UNIVERSITY OF VERONA

DEPARTMENT OF
Computer Science

DOCTORAL PROGRAM IN
Computer Science
XXXVIII cycle (2023)

Next-Generation Blockchain Systems: Innovations in Smart Contracts, Layer-2 Data Availability, Privacy-Preserving Compliance and Certification

S.S.D. IINF-05/A

Advisor: **Prof. Sara Migliorini**
Co-Advisor: **Prof. Mila Dalla Preda**

PhD program chair: **Prof. Umberto Castellani**

Candidate: **Muhammad Bin Saif**

ISBN: 9788869251801

Abstract

The rapid digitalization of modern society has transformed how information is stored, transactions are executed, and trust is established in digital environments. Blockchain technology has emerged as a promising foundation for decentralized systems by providing immutability, transparency, and trustworthiness. Despite these advancements, real-world blockchain adoption reveals fundamental limitations. Smart contracts still lack modular and secure design patterns for managing complex access control requirements. Blockchain systems face scalability and data availability constraints, and the immutability of on-chain data often conflicts with regulatory compliance, such as the General Data Protection Regulation (GDPR). At the same time, blockchain-based certification systems must balance public verifiability with the privacy of sensitive information, a challenge that current approaches struggle to address.

This thesis addresses these challenges by advancing the design of blockchain systems across multiple layers of the ecosystem. It introduces a hierarchical smart-contract design pattern that extends the traditional factory pattern with multirole authentication and authorization mechanisms. This approach enables contracts to be organized across hierarchical levels, ensuring that roles and permissions are consistently enforced, and improving scalability, governance, and security in decentralized applications that reflect real-world organizational structures. In addition, the thesis presents a decentralized data storage and retrieval framework that integrates blockchain technology with IPFS and cryptographic techniques. By storing cryptographic references on-chain and distributing encrypted data off-chain, the proposed framework ensures immutability, traceability, availability, and confidentiality while enabling efficient verification without exposing sensitive information. This design addresses limitations in managing large data volumes and supports privacy-preserving access in distributed environments.

Scalability and data availability are further addressed through a comprehensive study of Layer 2 (L2) solutions and the development of a novel approach that leverages blob storage and adaptive multifactor scoring for rollup data. This solution balances scalability, efficiency, and security, ensuring transaction data remains accessible for independent verification while preserving trustless guarantees in high-throughput scenarios.

To address the tension between blockchain immutability and data protection requirements, this thesis proposes an integrated framework combining on-chain and off-chain components with cryptographic techniques. The framework enables sensitive data to be managed in a manner that respects user rights, such as rectification and erasure, while maintaining auditability and accountability. In parallel, a privacy-

preserving digital certification framework is presented that combines blockchain technology with zero-knowledge proofs and Merkle trees, enabling selective disclosure of certificate attributes and verification without accessing sensitive details.

Taken together, these contributions provide a comprehensive response to key limitations hindering blockchain adoption. The thesis advances smart contract governance, decentralized data management, scalability, regulatory compliance, and certification mechanisms, strengthening the role of blockchain as a trustworthy, scalable, and privacy-aware infrastructure for real-world applications.

Keywords: smart contract, design pattern, factory pattern, multirole authentication and authorization, blockchain, decentralized storage, IPFS, data traceability, layer 2, data availability, rollup, calldata, blob, digital certificate, zero knowledge proof, verification, validation

Acknowledgements

Completing this thesis has not only been an academic challenge but also a profound journey of personal and professional growth.

I am immensely grateful to my advisor, Professor Sara Migliorini, and my co-advisor, Professor Mila Dalla Preda, for their invaluable guidance and patience. I would like to express my appreciation to my advisor, whose mentorship extended far beyond standard academic supervision. She constantly challenged me to explore new and complex topics, helping me navigate the technical depth of my research with clarity. Beyond her technical guidance, she was instrumental in creating opportunities for my professional growth, always believing in my potential even when I doubted myself. I am particularly moved by her unwavering support and advocacy, ensuring that my hard work was recognized and that I had a seat at the table when it mattered most. Her dedication to my success has been the driving force behind my development as a researcher. I would also like to extend my gratitude to the administrative and technical staff of the University of Verona for their constant assistance and for ensuring that the logistical aspects of my PhD journey ran smoothly.

I am also thankful for my friends, who provided both distractions when needed and encouragement when it seemed impossible to continue. The unwavering support from my peers at the university has enriched my research experience, making this journey not only possible but also enjoyable. Their presence was pivotal in helping me navigate the complexities of this process.

Finally, I am deeply thankful to my family. Although they are miles away, their emotional support has transcended the physical distance. Their prayers, constant encouragement, and unwavering belief in my academic pursuits have been the foundation of my success. Knowing that they were cheering for me from afar gave me the strength to persevere through the most challenging moments.

This dissertation reflects not only my own efforts but also the collective support of everyone who has touched my life academically and personally. This journey has taught me the value of resilience and the importance of a supportive community in achieving these goals.

Dissemination of Work

Most of the contributions presented in this thesis have been published by the author in international peer-reviewed conferences and journals during his PhD period. For each publication, the corresponding thesis chapter(s) in which the contribution is developed are indicated.

International Conferences

Presented and published in conference proceedings

- Saif, Muhammad Bin, Sara Migliorini, and Fausto Spoto. "Blockchain-Based Multirole Authentication and Authorization in Smart Contracts with a Hierarchical Factory Pattern." 2024 6th International Conference on Blockchain Computing and Applications (BCCA). IEEE, 2024. **This work is presented in Chapter 4.**

- Saif, Muhammad Bin, Sara Migliorini, and Fausto Spoto. "Adaptive Multi-Factor Scoring in Shared Blob for Improving Data Availability in Layer 2 Blockchains" 2025 7th International Conference on Blockchain and Cryptocurrency (ICBC). IEEE, 2025. **This work is presented in Chapter 8.**

- Saif, Muhammad Bin, and Sara Migliorini. "Blockchain-based Data Immutability and Traceability in the Era of GDPR" 2025 7th International Conference on Blockchain Computing and Applications (BCCA). IEEE, 2025. **This work is presented in Chapter 7.**

Peer-Reviewed journals

Published

- Muhammad Bin Saif, Sara Migliorini, and Fausto Spoto. "Efficient and secure distributed data storage and retrieval using interplanetary file system and blockchain." Future Internet 16.3 (2024): 98. **This work is presented in Chapter 5.**

- Muhammad Bin Saif, Sara Migliorini, and Fausto Spoto. "A Survey on Data Availability in Layer 2 Blockchain Rollups: Open Challenges and Future Im-

provements." Future Internet 16.9 (2024): 315. **This work is presented in Chapter 2 and 8.**

- Muhammad Bin Saif, Sara Migliorini, and Fausto Spoto. "A Smart Contract Architecture Using Hierarchical Factory Pattern and Multirole Access Control." Journal of Network and Systems Management, vol. 34, article 84, 2026. **This work is presented in Chapter 4.**

Under review

- Muhammad Bin Saif, Clara Rodriguez Nunez, Sara Migliorini, Miguel Isabel Marquez, and Fausto Spoto. "Blockchain and Zero Knowledge Proof-Based Digital Certificate Validation and Verification." Submitted to: Blockchain: Research and Applications. Original submission: 17-10-2025; current status: under review. **This work is presented in Chapter 6.**

Declarations

Declaration of Generative AI Use: During the preparation of this thesis, the author used ChatGPT, Grammarly's generative AI writing feature, and ProWritingAid AI writing assistant exclusively for linguistic refinement, proofreading, and idiomatic corrections. The core research concepts, data analysis, and scientific conclusions are the original work of the author. The author has reviewed all AI-generated suggestions to ensure accuracy and takes full responsibility for the final content of this manuscript.

