractor responsible for specific activities). The goal was to reproduce a realistic setting in which parties need to share operational information and project documents, while keeping strict control over who can read, write, and validate data on the ledger.

The channel was first defined using the standard Fabric configuration tools. A channel creation transaction was generated from the network configuration profile and submitted to the ordering service, which produced the genesis block for the new channel. This block establishes the initial rules of the environment: which organizations are members, which ordering service they rely on, and which policies must be satisfied to update configurations in the future. Once the channel was created, peers from each participating organization retrieved the genesis block and joined the channel. From that point on, they started to receive the same ordered sequence of blocks and to maintain identical copies of the channel ledger.

To enable effective communication between organizations, one or more peers were designated as anchor peers for each member organization. This step is often perceived as a minor configuration detail, but in practice it is what allows peers belonging to different companies to discover each other and exchange gossip traffic. In the PoC, defining anchor peers ensured that information about new blocks, transactions, and chaincode definitions could flow reliably across organizational boundaries, without exposing the channel to actors that were not explicitly admitted.

The channel configuration was then refined to align with the security and governance requirements emerging from the PoC use case. Endorsement policies were defined so that the validation of document-related transactions required signatures from peers of specific organizations, reflecting the idea that certain operations (such as registering a new project document or confirming a task completion) must be collectively attested. Access control lists and role mappings were aligned with the application-level roles of the platform (administrator, standard user, reviewer), so that the technical permissions enforced by Fabric matched the logical responsibilities visible to users in the front-end.

This configuration work has two direct effects on the overall solution. First, it enforces confidentiality by limiting visibility of documents and transactions to

the organizations that are effectively part of the project consortium. Second, it strengthens accountability: every significant action on project artefacts (uploads, updates, approvals, compliance checks) is recorded on a shared ledger that no single party can silently alter. In other words, the channel becomes the technical translation of the trust model required in multi-stakeholder construction projects, and it provides the foundation on which the document management smart contracts described in Chapter 6 operate. [31]

5.6 Chaincode Lifecycle

Within Hyperledger Fabric, the chaincode acts as the real engine of interaction between the network's infrastructure and the business logic it supports. It defines how data is created, changed, and validated, and it ensures that every transaction follows the same set of rules across all peers. In other words, the chaincode translates cooperation between independent organizations into a set of technical operations that everyone can verify. Understanding how this mechanism is deployed and managed is crucial to see how Fabric guarantees both trust and control in shared environments.

The most recent releases of Fabric introduced a redesigned lifecycle model intended to make the management of smart contracts more transparent. The idea behind it is quite simple: on a shared ledger, no single organization should ever be able to install or update logic on its own. For that reason, the process of deploying chaincode is structured in a way that explicitly records who approved what and when.

At the beginning of the process, the source code of the smart contract is packaged together with a short metadata file that defines its name, version, and language. This package can be distributed among all the organizations involved, so that each of them works with exactly the same code. Once this has been done, the organizations install the package on their peers. At this point, nothing has changed on the channel yet, but the peers are technically ready to execute the logic once it is formally activated.

After installation comes the approval stage. Each organization reviews a chaincode definition that describes how the contract will behave on a given channel including its version, endorsement policy, and sequence number. The definition must be approved by all relevant parties before it can be committed. This step is not just a formality: it represents a collective decision about the rules that will govern shared transactions. Only when the required number of approvals is reached can

the definition be committed to the channel, making the chaincode officially active. From then on, peers that belong to approved organizations can execute and endorse transactions according to that shared logic.

When updates are needed, the same procedure must be repeated. Whether the change involves a new feature, a policy modification, or a simple bug fix, the updated version must go through another round of approvals and be committed again. Every change increases the sequence number of the chaincode, creating a traceable version history. This mechanism prevents hidden alterations and ensures that all participants remain aware of how the logic evolves over time.

This lifecycle may appear procedural, but in practice it defines how collaboration happens within a Fabric network. It transforms the act of deploying business logic into a visible governance process. Each definition, approval, and update leaves a record that shows which organizations participated and agreed on the rules now embedded in the ledger. This traceability is particularly valuable in domains that demand accountability, such as compliance-driven industries or large multi-party projects.

The following chapter will demonstrate how this governance structure becomes concrete in a real-world scenario. The same lifecycle was applied to deploy a customized smart contract designed for document registration and verification within complex construction projects. There, the abstract principles of Fabric's lifecycle are translated into practical features that promote transparency, coordination, and trust among independent stakeholders.

5.7 Endorsement Policy

In Hyperledger Fabric, a transaction isn't simply accepted into the ledger once it's been ordered. That's only part of the story. Before a transaction can be considered valid and officially recorded, it must pass a crucial checkpoint: explicit approval from a designated group of organizations, according to what's known as the endorsement policy. This policy-centric model marks one of the fundamental distinctions between Fabric and more traditional public blockchains. While the latter rely on consensus protocols driven by anonymous nodes, Fabric takes a different route grounding its validation process in provable agreements among identified participants. Trust here isn't an abstract principle; it's codified, enforced, and cryptographically verifiable.

When a chaincode definition is committed to a Fabric channel, the endorsement policy becomes a centerpiece of that definition. It spells out which organizations must give their nod before any transaction result gets etched into the ledger. The way it works is both elegant and rigorous: each involved peer runs the same chaincode function, evaluates the input, and produces what's called an endorsement—a signed and auditable response. The client gathers these endorsements and hands the transaction off to the ordering service. Once the resulting block circulates among the peers, they collectively verify two things: that the endorsements meet the policy's criteria and that the results match across peers. If even a single required signature is absent, or if the outcomes diverge, the transaction is flagged invalid. No state update occurs. It's as if it never happened.

This entire process exists for a reason. It's designed to ensure that all consortium members genuinely agree on the validity of each change. The endorsement policy essentially acts as a programmable layer of governance. In real-world terms, it lets the network enforce existing organizational structures. Picture a construction project: perhaps no update to the project's records is accepted unless both the contractor and the client sign off. Or consider a financial consortium that requires at least two independent members to approve any disbursement or audit entry. The flexibility of these policies is intentional, they can be tailored to reflect real institutional protocols, whether strict or permissive.

But this is more than just a coordination mechanism. It's also a bulwark against unilateral tampering. By requiring multiple verifiable endorsements, Fabric prevents any single party from slipping unauthorized data into the system. Each transaction is not only endorsed but time-stamped and attributable, creating an immutable audit trail. In environments where multiple stakeholders share infrastructure but need to preserve operational autonomy, this feature becomes indispensable. If disputes arise, the endorsements themselves serve as a factual breadcrumb trail pointing to who did what—and when.

Ultimately, the endorsement policy embodies the collaborative spirit at the heart of Fabric's design. It bridges technical validation with institutional legitimacy, ensuring that what ends up on the ledger is both functionally correct and organizationally approved. Paired with the chaincode lifecycle, it forms a robust framework where logic and execution are subject to shared governance. This is how Fabric translates distributed trust from a theoretical idea into a tangible system—one where every transaction stands on a foundation of measurable, cross-party consensus.