Contents

List of Figures

List of Tables

List of Abbreviations

IPFS InterPlanetary File System

ZKP Zero-Knowledge Proof

GDPR General Data Protection Regulation

PoW Proof of Work

PoS Proof of Stake

dApp Decentralized Application

L1 Layer 1

L2 Layer 2

IoT Internet of Things

RBAC Role-based Access Control

SSI Self-sovereign Identities

MAC Multirole Access Control

CP-ABE Ciphertext-Policy Attribute-Based Encryption

CID Content Identifier

EIP Ethereum Improvement Proposal

EVM Ethereum Virtual Machine

SF Super Factory

CF Client Factory

DF District Factory

ACM Access Control Manager

WMSF Water Management System Factory

TPS Transactions per Second

DO Data Owner

DR Data Requester

PRE Proxy Re-Encryption

AES Advanced Encryption Standard

KDF Key Derivation Function

GB Gigabyte

KB Kilobyte

MB Megabyte

pH Potential of hydrogen

R1CS Rank-1 Constraint System

SC Smart Contract

SD Standard Deviation

TDS Total Dissolved Solids

PK Public Key

MSK Master Secret Key

SK Session Key

AMFS Adaptive Multi-Factor Scoring

Chapter 1

Introduction

The rapid digitalization of modern society has significantly transformed how information is stored, transactions are executed, and trust is established in digital environments. Traditional centralized information systems have long served as the backbone of digital services, yet their inherent limitations are becoming increasingly evident [1]. Centralized systems often face challenges, including single points of failure, trust bottlenecks, and data availability issues. These vulnerabilities can expose organizations and individuals to risks, including service disruptions, censorship, and unauthorized data modification. Particularly, in domains where transparency, security, and reliability are essential, these shortcomings undermine confidence in digital infrastructures and highlight the need for alternative approaches [2]. Blockchain technology has emerged as a promising response to these challenges. It maintains a decentralized, tamper-resistant, and transparent ledger that is distributed across a network of nodes, thereby eliminating the need for a central authority. Its core properties, immutability, traceability, and availability, allow multiple entities to coordinate and exchange data without mutual trust. This makes blockchain technology useful across sectors such as supply chains, finance, healthcare, identity management, and other critical infrastructure [3, 4]. However, while blockchain addresses integrity, transparency, and auditability, it does not directly support data confidentiality. Data stored directly on a public blockchain is visible to all participants; therefore, the blockchain does not inherently protect sensitive information from unauthorized disclosure unless additional privacy-preserving mechanisms are adopted [5]. At the same time, the emergence of smart contracts extends blockchain capabilities beyond simple data reading and writing, enabling the automation of agreements and the implementation of decentralized application logic. Additionally, supporting technologies are crucial in enhancing the performance and scope of the blockchain ecosystem. Decentralized storage networks, such as the InterPlanetary File System (IPFS) [6], provide scalable and persistent data storage. At the same time, cryptographic techniques, such as Zero-Knowledge Proofs (ZKPs) [7], enable privacy-preserving verification and confidential data sharing.

Despite these advancements, blockchain continues to face critical limitations that hinder its widespread adoption in real-world applications. Smart contracts often lack modularity and robust access control mechanisms, exposing complex applications to governance and security risks [8]. Moreover, the scalability of current plat-

forms represents a major bottleneck, as transaction throughput remains significantly lower than in conventional centralized systems [9]. Storage is another challenge, as blockchains are not designed to handle large volumes of raw data [10]. Therefore, the use of off-chain data storage solutions is required, but integrating them securely is challenging [11]. Additionally, the principle of immutability, while central to the blockchain trust model, can conflict with legal requirements such as the General Data Protection Regulation (GDPR), which guarantees rights to rectification and erasure [12]. Finally, as already mentioned, blockchain's inherent transparency in certification and verification processes can expose sensitive information unless privacy-preserving techniques are employed [13].

These limitations and open issues altogether establish the context and motivation for this thesis. They will be described in more detail in Sect. 1.1, while Sect. 1.3 introduces the contributions provided by this thesis to address them.

1.1 Research Challenges

Blockchain technology has demonstrated significant potential; however, its adoption in real-world systems faces critical challenges. Particularly, five areas remain challenging and directly limit the broader use of blockchain. First, smart contracts lack modularity and robust access controls, hindering the development of complex applications involving multiple roles and hierarchical structures. Second, storage and off-chain data management are crucial for handling large datasets; however, existing solutions introduce risks to availability, consistency, and privacy. Third, scalability remains limited; even with Layer 2 (L2) protocols, such as rollups, the problem of reliable data availability remains unresolved. Fourth, blockchain's immutability, while essential for transparency, conflicts with regulatory requirements such as the GDPR, which demand flexibility for rectification and erasure of data. Finally, blockchain must support verifiable certification processes without exposing sensitive information, a trade-off that current approaches struggle to achieve.

Smart Contracts and Access Control

Smart contracts are a central feature of blockchain systems, as they automate agreements and enforce predefined rules without relying on intermediaries. However, the current design of smart contracts remains limited in several aspects. For instance, existing approaches often lack modularity, making it difficult to build scalable and maintainable applications. The widely used factory pattern improves reusability and reduces deployment costs by instantiating multiple copies of a contract from a single template [8]. However, it does not support the hierarchical structures that many real-world systems require. Moreover, the modular contract deployment and governance mechanisms are critical in systems where permissions flow across hierarchy levels. The traditional factory pattern does not support a hierarchical structure, and implementing role-based access control mechanisms is also challenging. As a result, contracts involving multiple roles remain vulnerable to misuse, privilege escalation, and inconsistent governance.

Addressing these challenges requires new design patterns that extend modularity to hierarchical structures while integrating multirole access control directly into contract deployment and execution. Such solutions enable families of contracts to be organized, which is how blockchain systems manage and access off-chain data. Since smart contracts often rely on external data to handle large volumes, this underscores the need for secure, flexible frameworks to support complex dApps.

Data Storage and Off-chain Data Management

Data storage represents another major challenge in blockchain systems. Blockchain guarantees immutability and traceability, but is not designed to handle large volumes of raw data. Every additional byte recorded on-chain increases storage costs and reduces throughput, making direct storage impractical for most applications [10]. Blockchain is often combined with off-chain frameworks, such as IPFS, to overcome this limitation. In this scenario, blockchain provides integrity and auditability through cryptographic references, while IPFS distributes the actual data in a scalable, peer-to-peer network [11]. However, this integration introduces several problems, such as data unavailability, as IPFS nodes may go offline or remove content, rendering data inaccessible. Data integrity is another concern, since off-chain data can be altered or lost without robust verification mechanisms. Privacy is the most critical issue, as IPFS does not enforce access control by default, making sensitive information publicly retrievable unless protected by encryption [14]. Existing approaches attempt to mitigate these issues but often compromise security or performance, or rely on complex retrieval processes that hinder usability.

Therefore, ensuring secure, reliable, and privacy-preserving off-chain storage remains an open research problem. It requires integrating encryption, hashing, and access mechanisms to maintain blockchain guarantees while still enabling efficient data retrieval. As more data is stored and secured on the blockchain, privacy concerns become increasingly crucial, particularly for certifications that contain sensitive information.

Privacy-Preserving Certification

Certification is critical in education, healthcare, supply chains, and public governance, as it verifies authenticity and compliance. Blockchain can improve these operations by ensuring that issued certificates are tamper-resistant, auditable, and permanently recorded [15]. However, transparency also introduces a critical drawback: embedding sensitive information directly on-chain, which risks public exposure, raises confidentiality concerns, and potentially leads to misuse. For instance, educational records, medical qualifications, or operational compliance data may contain personal or organizational details that should not be accessible to all participants in a blockchain network. The challenge lies in striking a balance between verifiability and privacy. Indeed, on the one hand, certificates must be auditable and resistant to forgery to maintain trust, but on the other hand, unrestricted disclosure of certificate contents can undermine privacy and security [13]. Most existing blockchain-based certification systems lack mechanisms for fine-grained control over what informa-

tion can be revealed and to whom, leading to either excessive transparency or limited usability.

A promising direction is to combine blockchain with advanced cryptographic methods, such as zero-knowledge proofs, which enable verifying certificate validity without revealing sensitive data [16]. Developing such privacy-preserving certification frameworks is an open research challenge and a prerequisite for employing blockchain in domains where trust and confidentiality are indispensable.

GDPR Regulatory Compliance

Privacy is not the only limitation to the potential of blockchain applications in data management. Another important challenge concerns the conflict between blockchain immutability and data protection regulations. Immutability guarantees transparency and auditability by preventing unauthorized modifications. However, it also makes it impossible to alter or delete information once it has been recorded, which directly contradicts the rights of rectification and erasure defined in the GDPR [12]. For instance, individuals have the legal right to request corrections or deletions of their personal data; however, the design of blockchain technology makes it difficult to execute these requests without compromising the system's integrity. Compliance with these regulations is essential in healthcare, finance, and public administration, where responsible management of sensitive information is mandatory [17]. Current approaches often attempt to resolve this conflict by storing only encrypted data or cryptographic references on-chain, while keeping the actual information off-chain. While these strategies reduce legal risks, they introduce new assumptions about the trustworthiness of off-chain storage and raise concerns about accountability [18]. As a result, no widely accepted framework currently implements blockchain immutability alongside GDPR requirements while preserving verifiability.

Addressing this concern poses a significant research challenge, as regulatory compliance is essential to large-scale adoption of solutions in privacy-sensitive environments. While these conflicts can be mitigated, as storage frameworks continue to evolve, the broader issues of scalability and throughput remain equally crucial.