5.8 Advanced Security and Privacy Mechanisms in Hyperledger Fabric

In addition to identity management and endorsement policies, Hyperledger Fabric incorporates mechanisms for data protection and trust assurance among participating organizations. These features are essential in environments where shared information is sensitive, such as multi-stakeholder industrial projects or regulated financial settings.

Identity Management and Access Control

The Membership Service Provider serves as the central component for identity management. Each peer, orderer, and client holds X.509 certificates issued by the Certificate Authorities of their respective organizations, providing strong authentication and accountability for all operations [32]. The MSP allows organizations to define fine-grained access rules for channels and chaincode, linking identities to governance policies without exposing confidential information.

Data Segmentation and Confidentiality

To protect sensitive information, Fabric uses channels and private data collections. Channels isolate transactions and ledgers among groups of organizations, while private data collections allow selective sharing of information with authorized participants, publishing only verifiable hashes to others [33]. This structure makes it possible to execute confidential transactions between selected parties without compromising the integrity of the global ledger.

Chaincode Execution Security

Sensitive data can also be protected during chaincode execution by using Trusted Execution Environments such as Intel SGX. In this configuration, data and chaincode remain encrypted while being processed, reducing the risk of exposure even if the peer host is compromised [34]. The integration of Trusted Execution Environments extends security beyond the network layer, covering the computation phase of transactions.

Auditability and Accountability

Every transaction that meets the endorsement policy produces an immutable record that is attributable to the endorsing organizations. Combined with private data collections and identity management, this model enables a complete reconstruction

of approvals and modifications, ensuring transparency and accountability. In regulated environments, this capability is essential for compliance verification or dispute resolution.

Summary

The combination of certified identities, data segmentation, and execution protection reinforces the trust model in Hyperledger Fabric. Transactions are verified, confidential, and secure throughout their lifecycle. For complex document management scenarios, these mechanisms ensure that information remains private, actions are fully traceable, and organizations can collaborate with confidence.

Chapter 6

Proof of Concept

6.1 Purpose and Scope of the Proof of Concept

This chapter presents a proof of concept (PoC) for a document management platform that utilizes Hyperledger Fabric to ensure integrity, traceability, and strict governance in multi-stakeholder industrial projects. The design adopts a hybrid on-chain and off-chain architecture: heavy documents and rich operational metadata are stored in a conventional repository (off-chain), while a tamper-proof "digital twin" of the document—consisting of its unique identifier, cryptographic hash, and the provenance of the submitting user—is anchored to the distributed ledger. By decoupling content storage from integrity verification, the system ensures that file authenticity can be mathematically proven at any time without compromising data privacy or ledger performance.

The scope is narrow and aligned with an enterprise-grade "Hosted Consortium" model. The application layer comprises a React frontend and a Node.js backend that implements a Dynamic Identity Management system. Unlike simple connectors that use a single administrative identity, the backend integrates a Fabric Wallet to manage distinct X.509 certificates for each user role (Admin, Standard User, and Reviewer). When interacting with the ledger, the backend acts as a transparent proxy, creating a secure gateway connection using the specific cryptographic material of the authenticated user. This ensures that every transaction is digitally signed by the actual operator, preserving non-repudiation. Off-chain persistence relies on SQLite for this proof of concept, which effectively maps operational data to the specific tenant contexts within the organizations.

The ledger runs on a test network designed to simulate a realistic industrial separation of duties. The network consists of two primary organizations: Org1

(Infrastructure & Construction Provider) and Org2 (Validation Authority). Org1 hosts the peers (peer0 and peer1) that serve multiple construction companies acting as logical tenants, while Org2 operates its own peers to provide independent validation. All peers participate in state replication and block dissemination. The endorsement policy is configured to require approval from both organizations (AND('Org1MSP', 'Org2MSP')), enforcing a digital "four-eyes principle" where no data can be notarized without the cryptographic consent of both the provider and the auditor. Peers use CouchDB as their state database, and ordering is provided by a separate ordering organization (Org0) running a single ordering node.

Operational assumptions are stated to support reproducibility. The Fabric network runs on a macOS host, and the application stack runs on a separate corporate machine. Connections to peers and orderers use gRPC over TLS with mutual authentication, and the client is configured with asLocalhost=false because the two hosts are distinct. This arrangement mirrors common constraints in corporate environments and makes the effect of network distance visible without changing the logic of the system.

The Proof of Concept is considered successful when the end-to-end workflow demonstrates both functional integrity and Zero Trust security. Specifically, the system must meet two criteria: first, when a user uploads a document, the smart contract must automatically derive the user's identity and privileges directly from their X.509 certificate attributes, rejecting any attempt to spoof identity via transaction arguments. Second, a verification query must recompute the hash of a stored file and compare it against the immutable ledger record, returning a definitive boolean outcome. Each successful write operation returns a transaction identifier that serves as a cryptographic receipt for auditability.

Two design notes are recorded at the start. First, the ledger state uses composite keys to support efficient querying by document hash. Second, the system implements Attribute-Based Access Control (ABAC) within the chaincode, ensuring that write permissions are granted only to users possessing specific role attributes (e.g., role=standard_user or role=admin) issued by their respective Certificate Authority. Performance metrics and latency observations regarding this secure architecture will be reported later in this chapter.

6.2 High level introduction to the platform

The platform is designed as a practical workspace where project teams organise activities, manage documents and verify integrity on a permissioned ledger without

breaking their daily routines. At the top it offers a web interface that is familiar to enterprise users. Under the surface, it implements the "Hosted Consortium" architecture described in the previous section, separating operational data from the blockchain anchor so that the application stays responsive while the ledger provides a reliable and cryptographically enforcing audit trail. The design is role aware from the start and strictly links application-level permissions to the underlying blockchain identities.

There are three primary roles mapped directly to X.509 certificate attributes. An Admin oversees the configuration of projects and teams, creates and disables accounts, assigns permissions and monitors the overall health of the workspace. A Reviewer focuses on the quality of deliverables and the state of tasks, checks that the right documents are present and verified, and records approvals or requests for changes (digitally signing these actions as a member of the Validation Authority). A Standard User concentrates on day to day work, uploads files when a task requires them, consults project information and follows the status of assigned activities. Each role sees a tailored navigation with the functions that matter most for that responsibility, ensuring that operational duties are strictly segregated according to the endorsement policies.

The interface is organised around projects and tasks. A project acts as the main container for planning and reporting, while tasks provide the level where actual work happens. A task may require one or more documents; when a file is uploaded the application computes a cryptographic hash, stores operational metadata off chain and writes a minimal reference on the ledger using the specific user's cryptographic context. The same task shows the current state of its documents, the people responsible and the history of actions, so that users do not have to switch tools to understand what remains to be done.

A built in model viewer allows teams to open and inspect IFC files directly in the browser. Users can orbit, pan and zoom through the model, isolate elements of interest, switch viewpoints and read the properties that come from the IFC schema. When needed, a document that belongs to a task can be linked to model elements so that technical evidence and geometry stay aligned. This keeps coordination efficient: a reviewer can open the model, check associated files and confirm that the submitted content matches the design intent without exporting data to external tools.

The administrator view provides a clear dashboard of projects, users and system status. From here the admin creates projects, defines their basic attributes

and onboards new participants. User management in this phase triggers the interaction with the Certificate Authority to issue the appropriate credentials for the new identity. The admin can also review audit entries at a high level, see when documents were notarised on the ledger and trace transaction identifiers when an audit or an incident review requires it.

The reviewer view focuses on evidence and decisions. It opens with the list of tasks that require attention, highlights missing or outdated documents and offers quick actions to request an update or record an approval. When a file is present the reviewer can download it, open the IFC model if the task references one, and run an integrity check that recomputes the hash and compares it to the on chain anchor. The screen shows the outcome in plain terms and stores the result in the task history so that the team can see what changed and when.

The standard user view keeps complexity low and guides the workflow. It shows assigned tasks, due dates and the list of required documents. Uploading a file is straightforward: the user selects the file, adds a short description and submits. The backend performs the hash calculation, stores the operational data and initiates the blockchain transaction by retrieving the user's credentials from the secure Wallet. The screen then confirms the operation and shows the transaction identifier and the current verification state. If the user opens an IFC based task, the model viewer is available without leaving the page so the person can check the context before attaching the document.

Across all views the platform maintains consistent behaviour. Authentication is handled with tokens (JWT) for session management, while authorization follows a Defense-in-Depth strategy. Permissions are checked first at the API boundary for user experience purposes, and then rigorously enforced at the Smart Contract level via Attribute-Based Access Control (ABAC). Every write returns a clear confirmation and a transaction identifier; the application records these identifiers in its history so that an admin or a reviewer can later trace an event in the ledger if needed. Read operations are fast and use the local database, while verification paths call the ledger to confirm integrity. This balance keeps the experience responsive for everyday use while providing mathematical evidence when the project must prove what happened.

The navigation reflects this structure. The admin area groups project and user management together with high level audits. The reviewer area groups the task queue, document checks and approvals. The standard user area provides a clean layout for assigned tasks, uploads and quick access to project information. The IFC model viewer is available wherever a task references a model so that geometry

and documents can be reviewed side by side. Notifications and status badges help users see at a glance which items need attention and which ones are complete and verified. The result is a consistent high level workflow that matches the way project teams operate while adding a tamper-proof integrity layer through the ledger.

6.3 Blockchain Smart Contract for Document Management

The smart contract named `DocumentStorageContract` has been developed to manage project documentation within the proposed Blockchain & BIM Platform. It represents the on-chain logic responsible for storing, validating, and tracking digital documents in a secure and verifiable way, enforcing the Identity-Based logic described in the architecture.

6.3.1 General Overview

The contract was designed using the `Hyperledger Fabric` framework. Each function interacts with the distributed ledger, ensuring immutability and traceability of project files. Unlike standard storage contracts, this implementation adopts a "Zero Trust" approach: it does not rely on user inputs for critical metadata (such as the uploader's identity or timestamp) but derives them cryptographically from the transaction context.

The smart contract includes several key functions grouped into three main areas:

- **Document management:** creation, retrieval, deletion, and verification of files with embedded access control checks.
- **Version and history management:** tracking of different document versions over time, ensuring the provenance of each update.
- **Certification and audit features:** creation of digital certificates and filtering of uploaded files by user or project for compliance auditing.

6.3.2 Core Functions

The following function allows a user to upload a new document into the blockchain ledger. The function implements Attribute-Based Access Control (ABAC). Before processing the data, it verifies that the caller possesses the specific **'standard_user'** or **'admin'** role attribute in their X.509 certificate. Furthermore, the user identity is not passed as an argument but is extracted securely from the Client Identity

(CID) library, preventing spoofing attacks.
The uploadDocument function ensures that each document is uniquely stored and cryptographically bound to the specific user who signed the transaction. The use of the hash provides integrity verification, while the CID extraction guarantees non-repudiation.

```
async uploadDocument(ctx, id, filename, hash) {
  // 1. Initialize Client Identity to inspect the caller's certificate
  const cid = new ClientIdentity(ctx.stub);

  // 2. ABAC: Check if the user has the required role attribute
  // Only Standard Users or Admins (Org1) can upload new documents
  if (!cid.assertAttributeValue('app_role', 'standard_user') &&
      !cid.assertAttributeValue('app_role', 'admin')) {
    throw new Error('Authorization Failed: Caller lacks required privileges.');
  }

  // 3. Securely derive identity and timestamp from the ledger context
  // getID() returns the unique X.509 Subject DN
  const userId = cid.getID();
  const userOrg = cid.getMSPID();
  // Deterministic timestamp generated by the orderer
  const timestamp = ctx.stub.getTxTimestamp();

  const exists = await this.documentExists(ctx, id);
  if (exists) {
    throw new Error(`Document ${id} already exists`);
  }

  const document = {
    id,
    filename,
    hash,
    owner: userId,     // Securely derived, not passed as argument
    org: userOrg,      // Organization ID
    timestamp,
    docType: 'document'
  };

  await ctx.stub.putState(id, Buffer.from(JSON.stringify(document)));
  return JSON.stringify(document);
}
```

Listing 6.1: Secure Upload with Identity Derivation

6.3.3 Retrieving and Verifying Documents

To access a specific file, the `getDocument` function retrieves the document from the ledger. Its counterpart, `verifyDocument`, allows checking whether a given hash is already stored, confirming the document's authenticity against the immutable record.
These functions are fundamental to enable data retrieval and integrity checks, which are essential for compliance and transparency within construction projects.

```
async getDocument(ctx, id) {
  const data = await ctx.stub.getState(id);
  if (!data || data.length === 0) {
    throw new Error(`Document ${id} does not exist`);
  }
  return data.toString();
}

async verifyDocument(ctx, hash) {
  // This function uses a CouchDB rich query to search by hash
  // assuming the hash is indexed for performance
  const queryString = {
      selector: {
          docType: 'document',
          hash: hash
      }
  };
  const iterator = await ctx.stub.getQueryResult(JSON.stringify(
   queryString));
  const result = await iterator.next();
  // Returns true if at least one record matches the hash
  return !result.done;
}
```

Listing 6.2: Document retrieval and verification

6.3.4 Version and History Management

Version control is a relevant feature for the construction industry, where design files and technical documents frequently evolve. The function below allows to store and manage new versions of an existing document. Similar to the upload function, it strictly validates the submitter's identity to ensure that only authorized actors can append updates to the history.
This functionality ensures that each file version remains accessible and that previous states are never lost, providing a granular audit trail of who modified the document and when.

```
async addDocumentVersion(ctx, id, newHash) {
  const cid = new ClientIdentity(ctx.stub);

  // Security Check: Ensure caller is authorized (Standard User or Admin)
  if (!cid.assertAttributeValue('app_role', 'standard_user') &&
      !cid.assertAttributeValue('app_role', 'admin')) {
     throw new Error('Authorization Failed.');
  }

  const userId = cid.getID();
  const timestamp = ctx.stub.getTxTimestamp();

  const documentBytes = await ctx.stub.getState(id);
  if (!documentBytes || documentBytes.length === 0) {
    throw new Error(`Document ${id} does not exist`);
  }

  const document = JSON.parse(documentBytes.toString());

  // Append new version with secure provenance
  // We record WHO is making this specific update
  document.versions.push({
    hash: newHash,
    updatedBy: userId,
    timestamp
  });

  await ctx.stub.putState(id, Buffer.from(JSON.stringify(document)));
  return JSON.stringify(document);
}
```