Scalability and Data Availability

Scalability remains one of the most persistent barriers to blockchain adoption [19]. Public blockchains such as Bitcoin and Ethereum handle only about 7 and 15–30 transactions per second, respectively, compared to the 1,500–2,000 transactions per second supported by traditional payment systems like Visa [9]. This disparity prevents blockchains from supporting applications that require high throughput and low latency. Several solutions have been proposed at both Layer 1 (L1) and Layer 2 (L2) to address this issue. L1 approaches, such as increasing block sizes or changing consensus rules, often require disruptive changes or compromise security. In contrast, L2 solutions, such as rollups, process transactions off-chain while relying on the base chain for settlement, offering more practical scalability improvements [20]. However, these approaches introduce the problem of data availability. Even when proofs of computation are published on-chain, users still require access to the orig-

inal transaction data to verify state transitions independently. If operators withhold or censor this data, participants cannot verify the system's accuracy, thereby undermining its security [21]. Ensuring data availability entirely on-chain guarantees verifiability but imposes high storage and cost overheads. Conversely, relying on off-chain storage reduces costs but introduces more fragile trust assumptions and security risks.

Balancing scalability, efficiency, and verifiability remains an ongoing challenge, even with advancements such as blob storage and new data management structures. This challenge is particularly important because scalability determines whether blockchain can move from experimental to real use in critical infrastructures. However, even if technical performance improves, adoption is further constrained by legal and regulatory challenges, such as the conflict between blockchain immutability and compliance with frameworks such as the GDPR.

The challenges described together underscore the disparity between the theoretical strengths of blockchain and its practical application in real-world systems. Smart contracts lack robust design patterns and access control, while secure off-chain storage solutions remain incomplete. Scalability and data availability continue to limit performance, and immutability often conflicts with regulatory requirements such as the GDPR. At the same time, certification systems must strike a balance between verifiability and privacy, a goal that current approaches have yet to achieve. Addressing these open problems is crucial for transitioning blockchain from experimental prototypes to reliable infrastructures in critical domains. The next section introduces the thesis's contributions, each of which directly addresses these challenges and advances the state of the art in blockchain technologies.

1.2 Research Questions

The thesis investigates how to design blockchain-based systems to meet the demands of real-world applications. Several technical challenges currently limit real-world adoption in critical domains, such as limited smart-contract modularity, weak access-control integration, inefficient management of large off-chain datasets, privacy exposure during certification, the tension between immutability and GDPR compliance, and the data-availability challenge in Layer 2 systems. This thesis considers these challenges as connected design problems rather than isolated issues and addresses them through a dedicated set of contributions. Overall, the thesis addresses the following research questions.

RQ1. *How can smart contracts support hierarchical real-world systems while ensuring consistent multirole authorization at all levels?* The factory pattern improves reusability and reduces deployment costs, but only supports flat contract instantiation [8, 22]. Roles and permissions must propagate correctly at each level of a system that represents real-world organizational structures. There is no existing solution that combines a hierarchical factory pattern with a multirole access control mechanism tailored to this requirement [23, 24].

RQ2. *How can blockchain and off-chain storage be used to support the secure, traceable, and privacy-aware management of large data?* Storing raw data directly on-chain is challenging at scale due to cost and throughput constraints [10, 11]. While integrating blockchain with IPFS addresses storage scalability, this combination introduces risks related to data unavailability, integrity violations, and unauthorized access [14]. In particular, existing solutions rely on decryption during retrieval, thereby exposing private keys and introducing security risks [25].

RQ3. *How can blockchain-based certification systems enable verification and validation without disclosing sensitive certificate attributes?* Blockchain offers tamper-resistance and auditability for digital certificates [26]; its transparency makes sensitive information accessible to all network members. ZKPs can verify a statement without revealing the underlying data [7]. However, no existing solution integrates Merkle tree-based selective disclosure with on-chain ZKP verification and revocation management in a single framework.

RQ4. *How can blockchain design support GDPR regulations, specifically rectification and erasure, without compromising auditability and accountability?* Blockchain immutability prevents changes or deletions to data, thereby conflicting with GDPR data subject rights [18]. Current techniques keep sensitive data off-chain to limit legal exposure, but this pushes compliance guarantees beyond the blockchain and diminishes accountability [27, 28]. Currently, there is no on-chain verifiable framework that addresses both constraints.

RQ5. *How can L2 solution improve data-availability, particularly for small rollups, while preserving independent verification and trustless guarantees?* Rollups increase throughput by executing transactions off-chain, but transaction data must be posted on L1 so participants can verify state transitions and detect fraud [21, 29]. EIP-4844 introduces blob storage, reducing costs compared to calldata. Small rollups still suffer from inefficient storage usage due to fixed blob sizes and the lack of adaptive allocation strategies [30, 31].

These five research questions define the scope and direction of the thesis. Each of these questions is addressed by a dedicated contribution introduced in Sect. 1.3. The following section discusses these contributions and describes how they collectively advance the design of blockchain systems across multiple layers of the ecosystem.

1.3 Thesis Contributions

This thesis presents five key contributions that address the research challenges identified in the previous section. Each contribution is developed as a dedicated chapter, focusing on a specific aspect of blockchain adoption: smart contract design, decentralized storage, scalability, regulatory compliance, and digital certification. These contributions establish a clear research path that enhances blockchain's role as a trustworthy, privacy-preserving, and legally compliant infrastructure for critical applications. The following subsections sequentially introduce these contributions, each building on the last to address different layers of the blockchain ecosystem.

Smart Contract Design Pattern and Access Control

This thesis introduces in Chap. 4 a new design pattern for smart contracts that extends the traditional factory pattern into a hierarchical factory pattern with multi-role access control. The proposed pattern addresses the limitations of existing approaches, which typically allow only flat, repetitive contract instantiations and offer minimal support for complex authorization. In contrast, the hierarchical factory pattern enables contracts to be organized in multiple levels, reflecting the structure of real-world organizations, while ensuring that roles and permissions are consistently managed across the hierarchy. This contribution also integrates a multi-role authentication and authorization mechanism tailored for decentralized environments. The solution defines roles and permissions directly inside the contract structure and enforces them dynamically as contracts are created and interacted with at different levels. This guarantees that only authorized entities can perform sensitive operations, reducing the risks of unauthorized access or governance failures.

The proposed design pattern significantly enhances the scalability, adaptability, and reliability of dApps by integrating hierarchical smart contracts and secure access control mechanisms. It provides a robust framework for real-world use cases that require enforcing different roles and responsibilities, such as in critical infrastructure, supply chains, and identity management systems. While enhancing the governance of smart contracts improves the logic layer of blockchain applications, the following contribution addresses the equally significant challenge of how these applications manage and store extensive volumes of data.

Privacy-Preserving Decentralized Data Storage and Retrieval

To address the limitations of blockchain for managing large volumes of data, this thesis in Chap. 5 designs and develops a decentralized storage framework that integrates blockchain, IPFS, and encryption. As previously discussed, while blockchain guarantees immutability and traceability, it is not suitable for storing raw information at scale due to its high costs and limited throughput. In the proposed approach, only cryptographic references are stored on-chain, while the actual data is distributed across IPFS nodes, providing a more scalable storage architecture. However, using IPFS alone does not guarantee confidentiality or controlled access. To overcome this challenge, the framework integrates encryption and hashing mechanisms that preserve privacy and integrity. Encryption protects sensitive data stored off-chain, while hashing ensures tamper detection without exposing the original content. This dual-layer approach maintains confidentiality during retrieval and querying while enabling transparent verification through blockchain. By integrating blockchain, IPFS, and encryption, this contribution provides a practical solution that ensures immutability, availability, traceability, and privacy in distributed storage systems.

This framework offers a practical solution for real-world scenarios such as Internet of Things (IoT) environments, healthcare systems, and supply chain applications, where large data volumes must be stored securely and accessed reliably. While this contribution enhances blockchain's data management capabilities, it also highlights another challenge related to the privacy and the protection of sensitive information, which is the focus of the next contribution.

Privacy-Preserving Digital Certificate Verification and Validation

The final contribution of this thesis is a framework for digital certification that combines blockchain with zero-knowledge proofs to achieve both verifiability and privacy. Certification processes require that issued documents be tamper-proof and easily verifiable, but they can also involve sensitive information that cannot always be disclosed to the public. Existing blockchain-based certification systems enhance transparency but often fall short in adequately protecting private data. The proposed approach integrates smart contracts, Merkle trees, and ZKPs to enable selective disclosure of certificate attributes. This allows verifiers to confirm a certificate's authenticity and compliance without accessing sensitive details, thereby preserving privacy while maintaining trust. By anchoring certificates on the blockchain, the system guarantees immutability and availability, while cryptographic proofs ensure that only the necessary information is revealed during verification.

This contribution presents a privacy-preserving solution for digital certification that is applicable across multiple domains, including education, healthcare, supply chains, and public governance. It enhances confidence in certification processes while addressing one of the most pressing concerns in blockchain adoption: balancing transparency and privacy. However, besides privacy concerns, blockchain adoption is also limited by legal and regulatory issues, most notably the conflict between immutability and compliance requirements, such as the GDPR.