Listing 6.3: Secure Document version management

6.3.5 Audit and Filtering Functions

Finally, additional functions, such as `getDocumentsByOwner`, enable improved governance. Since the owner field is populated by the verified certificate ID, these queries provide high-assurance audit capabilities for project managers and reviewers.

```
async getDocumentsByOwner(ctx, targetUserId) {
  // Uses CouchDB selector to filter by the specific identity ID
  const query = {
    selector: {
      docType: 'document',
      owner: targetUserId
```

```
7       }
8     };
9     const iterator = await ctx.stub.getQueryResult(JSON.stringify(
        query));
10    const results = [];
11    while (true) {
12      const res = await iterator.next();
13      if (res.value && res.value.value.toString()) {
14        results.push(JSON.parse(res.value.value.toString('utf8')));
15      }
16      if (res.done) break;
17    }
18    await iterator.close();
19    return JSON.stringify(results);
20  }
```

Listing 6.4: Filtering documents by verified owner

6.3.6 Discussion

In conclusion, the `DocumentStorageContract` represents a security-hardened document management logic aligned with the goals of the B&B Platform. By embedding Attribute-Based Access Control and identity derivation directly into the chaincode, it demonstrates how blockchain can provide not just transparency, but active enforcement of governance rules that cannot be bypassed by the application layer.

6.4 Test Network Deployment

This section explains how the local Hyperledger Fabric network was deployed for the Proof of Concept and how I verified that the smart contract behaved correctly in a real environment. The main goal of this phase was to create a reproducible setup that allowed me to observe every component of the network while it executed real transactions, from identity creation to ledger updates.
The entire environment was based on Docker and Docker Compose, using the official Fabric images and binaries (version 2.x for Fabric and 1.x for Fabric CA). Running the test network locally made it easier to understand what was happening inside each container and to reproduce the same setup multiple times during development.
After generating the cryptographic material for the two organizations, Org1 and Org2, I started the network and created a dedicated application channel called `mychannel`. From that moment, the peers of both organizations were connected to the same ordering service and could exchange endorsed transactions within the shared channel.

Once the network was up and running, I deployed the smart contract described earlier in Chapter 6. Following the chaincode lifecycle model presented in Chapter 5, the contract was first packaged, then installed on the peers of both organizations, approved, and finally committed to the channel. I carried out all these steps directly from the command line. Working in the terminal allowed me to see every stage clearly, including which organization approved the definition and how the endorsement process unfolded. This approach helped me verify that each transaction followed the intended logic before being accepted into the ledger.

To test the deployment, I executed a simple but complete transaction: uploading a document record representing a BIM file. Before invoking the chaincode, I configured the CLI environment variables to use the cryptographic material (MSP) of an authorized user from Org1, simulating a legitimate session. The command-line interface reported a successful invocation with status `200` and returned the expected JSON payload. The subsequent query retrieved the record, confirming that the `owner` field was correctly populated by the smart contract deriving the identity from the transaction signature, rather than from an input argument.

During these tests I also monitored the logs of the peer containers to trace the endorsement, validation, and block commitment phases. This process was useful for identifying any discrepancies, such as mismatched arguments or missing approvals, which would immediately prevent the transaction from being applied. Running the tests from the terminal, instead of relying on a graphical interface, made the behavior of the network completely transparent and easier to debug.

In conclusion, the test network provided a clear and controlled environment to validate the functionality of the `DocumentStorageContract` and to ensure that two independent organizations could interact correctly through a shared channel. The results confirmed that the contract logic was executed consistently and that the recorded data remained synchronized across all peers. This setup became the technical foundation for the next phase of the project, where the topology was extended and integrated with the upper application layer.

6.5 Adding and Managing a Peer

The expansion of the network topology beyond the minimal configuration marked an important step in aligning the Proof of Concept with the reliability requirements of the Hosted Consortium architecture. In the initial phase, a single peer per organization was sufficient to validate the smart contract logic. However, given that Org1 acts as the infrastructure provider for multiple construction companies, operating

a single node would represent a single point of failure for the entire consortium. Therefore, the network was extended to implement High Availability (HA): both Org1 and Org2 now operate two distinct peers, ensuring that endorsement services and ledger access remain uninterrupted even during maintenance or faults.

The operational process behind the addition of a new peer follows a predictable structure but requires careful configuration to maintain the security boundaries. For each organization, a new service was defined in the Docker Compose file, reusing the same hardened peer image while assigning independent MSP directories and separate volumes for the ledger and the state database. This choice keeps the data of each node isolated, simplifies clean-up during iterative testing and reduces the likelihood of interference between peers belonging to the same organization. Once the containers were launched, the new peers inherited the environment variables of their existing counterparts, including the orderer addresses and the organization's MSP context, but naturally started with an empty ledger.

Joining the channel represents the next phase. Each new peer was configured through the organization's CLI context, retrieved the genesis block of `mychannel` and executed the join operation. From that moment onward, the peer began to synchronize with the network using the gossip protocol, downloading missing blocks until reaching the current block height. This behavior is visible both from the container logs and from inspection commands such as `peer channel getinfo`, which report the same block height across all peers once the synchronization is complete.

The extension also required revisiting the chaincode lifecycle. In Hyperledger Fabric 2.x, the approval of a chaincode definition is associated with an organization rather than with individual peers. As a result, once the definition of the smart contract had been approved and committed by the governance of Org1 and Org2, introducing additional peers did not modify the definition itself. The only necessary step was to install the same chaincode package on the new nodes. This step is critical: without the installed bytecode, the new peers cannot simulate the ABAC checks described in the previous section, and thus cannot act as endorsers.

To ensure that the network behaved correctly with the extended topology, several validation steps were carried out. First, all four peers were checked to ensure that they were running and had successfully joined the channel. The document upload and verification transactions were then repeated. Thanks to the Service Discovery feature of the Fabric Gateway used in the backend, the application automatically detected the new peers and load-balanced the endorsement requests across them without requiring code changes.In every case, the transactions were

endorsed, ordered, and committed, and the resulting blocks appeared in the logs of the corresponding container.

The extension of the network also provided an opportunity to observe certain practical aspects of peer management. A newly added peer initially experiences a brief period of heavy activity while synchronizing the ledger, after which its behavior stabilizes. The separation of volumes for the ledger and the state database proved useful during repeated experiments, as it limited the amount of data that needed to be regenerated.

Overall, the addition and management of multiple peers demonstrated that the Hosted Consortium can scale horizontally without altering the application interface or the guarantees provided by the endorsement policy. The infrastructure gains redundancy and operational flexibility, providing a solid foundation for the security-related experiments addressed in the following section, where the presence of multiple peers per organization becomes an important element in assessing network resilience under adverse conditions.

6.6 Platform Integration and BIM Visualization

The application layer integrates all components of the Proof of Concept into a single environment where users authenticate, manage project information and interact with documents whose integrity is anchored to the blockchain. While earlier sections introduced the individual roles and the general structure of the platform, this part focuses on the technical implementation of the Dynamic Identity Management system, explaining how the platform bridges the gap between traditional web authentication and the cryptographic requirements of Hyperledger Fabric.

Authentication is handled on the server side by a Node.js backend acting as a Transparent Proxy. User credentials (username/password) are stored in a local SQL database with salted and hashed passwords. However, this is only the first layer of security. To ensure true non-repudiation on the ledger, the system implements a Identity Mapping mechanism: each user in the SQL database is mapped to a specific X.509 certificate stored in a server-side Fabric Wallet. When a client authenticates via JWT, the backend does not simply grant access to API endpoints; it loads the corresponding cryptographic identity from the Wallet to prepare for blockchain interactions.

The dashboard represents the central point of interaction for all roles and brings

together project information, task assignments and document status. At this layer the system relies on SQLite, which is lightweight but sufficiently structured to support the relationships between projects, tasks and documents. The database stores operational information only, while all integrity-related data flows toward the blockchain.

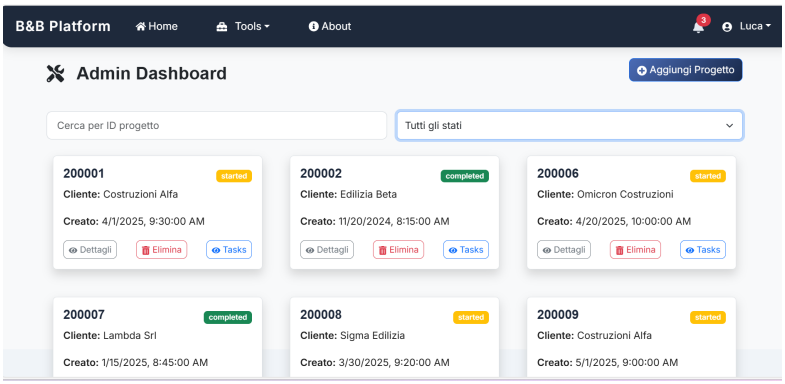

Figure 6.1: Dashboard of the B&B Platform, showing project information and document status. This layer handles operational data locally while relying on the blockchain for integrity anchoring.

The transaction submission flow is designed to preserve the user's identity context. When a user uploads a file, the backend executes the following sequence:

1. **Context Resolution:** The backend extracts the user ID from the session token.
2. **Wallet Lookup:** The system queries the FileSystemWallet to retrieve the specific X.509 credentials associated with that user (e.g., `user_mario@org1`).
3. **Gateway Connection:** A connection to the Fabric Gateway is established using that specific user's identity.
4. **Submission:** The transaction is signed with the user's private key and submitted to the network.

This mechanism ensures that the Smart Contract receives a transaction signed by the actual operator, enabling the Attribute-Based Access Control (ABAC) checks described in Section 6.3.

The integration with BIM content represents the most distinctive functional part of the application layer. The platform embeds a 3D viewer capable of rendering IFC models directly in the browser using a JavaScript library suited for this purpose.

The viewer allows users to explore the model, inspect properties and focus on specific elements without exporting data to external tools. In tasks that require the submission of technical files, the model helps users contextualise their documents before uploading them, and reviewers can cross-check the submitted material against the geometry. This feature is tightly paired with the blockchain component: while the model remains off-chain for performance reasons, its associated documents are anchored to the ledger, enabling integrity checks at any time.

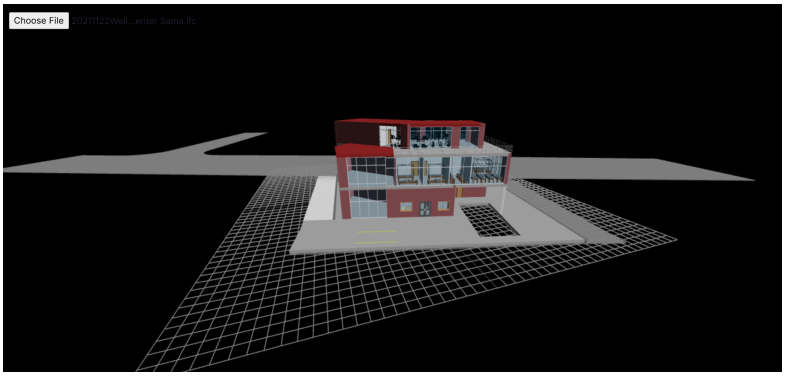

Figure 6.2: Integrated IFC model viewer. The model is rendered in the browser while its associated documents are anchored to the ledger for integrity verification.

In summary, the application layer brings coherence to the entire Proof of Concept. It offers a familiar interface for daily operations while implementing a sophisticated identity bridging logic in the background. By coupling a lightweight relational database with blockchain-based verification, the platform supports real projects while embedding integrity guarantees directly into the document lifecycle. The next section builds on this integrated architecture to examine how the system behaves under security-focused scenarios and in the presence of adverse conditions.

6.7 Cybersecurity Testing Scenario

After validating the standard workflow, the focus shifts to security resilience. This section presents specific stress tests designed to verify whether the system can effectively withstand data tampering attempts and strictly enforce governance rules under adverse conditions.

6.7.1 Integrity Violation on the Application Database

This first experiment assesses the integrity guarantees provided by the blockchain layer when external storage undergoes deliberate alteration. The Proof of Concept

(PoC) bases its operations on a standard SQLite backend, which stores metadata like filenames, timestamps, and upload hashes. We deliberately configured this database to be insecure: it lacks tamper-resistance mechanisms, meaning any actor with server access has full freedom to modify the records. The objective is to verify if the system detects these inconsistencies by checking against the ledger as the definitive source of truth.

The simulated attack scenario involves an adversary breaching the host server to write directly to the SQLite file. The sequence is simple: a legitimate user uploads a document, and the backend generates a SHA-256 digest, recording it both locally and on-chain via the DocumentStorageContract. Next, the attacker targets the local database, silently altering the document's stored hash but ignoring the ledger entry. To the application, this appears as a "silent corruption" typical of centralized systems, where such changes often go unnoticed without specific monitoring tools.

To test if the platform exposes this manipulation, the experiment triggers the built-in integrity check. Upon a verification request, the backend ignores the local metadata; instead, it reads the file directly from the disk, computes a fresh hash, and demands a comparison from the on-chain registry. Since endorsement policies protect the ledger's append-only history, the original hash survives regardless of local database edits. This divergence triggers a system alert, notifying the user of the integrity failure immediately, despite the successful manipulation of the local database.

This outcome validates the architectural choice of decoupling off-chain data from on-chain integrity anchors. An attacker might compromise the SQLite backend, but the history stored on the ledger remains out of reach. Functioning as an unyielding reference point, the blockchain forces the detection of unauthorized modifications. The result is a clear audit trail for attacks that centralized architectures would essentially overlook, demonstrating how distributed ledgers add value without needing to replace the entire application stack.

6.7.2 Enforcement of the Endorsement Policy under Faults or Attacks

This second phase shifts the focus from data integrity to network governance. Specifically, we need to validate the robustness of the Endorsement Policy within Hyperledger Fabric. In a distributed environment, this policy dictates the consensus rules required to validate a transaction; the objective here is to confirm that these rules hold up strictly, even when the network faces critical hardware failures or

unilateral manipulation attempts.