Blockchain Privacy and Compliance for GDPR

To address the immutability of blockchain technology in relation to data protection requirements, this thesis proposes, in Chap. 7, a framework that integrates on-chain and off-chain solutions, leveraging cryptography, to support compliance with the GDPR. While immutability is essential to guarantee traceability and auditability, it directly contradicts certain rights, such as the right to rectification and erasure required by law. This hinders the widespread adoption of blockchain in sectors where data protection compliance is essential, highlighting a crucial barrier to its potential. To mitigate this, the thesis proposes a framework integrating blockchain with off-chain storage, encryption, and smart contracts to support GDPR rights without compromising the system's integrity. The framework ensures sensitive data is managed in a manner that respects user rights while maintaining the audit trail and accountability features that make blockchain technology valuable. By combining on-chain and off-chain elements, it achieves a balance between legal compliance and technological trustworthiness. The contribution demonstrates that blockchain systems can be designed to align with data protection requirements while preserving decentralization and verifiability. It offers a solution for integrating blockchain in healthcare, finance, and public administration sectors, where transparency and compliance are equally critical.

The last contribution of this thesis deals with the need to increase transaction throughput while reducing transaction costs. This issue is particularly important in the context where frequent interactions are necessary to post and secure information.

L2 Blockchain Scalability and Data Availability

Scalability and data availability are addressed in this Chap. 8 through a comprehensive study of L2 solutions and a solution to improve data availability. While rollups and other L2 protocols significantly improve throughput by executing transactions off-chain, they introduce the problem of ensuring that transaction data remains accessible for independent verification. Without reliable data availability, users cannot reconstruct the system state or detect fraudulent behavior, undermining the blockchain's trustless nature. This thesis surveys existing approaches to data availability in rollup-based architectures and identifies their advantages and limitations. Building on this analysis, it introduces novel solutions that leverage techniques such as blob storage and adaptive scoring mechanisms to reduce costs while maintaining transparency and verifiability. This approach aims to strike a balance between scalability, efficiency, and security, ensuring that L2 solutions can support high-volume applications without compromising the trustless guarantees of blockchain. This can help facilitate the broader adoption of blockchain, particularly in performance-critical environments.

The described contributions offer a comprehensive response to the research challenges outlined earlier. They advance the design of smart contracts with secure access control, privacy-preserving decentralized storage, improve scalability and data availability in L2 solutions, reconcile blockchain immutability with data protection requirements, and establish a verifiable yet privacy-preserving certification framework. Although these contributions are developed and evaluated within blockchain systems, their underlying design principles connect to broader paradigms of decentralized computation and verifiable data processing, where correctness, auditability, privacy, and efficient access to distributed data must coexist across on-chain and off-chain components. By addressing these issues coherently, the thesis strengthens the role of blockchain not only as a trustworthy, scalable, and legally compliant technology for critical applications, but also as a concrete foundation for broader, decentralized, and privacy-aware digital infrastructures.

Fig. 1.1 summarizes how the five contributions of this thesis are organized across the blockchain stack and which design dimensions each one addresses, while the following section outlines the thesis structure and describes how each contribution is developed in the subsequent chapters.

1.4 Thesis Overview

The remainder of this thesis is organized as follows. Chap. 2 provides the necessary background on blockchain technology, smart contracts, decentralized storage, scalability solutions, privacy-preserving cryptographic methods, and data protection regulations. It establishes the conceptual and technical foundation required for the contributions. Chap. 3 provides a literature review of the major results already available in the literature about the addressed topics. Chap. 4 introduces the proposed design pattern for smart contracts, which combines a hierarchical factory model with multi-role authentication and authorization. Chap. 5 presents the framework for decentralized and privacy-preserving data storage and retrieval, which integrates blockchain,

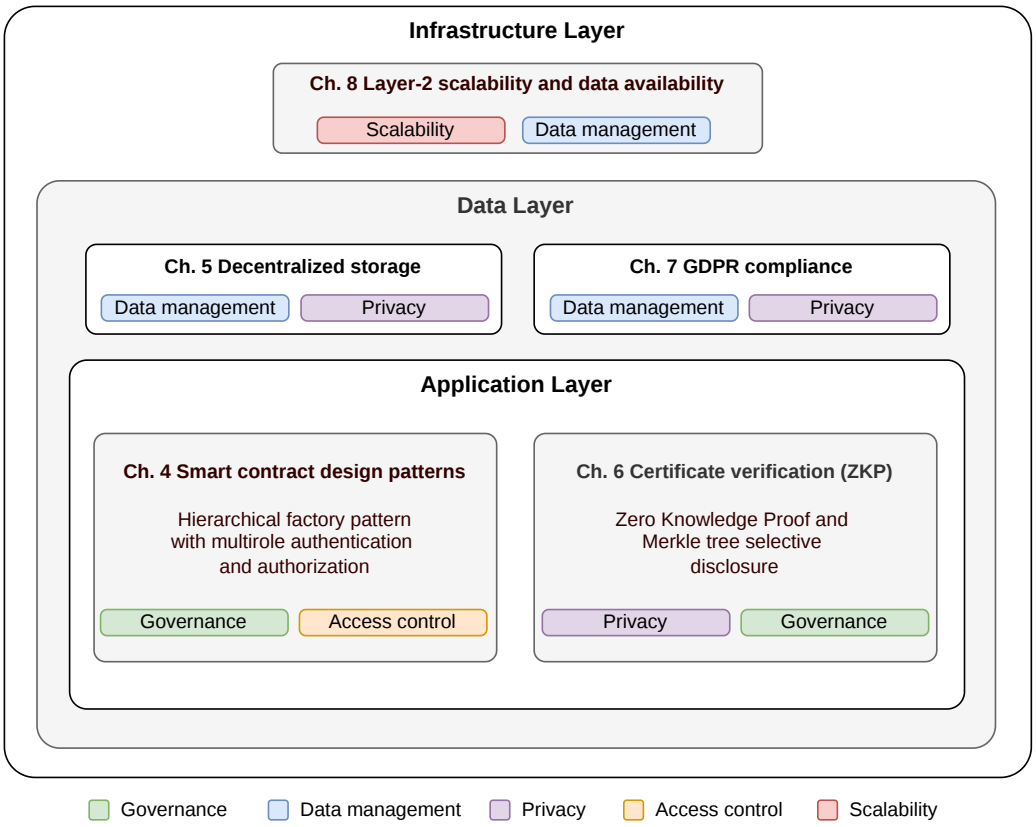


Figure 1.1: Conceptual organization of the thesis contributions across the application, data, and infrastructure layers of blockchain system design.

IPFS, and encryption. Chap. 6 develops a blockchain-based certification system that leverages ZKPs to provide verifiability while preserving sensitive information. Chap. 7 addresses the conflict between blockchain immutability and GDPR requirements by proposing an integrated framework for privacy and compliance. Chap. 8 focuses on scalability and data availability in L2 solutions, including a survey of existing approaches and the development of novel blob-based mechanisms. Finally, Chap. 9 concludes the thesis by summarizing the contributions, highlighting their broader implications, and outlining directions for future research.

Chapter 2

Background

This chapter provides a comprehensive overview of blockchain technology, with a particular focus on the aspects covered in this thesis, including smart contracts, the underlying principles of L2 solutions, and the concept of ZKPs. Sect. 2.1 introduces the evolution of blockchain technology, detailing its consensus mechanisms and the concept of eventual consistency. Sect. 2.2 defines smart contracts, explains their lifecycle and deployment, and shows how they induce state transitions within the network. Sect. 2.3 and Sect. 2.4 address the scalability limitations of blockchain by examining L2 solutions and provide the necessary background for the specific challenge of data availability, which will be discussed extensively in Chap. 8. These sections provide the essential background on the general functioning and architecture of rollups to facilitate a deeper understanding of the problem. Finally, Sect. 2.4.3 introduces ZKPs and describes their role in enabling privacy-preserving verification. These concepts provide the background required for the architectures, evaluations, and implementations developed in the thesis.

2.1 Blockchain Technology

Blockchain technology originated in 2009 with the introduction of Bitcoin [32], which was designed to address the problem of trust in digital transactions. In particular, it provides a solution to the double-spending problem in a fully decentralized environment, which was considered unsolvable for decades.

Traditional financial and information systems rely on centralized intermediaries, such as banks or service providers, to validate and record operations [33]. Although effective, these centralized structures introduced bottlenecks of trust, single points of failure, and systemic risks, as the availability and integrity of the entire system depended on a single authority. Moreover, centralized systems are often vulnerable to censorship, data manipulation, and security breaches. Blockchain emerged as an alternative model in which trust is not placed in a central institution but distributed across a network of independent participants. By embedding trust in cryptographic protocols and consensus mechanisms, blockchain substitutes institutional trust with algorithmic trust, enabling secure coordination in decentralized environments.