Threat Model and Network Configuration

The experiment models two primary threats. First, we consider an internal adversary who possesses exclusive control over a single organization (Org1 or Org2) and attempts to force a ledger update without the counterparty's consent. Second, we look at fault tolerance, where hardware issues render nodes from one or more organizations unavailable.
To counter these risks, the basic chaincode deployed on `mychannel` operates under a restrictive policy:

```
AND('Org1MSP.peer','Org2MSP.peer')
```

This imposes a hard constraint: no transaction can commit to the ledger unless it carries a cryptographic signature from at least one peer in each organization. The initial setup assumes full operational capacity, with two active nodes per member (peer0 and peer1 for both Org1 and Org2).

Methodology and Progressive Execution

The test follows a "progressive degradation" strategy. We start with optimal conditions and gradually remove components of the network until a single organization remains active.

- **Baseline:** With the entire network online, we invoked the `uploadDocument` function. The system responded as expected: peers from both organizations simulated the transaction, attached their signatures, and the client submitted the package to the ordering service. The ledger updated successfully (Status 200), confirming correct configuration.
- **Partial Faults:** We deliberately shut down the secondary nodes (peer1) for both organizations, leaving only one peer active on each side. Upon sending a new transaction, the system maintained full operability. The remaining peers (peer0) satisfied the policy requirements, proving that the network can tolerate individual node failures without service interruption, provided at least one validator remains active per required organization.
- **Total Separation / Attack Simulation:** All Org2 peers were forced offline, leaving Org1 as the sole active entity on the channel. Invoking the chaincode using only Org1's peers mirrors a scenario where a rogue administrator tries to alter system state—such as modifying document metadata—without the approval of the other party. The transaction attempt failed before it could reach the ledger: although Org1's peers executed the simulation and signed

the proposal, the client could not assemble a valid transaction package because the mandatory signature from Org2 was missing.

Table 6.1: Summary of results for the endorsement policy enforcement experiment.

Step	Active Peers	Expected Outcome	Observed Outcome
Baseline	All peers	Transaction approved	Transaction approved (Status 200)
Partial Faults	peer0 Org1 + peer0 Org2	Transaction approved	Transaction approved (Status 200)
Total Separation	Only Org1 peers	Transaction rejected	Transaction rejected (endorsement policy failure)

Analysis and Implications for BIM

The results confirm the effectiveness of the enforcement mechanism and illustrate several key principles:

- **Operational Resilience:** The network survives specific node losses without service degradation.
- **Separation of Powers:** No single organization holds absolute control. Even with full administrative rights for Org1, an attacker cannot bypass the need for Org2's consent.
- **Defense in Depth:** Security is distributed through cryptographic policy verification, not reliant on a single checkpoint.

In the context of Building Information Modeling (BIM) workflows, the AND policy functions as a digital contract, preventing, for example, a construction firm from altering a technical drawing or cost estimate without explicit (cryptographic) validation from the client or architects. The system enforces procedural agreements as unbreakable technical constraints, drastically reducing the risk of disputes based on tampered or misaligned data.

6.7.3 Enforcement of the Endorsement Policy under Faults and Adversarial Conditions

This experiment investigates the enforcement of Hyperledger Fabric's endorsement policy under scenarios where one organization becomes unavailable, either due to failures or malicious attempts to bypass governance. Endorsement policies in Fabric define cryptographic rules that ensure multi-party agreement before a transaction can modify the ledger. In multi-stakeholder environments, such as Building Information Modeling (BIM) workflows, strict enforcement of these policies guarantees that no single participant can unilaterally alter shared data.

Network Configuration

The network consists of two organizations: `Org1` and `Org2`. Each organization contributes two peers, `peer0` and `peer1`, which serve as both endorsers and committers. The chaincode `basic` is installed on all peers, and the endorsement policy is set as:

`AND('Org1MSP.peer', 'Org2MSP.peer')`.

This means that for any transaction to be valid, it must carry at least one endorsement from a peer in each organization. This policy is central to maintaining integrity and trust in collaborative operations.

Threat Model

We consider two main threat scenarios:

1. An **internal adversary** who controls all peers in a single organization (`Org1`) and attempts to submit a transaction without obtaining endorsements from the other organization.

2. **Node failures** that render all peers of one organization (`Org2`) offline, representing hardware or network faults.

Both scenarios are realistic in multi-stakeholder networks, where data integrity and accountability are critical.

Experimental Procedure

The experiment is carried out in three progressive stages:

1. **Baseline: All peers online.**
   In this initial phase, a document upload transaction (`uploadDocument`) is invoked. Both organizations' peers endorse the proposal, and the transaction is successfully committed to the ledger. This establishes the expected behavior under normal operating conditions.

2. **Partial Faults: Secondary peers offline.**
   The secondary peers (`peer1`) of both organizations are intentionally shut down. A new transaction is invoked. Despite the reduction in available peers, the remaining primary peers (`peer0`) provide sufficient endorsements to satisfy the policy. The transaction is committed successfully, demonstrating the network's resilience to individual node failures.

3. **Total Separation: Org2 peers offline.**
   Both peers of Org2 are shut down, leaving only Org1 peers active. The client attempts to invoke the chaincode to upload a new document. This simulates a rogue administrator trying to update the ledger unilaterally without the required endorsement from Org2.

Compact Illustration of the Flow

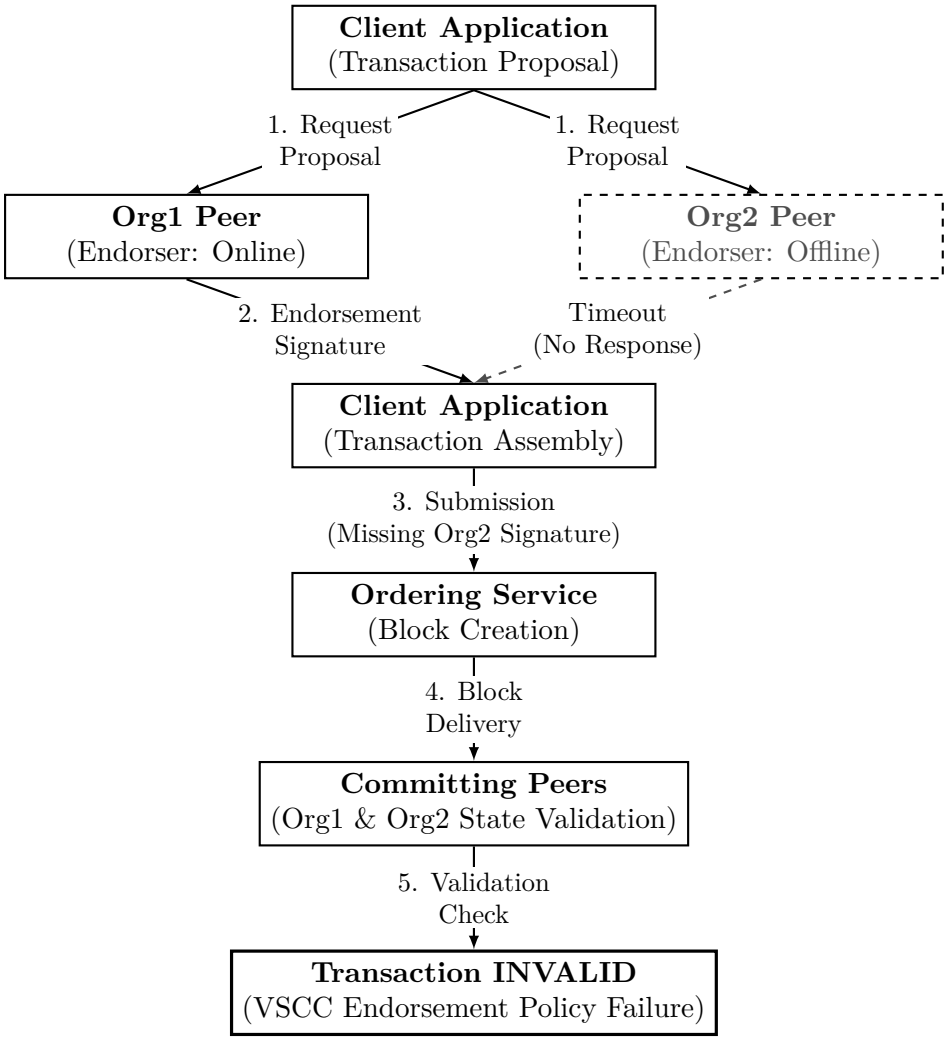

Figure 6.3: Transaction flow under adversarial conditions or hardware faults. The diagram structurally outlines the lifecycle of a proposal that fails to satisfy the multi-organization endorsement policy, illustrating how the absence of a mandatory cryptographic signature leads to a VSCC validation rejection.

Relevance for BIM Workflows

In collaborative BIM scenarios, documents such as design files, cost estimates, and technical drawings are critical. The endorsement policy enforces a cryptographic approval process, preventing any participant from silently modifying shared assets. This mechanism translates procedural agreements into technical guarantees, making ledger tampering practically impossible and ensuring accountability among stakeholders.

Conclusion

The test demonstrates that Hyperledger Fabric enforces multi-organization governance robustly. Even under adverse conditions, including the unavailability of one organization's peers, the ledger's integrity is preserved. This ensures that all participants must cooperate to effect changes, making the system suitable for audit-sensitive applications where trust and immutability are paramount.

6.8 Results and Observations

The experimental campaign conducted on the Proof of Concept has validated the core architectural hypothesis: that a permissioned blockchain can effectively act as a trust anchor for industrial document management without replacing the efficiency of traditional databases. The observations collected during the testing phase offer insight into both the security guarantees and the operational viability of the proposed solution.

From a security perspective, the results obtained in the integrity violation scenario demonstrate that the "on-chain and off-chain" separation is robust. When the local SQLite database was subjected to direct manipulation, simulating a silent corruption or an internal attack, the platform successfully identified the discrepancy. This confirms that while the application layer remains vulnerable to traditional intrusion vectors, the immutability provided by the ledger prevents these attacks from going undetected. The system forces a "trust but verify" workflow where the blockchain acts as an incorruptible witness to the true state of the project documentation.

Furthermore, the stress tests performed on the endorsement policy revealed the resilience of the Hyperledger Fabric network under adverse conditions. The system behaved deterministically during the partial fault scenarios; as long as a single peer per organization remained active, operations continued without latency penalties or service interruptions. More importantly, the "Total Separation" test confirmed the

governance model's strength. Even with full administrative control over Organization 1, it was technically impossible to alter the ledger without the cryptographic consent of Organization 2. This result is particularly significant for the construction industry context, as it mathematically enforces the contractual requirement that no single stakeholder can unilaterally rewrite the project history.

On the operational side, the integration of the BIM viewer with the blockchain backend proved that security features need not degrade the user experience. The latency introduced by the transaction proposal and ordering process was perceptible but remained within acceptable limits for document registration tasks, which are not real-time sensitive. However, it was observed that the initial synchronization of new peers involves a significant bandwidth overhead, suggesting that in a production environment, snapshot-based bootstrapping might be necessary to scale the network efficiently.

Ultimately, the Proof of Concept demonstrates that the complexity of Hyperledger Fabric can be effectively abstracted behind a familiar web interface. The platform successfully bridges the gap between the rigid security requirements of a distributed ledger and the dynamic needs of construction project management, providing a verifiable audit trail that survives both infrastructure failures and malicious internal actors.

Chapter 7

Conclusion

7.1 Conclusion

The research presented in this thesis originated from a direct professional experience within the Industry X team at Accenture. This practical immersion highlighted a critical issue that affects large industrial projects. The primary challenge is often not a lack of advanced digital tools but rather a fundamental lack of trust between the various stakeholders involved. In the current landscape of construction and engineering it is evident that significant resources are expended not on productive activities but on verifying documents and reconciling conflicting records. The objective of this work was therefore to design a solution that does not attempt to replace necessary human interactions but instead provides a reliable technological framework to support them.

Developing the Proof of Concept meant keeping things grounded in engineering reality. We know that blockchain theory is huge, but we also know the industry runs on specific habits and workflows that you can't just throw out the window overnight. For this reason the decision was made to adopt a hybrid architecture. Large operational files are retained in traditional storage systems while only the cryptographic proof of their integrity is anchored to the blockchain. The results obtained during the testing phase confirmed the validity of this choice as it allowed the platform to remain efficient and responsive while offering a level of security that centralized systems are unable to provide.

The experimental simulations conducted on the platform revealed significant insights regarding digital collaboration. When the system was subjected to simulated internal attacks or unilateral attempts to alter data, the governance model functioned exactly as intended. It effectively prevented any single entity from modifying

the project history without the explicit consent of the counterparty. This change completely alters the dynamic of project management. It gives engineers and managers the peace of mind that once a file is approved, it stays that way forever, verifiable at any moment. We essentially stop relying on people checking other people's work and start relying on a mathematical process that doesn't make mistakes.

Looking beyond the current implementation this prototype serves as a foundation for more advanced future developments. Although the current system primarily addresses document tracking and verification, it has the potential to expand into the financial and contractual domains of project management. In the next versions, we could hook up smart contracts that release payments the second a deliverable gets validated. This kind of automation would cut out the administrative waiting times that hurt cash flow for everyone. Also, we need to look into ways to make the network handle way more suppliers and contractors without it getting slow or clunky.

There is also a broader lesson derived from this work that extends beyond the technical specifications. Building this taught us that innovation only really works if it fixes a specific headache without creating new ones for the user. The real win here is that we hid all the complicated blockchain machinery behind a screen that looks just like normal software. It shows that you can bring serious security tools into old-school industries without asking workers to learn a whole new skillset.

In conclusion this thesis demonstrates that the industry is ready for a new paradigm of shared responsibility. By addressing the chronic issues of data fragmentation and low trust we are proposing a tangible improvement to the way industrial projects are executed. We are moving towards a shared environment where stakeholders can dedicate their efforts to construction and value creation rather than litigation and dispute resolution. The prototype developed here stands as proof that innovation and pragmatism can coexist successfully. As we look to the future the ultimate goal is for these technologies to become an invisible yet essential part of our infrastructure so that collaboration is no longer defined by the management of risk but by the confidence in shared truth

Bibliography

[1] Accenture plc. *Fact Sheet Fiscal Year 2025*. 2025. URL: https://newsroom.accenture.com/fact-sheet (cit. on p. 2).