Blockchain is a distributed, append-only ledger that records transactions across a network of peer-to-peer nodes, eliminating the need for a central authority [34]. The

information in this ledger is organized into blocks, each containing a list of transactions, a timestamp, and a cryptographic hash linking it to the previous block. This design creates a continuous chain of records, where any attempt to alter past data invalidates all subsequent blocks, making data tampering computationally infeasible. The ledger is replicated across multiple nodes, ensuring redundancy, transparency, and fault tolerance, such that the failure or compromise of a single participant does not undermine the system's integrity. In this way, blockchain establishes a shared, verifiable source of truth accessible to all network participants.

At a high level, blockchain operates through interconnected nodes that create a decentralized, secure system. Transactions constitute the fundamental units of information that record digital operations such as payments, asset transfers, and contract executions. These transactions are packed into blocks, which serve as containers that include metadata such as timestamps and cryptographic references to preceding blocks. Blocks are maintained and propagated by a network of participating nodes. Full nodes can store and verify the entire block, while lightweight clients store only partial information but can still interact securely with the system. The cryptographic techniques, including hashing functions that connect blocks and digital signatures that verify transactions, guarantee the integrity of the blockchain system. Consensus protocols coordinate nodes to agree on the validity of blocks, ensuring that all participants agree on a consistent blockchain state. In this way, blockchain achieves consistency and trust without the need for oversight by a central authority.

In its essence, consensus is the set of rules that enables blockchain participants to agree on the current state of the ledger without relying on a central authority. In other words, it aims to guarantee that all honest nodes maintain a consistent view of the blockchain, even in the presence of dishonest or malicious actors [35]. Different families of consensus mechanisms have been developed, each with its own trade-offs. Proof of Work (PoW) relies on solving computationally intensive puzzles to validate new blocks, offering robustness but incurring significant energy costs [36]. Proof of Stake (PoS) assigns block creation rights proportionally to the amount of tokens staked by validators, significantly reducing energy consumption while maintaining security [37]. Byzantine Fault Tolerant protocols achieve agreement by requiring a majority of nodes to validate blocks through direct communication, offering faster finality but with scalability limitations [38]. Despite the differences, all consensus mechanisms aim to maintain trust, prevent double-spending, and ensure that blocks remain part of an immutable and shared history once confirmed.

The operation of blockchain can be better understood by following the lifecycle of a transaction, as shown in Fig. 2.1. It begins when a user creates a transaction and broadcasts it to the network, where participating nodes temporarily store all transactions [39]. Each transaction is digitally signed, enabling nodes to verify its authenticity and validity in accordance with the protocol. Designated nodes collect valid transactions into a candidate block, for instance, miners in PoW or validators in PoS consensus. The candidate block is proposed to the network and subjected to the consensus process, which ensures that the majority of honest participants agree on its validity. Once the block is accepted, it is appended to the existing chain and propagated across the network, where every node updates its copy of the ledger. At this stage, the status of transactions in the block changes to confirmed, and depend-

ing on the consensus protocol, finality is either immediate or achieved after adding several subsequent blocks. This process ensures that all participants maintain a consistent, verifiable record of events without relying on a central authority, while inherently adopting the notion of eventual consistency [40]. In contrast to traditional centralized databases that enforce strong consistency, blockchain systems operate in a decentralized, asynchronous environment where information propagates gradually across the network. As a result, different nodes may temporarily observe slightly different views of the ledger due to network latency or the presence of short-lived forks. Eventual consistency guarantees that as the consensus protocol continues to operate, these temporary differences are gradually resolved and all honest nodes converge to the same global state. Over time, the likelihood of the ledger being reorganized becomes very small, so participants agree on a single, canonical, and practically immutable history.

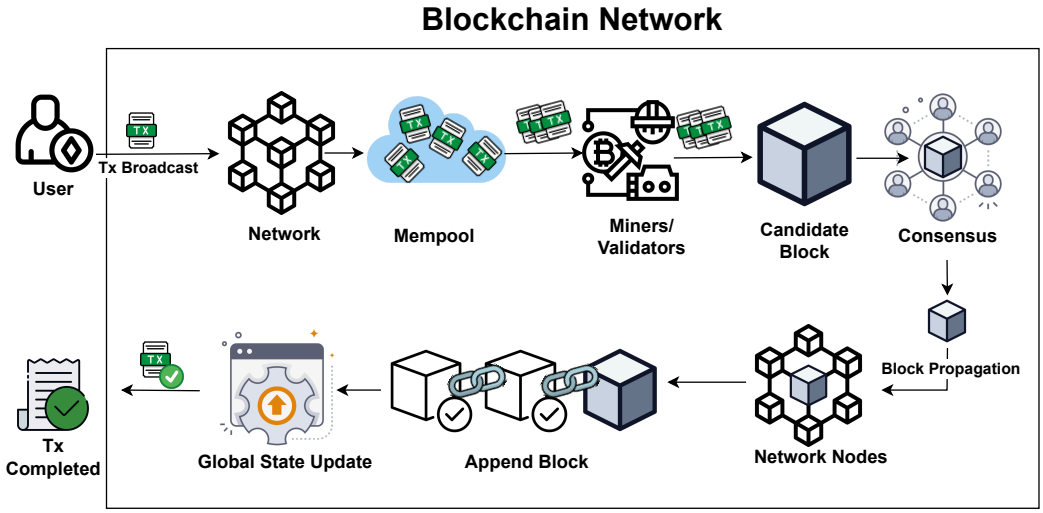


Figure 2.1: Blockchain transaction lifecycle.

The distinctive properties of blockchain lay the foundation for its application across diverse domains. Immutability ensures that once data is stored, it cannot be altered or deleted without detection, creating a permanent and tamper-evident audit trail [41]. Traceability enables the complete history of a transaction or asset to be reconstructed, particularly useful in areas such as supply chains and healthcare, where origin and accountability are critical [42]. Availability is guaranteed by replicating the ledger across many nodes, providing resilience against failures or malicious attacks, and ensuring data accessibility even when some network nodes are disrupted [43]. Finally, decentralization eliminates the need for intermediaries and reduces reliance on a single authority, thus protecting against censorship and centralized control [2]. These properties provide transparency, trust, and reliability, making blockchain an attractive solution for systems that require secure and verifiable data management.

2.2 Smart Contracts

A major extension of blockchain beyond simple transaction recording is the introduction of smart contracts, which were practically introduced and popularized by Ethereum [44], transforming blockchain from a distributed ledger for payments into a general-purpose computational platform. A smart contract is a self-executing program deployed on the blockchain that automatically enforces predefined rules and agreements once specific conditions are met. Unlike traditional contracts, which rely on intermediaries for validation and enforcement, smart contracts operate in a decentralized and trustless environment where the logic is transparent, verifiable, and tamper-resistant [45].

A smart contract is deployed by submitting its compiled bytecode to the blockchain through a transaction, after which it is assigned a unique contract address and becomes part of the immutable ledger. Once deployed, interaction with the contract occurs by submitting transactions to its address, which include a data payload specifying the invoked function and its input arguments; in addition, read-only calls can be used to query contract data without modifying the ledger. Each state-changing invocation triggers execution by the virtual machine and produces a deterministic state transition that is validated and replicated across the network. The smart contract extends the definition of blockchain state beyond token ownership alone. In first-generation blockchains such as Bitcoin, the state is primarily financial and can be described in terms of token ownership, whereas smart contract platforms introduce a broader system-wide state that also includes the persistent storage of all deployed contracts. The global system state includes not only account balances but also the stored values maintained by smart contracts, such as variables, mappings, and application-specific data structures, together with the rules that control how they can be updated. Therefore, a state transition in smart contract platforms is not only a transfer of value but also the deterministic execution of contract logic that updates the shared on-chain storage. This capability enables developers to build Decentralized Applications (dApps) that automate processes, reduce operational costs, and mitigate the risks of fraud or manipulation.

Smart contracts are increasingly applied in various domains, including finance [46], healthcare [47], supply chains [48], and critical infrastructure [4], facilitating reliable interactions among multiple parties without the need for centralized authority. Programmable logic atop immutable ledgers enables smart contracts to expand the role of blockchain from a data management system into a general-purpose platform for decentralized computation. In supply chain management, blockchain enables transparent tracking of goods, ensuring origin and reducing fraud. It supports secure handling of medical records and controlled data sharing in healthcare. For critical infrastructure such as energy and water systems, blockchain provides tamper-proof logging and secure coordination of distributed operations. In the field of digital identity, it offers decentralized authentication systems that reduce reliance on central authorities and improve user control over personal data. Blockchain also plays an important role in preserving cultural heritage, ensuring the authenticity of restoration processes and archival materials [49]. Finally, in certification systems, blockchain guarantees the validity of issued certificates and, when integrated with ZKPs, en-

ables privacy-preserving verification [50]. These diverse applications demonstrate the versatility of blockchain as a foundation for trustworthy, decentralized systems across multiple domains.

2.3 Layer 2 Solutions

This section provides a brief introduction and comparison of the available L2 solutions, highlighting their key differences. L2 solutions have been introduced to speed up existing blockchains, particularly Ethereum, by increasing their transaction throughput and reducing scalability concerns. The main idea behind an L2 solution is to move most of the computation and transactions off the main L1 blockchain while preserving its security guarantees, using it as a final settlement and dispute-resolution mechanism. This behavior is particularly useful in the context of smart contracts, where processing a transaction may involve executing complex functionality and require substantial resources. At present, four main families of L2 solutions have been developed, which are compared in Tab. 2.1 in terms of transaction throughput, latency, security, and trust assumption, and are better discussed in the remainder of this section.

State channels [51] is an offline solution that performs micro-payments outside the main chain via an established secure channel and aggregates them into a single transaction on the main chain at the end of the payments. The transactions with state channels are performed off-chain, which requires mutual trust, agreement, and secure communication among participants. The vulnerability arises if one party goes offline or fails to end the channel by broadcasting the final state to the blockchain within the specified time window. In this case, a malicious actor might take advantage by submitting an outdated state or withholding the final state, resulting in an incorrect settlement of funds [52]. Therefore, this solution requires trust and the use of secure channels among participants.

Sidechains [53] consists of the use of a secondary chain that is created as an attachment of the primary (main) one to allow the transfer of assets from the main chain to the secondary one at predetermined rates. The secondary chain is closely related to the main one, but the two remain separate and can use different consensus protocols. Thus, no security issues are transferred between the two chains. Therefore, the integrity and security of a sidechain solution depend solely on the characteristics of the secondary chains, as the L1 protocol does not guarantee security.

Plasma [54] is a specific sidechain for the Ethereum network that uses a smart contract as its core. Specifically, transactions are processed on the sidechain to alleviate the load on the Ethereum network. In contrast, the block headers of the Plasma sidechain are posted on the main chain periodically for verification. Plasma chains operate in an optimistic manner, assuming transactions are correct until an agent, known as a watcher, observes a fraudulent transaction and initiates dispute resolution. The original data is not published on the main chain. Thus, there is a need for trust in the Plasma operator and for at least one watcher to be available at all times to ensure that transactions published on L1 are not fraudulent.

Rollups are similar to plasma, since transaction execution is shifted to L2, but all data derived from that execution is published back to the main chain [55], allow-

Table 2.1: Comparison of L2 blockchain solutions. The reported throughput and latency are approximate and theoretical, and may vary depending on the specific implementation and network conditions.

Solution	**Throughput (TPS)**	**Latency (sec)**	**Security Level**	**Trust Assumptions**
State Channels	Very High (1000 - 10000)	Low (0.01 - 0.10)	High	Requires trust in participants
Sidechains	High (100 - 1000)	Moderate (10 - 60)	Varies	Depends on sidechain's security mechanism
Plasma	High (500 - 5000)	Low to Moderate (0.1 - 1)	High	Trust in Plasma operator for data availability
Optimistic Rollups	Moderate to High (4000 - 10000)	High (0.1 - 7 days due to fraud-proof period)	High	Trust in fraud-proof mechanism
ZK-Rollups	High (2000 - 4500)	Low (0.1 - 1)	Very High	No additional trust assumptions
Validiums	Very High (9000)	Low (0.1)	High	Trust in data availability committee

ing the L1 blockchain to host the transactions processed by the L2. Barry Whitehat [56] introduced the concept of rollup in 2018; since then, rollups have emerged as a robust solution to overcome scalability challenges, allowing transactions to be processed off-chain while retaining the security and decentralization of the underlying L1 blockchain. *ZK-rollups* and *optimistic rollups* are the two primary types of rollup-based L2 solutions. In particular, zk-rollup is an off-chain solution that uses zero-knowledge (zk) proofs to bundle multiple transactions into a single light transaction and post it back to the main chain, also providing evidence of transaction validity. Conversely, an optimistic rollup eliminates the need for a zk-proof, reducing computational intensity at the expense of additional trust assumptions.

Finally, *validiums* [57] are novel systems similar to rollups, but differ in their data availability mechanism. In contrast to rollups, in validiums, transaction data and a state root hash are stored off-chain, while validity proofs are stored on-chain. A data availability committee, such as a centralized oracle that needs to be trusted, can regulate the availability of off-chain data.

This thesis focuses on rollup-based L2 architectures, as they offer an optimal balance between efficiency and security (without additional trust assumptions). The following section discusses the number of promising solutions developed to date.

2.4 Overview of L2 Rollups

The main idea behind rollups is to process transactions off-chain and then store the results of that processing back on the main L1 chain. In this case, rollups offload smart contract computational effort and reduce storage requirements on the main blockchain network. This allows the network to scale while inheriting the security from the underlying L1 blockchain.

This section provides a comprehensive overview of the components of rollups, their execution interactions, and the different types of rollups considered, with the aim of better analyzing the data availability issue in Chap. 8.

2.4.1 Components and Workflow of Rollups

An L2 rollup-based solution consists of multiple components that allow transactions to be executed off-chain and the results posted on-chain. Fig. 2.2 and Fig. 2.3 respectively illustrate the overall architectures of optimistic rollups and zero-knowledge rollups. The nature of each of these components can depend on the specific implementation that is considered, but they can be summarized in the following categories:

User – User initiates the transaction by sending a transaction request to a rollup sequencer. This step is similar to submitting a normal blockchain transaction, but in this case, the transaction is directed to a specific rollup protocol instead of an L1 solution.

Sequencer – This is one of the most important components in rollups. The sequencer collects the user-initiated transactions and sequences them in order. This is a critical step to mitigate the risk of fraudulent practices, such as a front-running attack in which a malicious actor can execute a transaction before other pending ones to take financial advantage. The sequencer ensures that all transactions are processed in the correct order and error-free.

Aggregator – This is another crucial rollup component. The aggregator receives transactions from users, executes them off-chain, and then rolls the results into a single batch. This last activity is where the name rollup is derived. The final step for the aggregator is to post the batched transactions and the resulting blockchain state as Merkle root to the smart contract deployed on the L1 main chain.

On-chain Smart Contract – L2 solutions are based on the presence of a smart contract deployed on the L1 main net. Such a contract serves as a foundation for off-chain rollup activities. In particular, it verifies and monitors the state transition performed off-chain by linking off-chain rollup executions and on-chain data integrity.

Dispute Resolution (specific to Optimistic Rollups) – Unlike ZK-rollups, optimistic rollups consider every transaction valid and do not submit the validity proofs on-chain, which is why it is called optimistic. Optimistic rollups define a challenge period in which anyone can challenge a transaction and create a dispute before finalizing the transactions on the main chain. In this case, a *fraud proof* is provided to the final user to demonstrate the correctness of the data posted on L1.

Prover (specific to ZK-Rollups) – The prover component is specific to ZK-rollups. It generates cryptographic proof using ZK-SNARK or ZK-STARK for off-chain computation. This proof is then posted on-chain to verify that the computation was performed correctly and that the state transition followed the predefined rules, without revealing the underlying data or the exact computational steps. The prover also stores the transaction data for availability on L1. Therefore, anyone can reconstruct the current state from transaction data and independently verify it on-chain.

Data Availability – Transaction data is critical for ensuring the integrity and security of rollups. This data is posted to the L1 blockchain, making it available and accessible to everyone. Data availability ensures that any participant can independently reconstruct and validate the current state of the rollup by accessing data on the L1 blockchain. Different L2 rollup-based solutions can implement data availability differently, as will be discussed in more detail in Chap. 8.

The following two subsections provide a more detailed comparison of optimistic and ZK-rollups, discussing the information stored in each case. A summarized comparison of ZK-rollups and optimistic rollups is also presented in Tab. 2.2.

Table 2.2: Comparison of Optimistic and ZK-Rollups

Feature	**Optimistic Rollups**	**ZK-Rollups**
Verification Method	Utilizes fraud-proofs, which require a challenge period to contest transactions.	Uses zero-knowledge proofs to validate transactions, ensuring immediate verification without contestation.
Data Handling	Stores full transaction data on-chain to facilitate potential fraud proofs and challenges.	Posts only transaction proofs and state differences on-chain, reducing data load.
Withdrawal Delay	Withdrawals can take up to one week due to the required fraud-proof window.	Enables instant withdrawals due to the immediate finality of transactions.
Transaction Finality	Transaction finality is delayed, pending the expiration of the fraud-proof challenge period.	Transaction finality is immediate as cryptographic proofs confirm validity at the time of transaction posting.
Security Assumption	Operates under the assumption that validators are honest and will actively challenge incorrect transactions during the dispute period.	Relies on the mathematical validity of zero-knowledge proofs, assuming accuracy in proof generation.
Developer Complexity	Implementation is less complex than ZK Rollups, but still requires mechanisms to handle disputes and fraud-proof.	Typically involves more complex cryptographic operations, increasing development complexity because of advanced mathematical concepts.