[2] Accenture plc. *What is Industry X?* URL: https://www.accenture.com/content/dam/accenture/final/a-com-migration/manual/r3/pdf/pdf-142/Accenture-What-Is-Industry-X.pdf (cit. on p. 2).

[3] National Institute of Standards and Technology. *Announcing the Advanced Encryption Standard (AES)*. FIPS Publication 197. NIST, 2001. DOI: 10.6028/NIST.FIPS.197. URL: https://doi.org/10.6028/NIST.FIPS.197 (cit. on pp. 4, 5).

[4] *What is symmetric encryption?* IBM. 2024. URL: https://www.ibm.com/topics/symmetric-encryption (cit. on pp. 4, 5).

[5] IBM. *Asymmetric Encryption – IBM Think*. https://www.ibm.com/think/topics/asymmetric-encryption. 2024 (cit. on pp. 5, 6).

[6] IBM. *Cryptographic Algorithm and Key Length – IBM Docs*. https://www.ibm.com/docs/en/gklm/4.2.0?topic=overview-cryptographic-algorithm-key-length. 2024 (cit. on p. 6).

[7] Elaine Barker. *NIST Special Publication 800-57 Part 1 Revision 5: Recommendation for Key Management*. Tech. rep. National Institute of Standards and Technology (NIST), 2020. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf (cit. on p. 6).

[8] National Institute of Standards and Technology (NIST). *Secure Hash Standard (SHS)*. Tech. rep. FIPS PUB 180-4. U.S. Department of Commerce, 2015. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (cit. on pp. 6, 7).

[9] National Institute of Standards and Technology (NIST). *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Tech. rep. FIPS PUB 202. U.S. Department of Commerce, 2015. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf (cit. on pp. 6–8).

[10] National Institute of Standards and Technology (NIST). *Digital Signature Standard (DSS)*. Tech. rep. FIPS PUB 186-4. U.S. Department of Commerce, 2013. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf (cit. on p. 8).

[11] William Stallings. *Cryptography and Network Security: Principles and Practice*. 9th. Pearson, 2023 (cit. on pp. 8, 9).

[12] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch. *PKCS #1: RSA Cryptography Specifications Version 2.2*. RFC 8017. Nov. 2016. DOI: 10.17487/RFC8017. URL: https://www.rfc-editor.org/info/rfc8017 (cit. on p. 8).

[13] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280, IETF. 2008. URL: https://datatracker.ietf.org/doc/html/rfc5280 (cit. on p. 9).

[14] M. Dworkin. *Recommendation for Applications Using Approved Hash Algorithms*. Tech. rep. NIST Special Publication 800-107 Revision 1. National Institute of Standards and Technology (NIST), 2012. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-107r1.pdf (cit. on pp. 9, 10).

[15] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*. Tech. rep. NIST Special Publication 800-38B. National Institute of Standards and Technology (NIST), 2005. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38b.pdf (cit. on p. 10).

[16] IBM. *What is Public Key Infrastructure (PKI)?* https://www.ibm.com/think/topics/public-key-infrastructure. 2025 (cit. on pp. 10, 11).

[17] Adam Back. *A Partial Hash Collision Based Postage Scheme*. https://www.hashcash.org/papers/announce.txt. 1997. (Visited on 10/30/2025) (cit. on pp. 13, 14).

[18] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: https://bitcoin.org/bitcoin.pdf (cit. on pp. 14, 18).

[19] Nick Szabo. *Smart Contracts: Building Blocks for Digital Markets*. Unpublished manuscript. 1997. URL: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html (cit. on pp. 14, 15).

[20] Nick Szabo. *Formalizing and Securing Relationships on Public Networks*. First Monday. 1997. URL: https://firstmonday.org/ojs/index.php/fm/article/view/548 (cit. on p. 15).

[21] Wei Dai. *b-money*. https://www.weidai.com/bmoney.txt. 1998. (Visited on 10/30/2025) (cit. on p. 16).

[22] Hal Finney. *RPOW - Reusable Proofs of Work*. Post to the Cypherpunks mailing list, Aug 15, 2004. 2004. URL: https://nakamotoinstitute.org/library/rpow/ (visited on 10/30/2025) (cit. on p. 17).

[23] Nick Szabo. *Bit Gold*. https://nakamotoinstitute.org/library/bit-gold/. 2005. (Visited on 10/30/2025) (cit. on p. 18).

[24] Vitalik Buterin. *A Next-Generation Smart Contract and Decentralized Application Platform*. https://ethereum.org/en/whitepaper/. 2014. (Visited on 11/03/2025) (cit. on p. 19).

[25] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. *Blockchain Technology Overview*. Tech. rep. NIST Interagency/Internal Report (NISTIR) 8202. National Institute of Standards and Technology (NIST), 2018. DOI: 10.6028/NIST.IR.8202. URL: https://nvlpubs.nist.gov/nistpubs/ir/2018/nist.ir.8202.pdf (cit. on pp. 20–28).

[26] Hyperledger Fabric Documentation. *Prerequisites*. 2025. URL: https://hyperledger-fabric.readthedocs.io/en/latest/prereqs.html (visited on 11/06/2025) (cit. on pp. 32, 33).

[27] Hyperledger Fabric Documentation. *Using the Fabric Test Network*. 2025. URL: https://hyperledger-fabric.readthedocs.io/en/latest/test_network.html (visited on 11/06/2025) (cit. on pp. 32, 33).

[28] Hyperledger Fabric Documentation. *CouchDB as the State Database*. 2025. URL: https://hyperledger-fabric.readthedocs.io/en/latest/couchdb_as_state_database.html (visited on 11/06/2025) (cit. on pp. 32, 33).

[29] Hyperledger Fabric Documentation. *Fabric Gateway*. 2025. URL: https://hyperledger-fabric.readthedocs.io/en/latest/gateway.html (visited on 11/06/2025) (cit. on pp. 32, 33).

[30] Hyperledger Fabric Documentation. *Fabric Chaincode Lifecycle*. 2025. URL: https://hyperledger-fabric.readthedocs.io/en/latest/chaincode_lifecycle.html (visited on 11/06/2025) (cit. on pp. 32, 33).

[31] The Linux Foundation. *Channels — Hyperledger Fabric Documentation*. 2025. URL: https://hyperledger-fabric.readthedocs.io/en/latest/channels.html (cit. on p. 36).

[32] Hyperledger Fabric Documentation. *Membership Service Provider (MSP)*. 2025. URL: https://hyperledger-fabric.readthedocs.io/en/latest/membership/membership.html (cit. on p. 39).

[33] Christian Cachin and Marko Vukolić. *Blockchain Security and Privacy in Hyperledger Fabric*. Tech. rep. IBM Research, 2019. URL: https://arxiv.org/abs/1801.10228 (cit. on p. 39).

[34] Hyperledger Fabric Documentation. *Fabric Private Chaincode (FPC)*. 2025. URL: https://hyperledger-fabric.readthedocs.io/en/latest/private_chaincode.html (cit. on p. 39).

[35] Timothy C. May. *The Crypto Anarchist Manifesto*. https://www.activism.net/cypherpunk/crypto-anarchy.html. 1992. (Visited on 10/30/2025).