2.4.2 Optimistic Rollups

Optimistic rollups assume that all transactions are valid by default unless challenged, which gives the name *optimistic*. In this approach, L2 verifies all transactions and computes the new Merkle root of the state, ensuring the integrity of the rollup. However, transactions and state transitions are verified on L1 using fraud proofs only if a challenge is raised during the *dispute period*. This approach significantly reduces the on-chain computation and transaction verification on L1. However, it introduces an inevitable challenge period in which transactions can be disputed, potentially taking up to a week. Therefore, transaction finality latency is a primary challenge for optimistic rollups. This can be a critical issue in applications requiring immediate transaction finality. Furthermore, the security of optimistic rollups relies on an actively participating network of nodes that can detect and challenge fraudulent transactions, therefore relying greatly on network integrity and node honesty. With reference to Fig. 2.2, the fraud proofs are maintained in the L2 blockchain, while only the rolled-up data are posted back on the L1 one.

2.4.3 ZK-Rollups

Zk-rollup solutions primarily use zk-proofs to secure computations on the L2 network and provide a guarantee of their correctness.

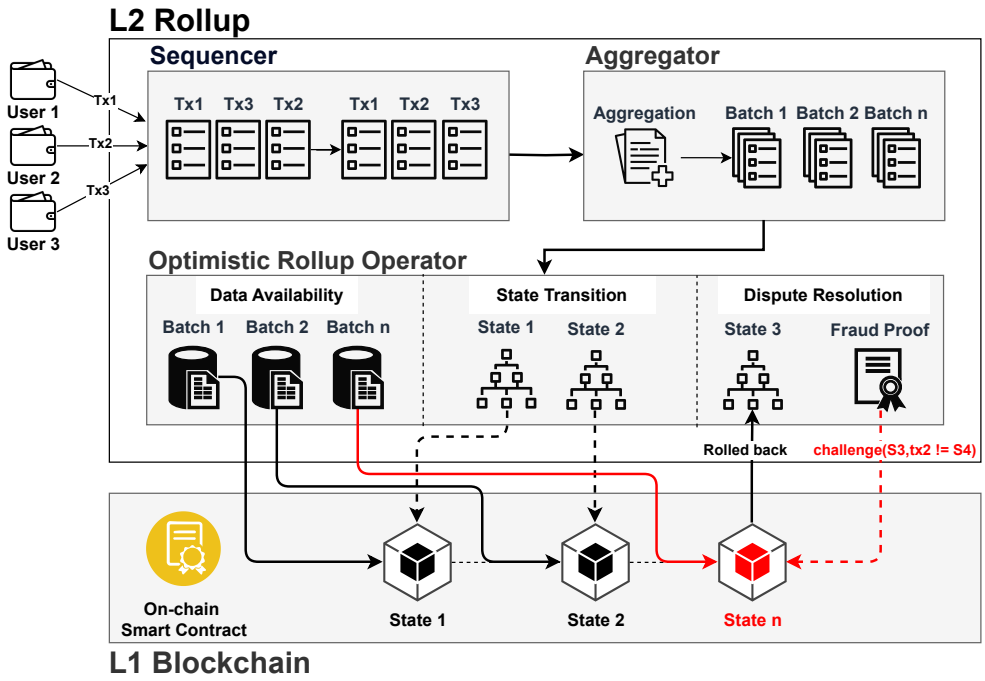


Figure 2.2: General architecture of Optimistic Rollup.

Zero-Knowledge Proofs

In cryptography, a zk-proof is a method by which one party (the prover) can demonstrate to another party (the verifier) that some given statement is true while still avoiding revealing to the verifier any additional information beyond the mere fact of that statement's truth [58]. This unique property makes ZKPs a pivotal technology for privacy-preserving blockchain applications, enabling the verification of data integrity and compliance without revealing sensitive underlying information [59]. Fundamentally, a ZKP protocol must satisfy three core properties:

Completeness – If the statement is true, an honest prover will always be able to convince the verifier of this fact.

Soundness – If the statement is false, no dishonest prover can convince the verifier that it is true, except with a negligible probability.

Zero-Knowledge – if the statement is true, the verifier learns nothing other than the fact that the statement is true; specifically, they do not learn the secret information (often called the witness) used to generate the proof.

A zk-proof is often formulated by representing the statement to be proven as a computation that can be verified using a system of constraints. The computation is represented as a zk-circuit, in which the desired relationship between inputs and outputs is encoded as constraints that must be satisfied [60]. The private values required to satisfy these constraints are considered as a *witness*, while any values that must be constrained by the public claim are presented as *public inputs*. In practice, the witness corresponds to a complete assignment of all circuit signals that satisfies every constraint, including those derived from secret inputs. To implement such circuits

in a structured and reusable way, domain-specific languages have been developed, such as Circom [61], which is widely used to define circuits by specifying signals and constraints and by composing circuit templates into larger constructions. The circuit is then compiled into a Rank-1 Constraint System (R1CS), which provides a formal foundation for proof generation and verification [62].

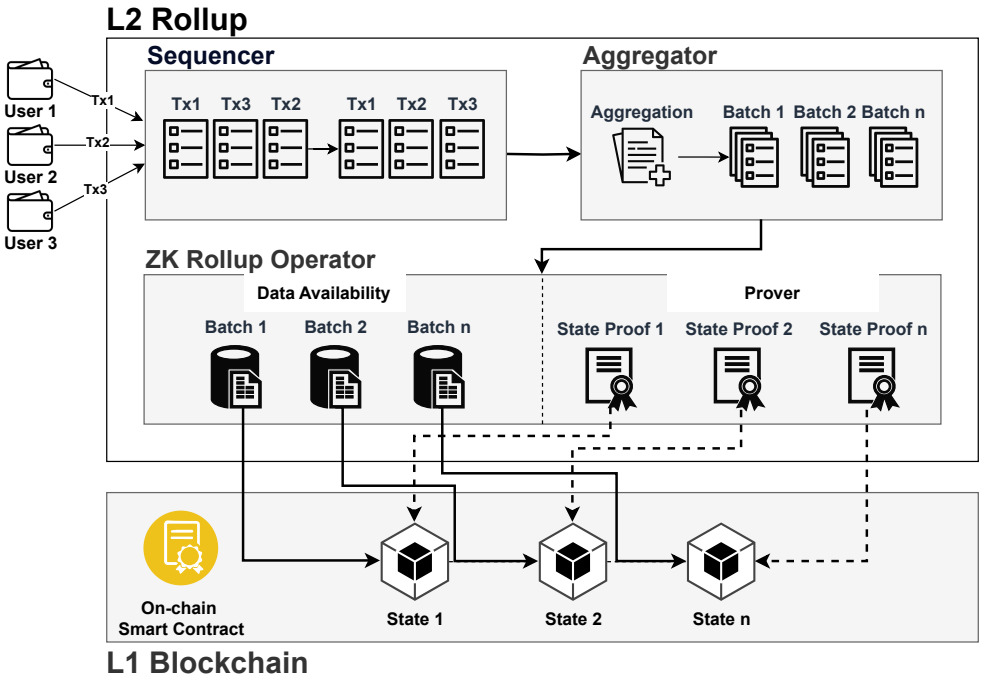


Figure 2.3: General architecture of Zk-Rollup.

After the circuit has been defined and compiled, a proof is constructed by executing a proving algorithm that takes as input the compiled constraints, the public inputs, and a satisfying witness. This process relies on cryptographic parameters, often referred to as a proving key, which are typically derived during an initial setup phase of the proving system [62]. The proving step is computationally expensive and is therefore usually performed off-chain, whereas verification is designed to be substantially cheaper. The result is a succinct cryptographic proof that can be transmitted and verified independently of the original private inputs, together with verification parameters associated with the circuit. In blockchain-based systems, these verification parameters can be used to generate a verifier smart contract, commonly implemented in Solidity using supporting tools such as snarkjs [63]. Once deployed on-chain, the verifier contract exposes a verification function that allows the blockchain to validate submitted proofs against the corresponding public inputs. In this way, the blockchain can act as a neutral verification layer, confirming the correctness of a statement without revealing the private witness, while preserving the standard execution and state-transition model of smart contracts.

Using ZK-proofs, ZK-rollups enhance scalability and privacy by allowing off-chain nodes to prove transaction validity without revealing any details. The results of transaction executions are compressed into a single batch and stored in the L1

blockchain, along with a cryptographic proof of the state transition, as illustrated in Fig. 2.3. In this case, an on-chain verifier smart contract verifies the transactions and updates the blockchain state. This ensures instant transaction finality without a dispute period.

The two main types of ZKPs used in ZK-rollups are Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARKs) and Zero-Knowledge Scalable Transparent ARguments of Knowledge (zk-STARKs). The following two subsections provide some details about these two alternatives.

Zk-SNARKs

The name zk-SNARK stands for Zero-Knowledge Succinct Non-Interactive Argument of Knowledge [64]. As the name suggests, the proof size of zk-SNARK is very compact, regardless of the complexity of the statement. This makes such solutions widely adopted in currently available L2 projects due to their efficiency and their small proof size relative to the length of the actual computation. Moreover, the non-interactive nature of zk-SNARKs eliminates the need for continuous communication between the prover and the verifier during proof generation. However, zk-SNARK requires a one-time trusted setup for generating proof parameters, which can pose a security risk.

Formally, a zk-SNARK protocol consists of three steps: Setup (***S***), Prove (***P***), and Verify (***V***), as defined below:

- **Setup *S*** – The first step involves the key generation, which takes a secret parameter λ also known as toxic waste, to generate a proving key p and a verification key v.

$$(v, p) \longleftarrow S(1^{\lambda}) \tag{2.1}$$

  This phase relies on elliptic curve pairings and requires a trusted environment.

- **Prove *P*** – The prover generates a proof π using the proving key p, the public statement y and the secret information also known as witness ω.

$$(\pi) \longleftarrow P(p, y, \omega) \tag{2.2}$$

- **Verify *V*** – The verifier can verify the public statement y by using proof π and verification key v, without revealing any information about ω. Verification will return *true* if proof of the statement y is correct.

$$V(v, y, \pi) = true|false \tag{2.3}$$

Zk-STARKs

Zk-STARK stands for Zero-Knowledge Scalable Transparent Argument of Knowledge [65]. It is an evolution of zk-SNARK that eliminates the need for a trusted setup, thereby enhancing security and transparency. Indeed, while zk-SNARK relies on elliptic curve pairings and requires a trusted setup phase to generate public parameters, zk-STARK employs polynomial commitments and interactive oracle proofs,

which not only ensure quantum-resistant proofs but also enable scalability for larger computations and transparency. In particular, zk-STARK leverages the Fast Fourier Transform (FFT) for efficient computation and verification of large datasets, while collision-resistant hash functions provide transparency and quantum resistance, ensuring long-term cryptographic security. However, this results in larger proof sizes than those of zk-SNARKs, which can pose problems for transaction costs.

The proof generation and verification of zk-STARK involves the usual three main steps described for zk-SNARK: Setup (S) to initialize public parameters; Prove (P) to generate proofs; and Verify (V) to check the validity of the proofs. The main difference resides in the characteristics and assumptions made in the setup phase.

Chapter 3

Literature Review

This chapter provides a comprehensive review of the state-of-the-art literature relevant to the research challenges addressed in this thesis. It is organized into five main sections, each focusing on a specific layer of the blockchain ecosystem. Section 3.1 discusses the smart contract design patterns and access control mechanisms that support modularity, scalability, and secure governance in decentralized applications, as well as their limitations in hierarchical governance. Section 3.2 presents an analysis of decentralized storage solutions and the challenges of ensuring data privacy in distributed networks. Section 3.3 explores the role of Zero-Knowledge Proofs (ZKPs) that enable privacy-preserving verification for digital certificates and selective disclosure. Section 3.4 analyzes how blockchain-based systems can be designed to better align immutability and traceability requirements with GDPR rights such as rectification and erasure. Finally, Section 3.5 reviews Layer 2 scalability solutions, with a specific focus on data availability challenges in rollup architectures. This analysis integrates different topics by illustrating how security, privacy, regulatory compliance, and scalability collectively limit the design of functional blockchain systems, and it highlights the gaps that motivate the contributions developed in the following chapters.

3.1 Smart Contract Design Pattern and Access Control Mechanisms

The literature on smart-contract design patterns demonstrates the relevance of modular, reusable approaches, with the factory pattern often discussed as a means to improve scalability and reduce deployment costs. However, its application in decentralized environments remains constrained because it does not support hierarchical contract families or adaptable role-based mechanisms. Similarly, studies on access control reveal that while role-based models are well-established in traditional systems, they cannot be directly transferred to blockchain, where decentralization and immutability prevent centralized enforcement. Token-based access control solutions have been explored, but they often lack the flexibility needed in multirole contexts. These gaps motivate the investigation of enhanced design methodologies that integrate hierarchical structures with multirole authentication and authorization

to support complex decentralized applications.

In [8] Rajasekar et al. explore various design patterns developed for blockchain applications. Among these, the most important for the proposed solution is the *factory contract* pattern. The factory pattern stores a template contract, known as the factory, on the blockchain, from which it is possible to instantiate similar child contracts, thereby improving modularity and consistency. The other patterns proposed are: the *checks-effects-interactions* pattern, which emphasizes secure transaction sequences, the *oracle* pattern focusing on integrating external data into the blockchain, the *off-chain data storage* and the *state channel* patterns, which together address the challenges of efficient data storage strategies, minimizing on-chain data while ensuring integrity. Finally, the authors propose the *contract registry* pattern, which enables flexible updates to contract logic, and the *emergency stop* pattern, which introduces a safety mechanism to counteract malicious activities. All these patterns demonstrate evolving best practices in the design of blockchain applications, emphasizing the importance of security, efficiency, and adaptability. With reference to design patterns for specific application domains, Zhang et al. [66] investigate their application to enhance the design and functionality of blockchain-based healthcare systems. They identify the following four main patterns: the Abstract Factory, Flyweight, Proxy, and Publish-Subscribe patterns. The *abstract factory* pattern provides a structured approach to creating user accounts. It allows the instantiation of objects without specifying exact classes, simplifying the creation of new related contracts for departments and subdivisions. The *flyweight pattern* addresses the critical challenge of large data storage for smart contracts by sharing data across similar objects. It is useful in scenarios where large amounts of patient data, such as insurance and billing information, need to be stored efficiently and securely. The *proxy pattern* serves as a placeholder to ensure patient data privacy by giving access to metadata. This approach maintains an audit trail for data operations, enhancing transparency and data integrity on the blockchain. The *publisher-subscriber* pattern enhances information dissemination by notifying relevant stakeholders, such as when a prescription request is made. In relation to the factory pattern, Wu et al. in [22] emphasize its significance in smart contract design. In particular, they demonstrate their usefulness for applications that require consistent behaviors to be replicated across multiple smart contracts. Additionally, the implementation of a factory contract pattern in 28 DApps and 2671 smart contracts highlights its pivotal role in DApp development.

Compared to the traditional factory pattern already present in the literature, the hierarchical factory pattern proposed in this thesis offers an improved, structured, and scalable approach. Namely, while the traditional factory or abstract factory patterns can only deploy similar instances of smart contracts, the hierarchical factory pattern can deploy diverse factories and contracts. This makes it more versatile and adaptable to diverse applications, as well as more compatible with complex hierarchical organizational structures.

An important aspect of smart contract design is defining who can access and execute functions and data. However, the decentralized and transparent nature of blockchain makes it challenging to implement access control. Several design patterns have been proposed to address the access control challenge in smart contracts.

The *role-based access control* (RBAC) pattern is an access control design pattern that assigns roles to users and grants permissions to those roles. In [23], Cruz et al. present RBAC-SC, which employs Ethereum smart contracts to model trust and endorsement relationships among roles, organizations, and services. The scheme also includes a challenge-response authentication protocol to confirm the ownership of user roles. This design is effective for flat hierarchies but does not support recursive factories or multi-layered governance. Another access control design pattern is the *token-based access control* (TBAC), which employs tokens to represent user access rights. The TBAC enables users to transfer or delegate access rights by transferring tokens. Liu et al. in [67], introduce the SMACS framework, which enables the low-cost implementation of updatable and sophisticated access control rules (ACRs) for smart contracts. It reduces gas costs by 22% compared to RBAC-SC. However, SMACS focuses on static roles and cannot adapt permissions dynamically, based on real-time sensor data, a limitation in IoT-driven systems such as water management. In [68], Zhou et al. propose a smart contract-based ownership access control framework for the IoTs in smart home systems. The proposed framework utilizes the ownership design pattern to define a hierarchy of owners, including device, home, and system owners.

With reference to the application of Blockchain technology in specific domains, it has gained significant attention in critical infrastructure in recent years due to its ability to ensure transparency, immutability, and decentralized trust. In energy management, Xia et al. [69] propose a blockchain framework for intelligent water systems, focusing on data integrity through hash storage on the chain. However, this work lacks granular access control, relying instead on a centralized administrator for permissions, which is a critical flaw in distributed systems. Similarly, Rana et al. [70] explore blockchain applications for sustainable tourism, demonstrating how smart contracts automate service agreements; however, they overlook scalability challenges in hierarchical organizations. In healthcare, Pinto et al. [71] designed a blockchain-based system for the traceability of health data, leveraging RBAC to manage the roles of patients and providers. While their approach enforces strict access policies, it relies on a single factory contract for deployment, which limits scalability in multi-institutional ecosystems. Recent advancements in decentralized identity management, such as those by Stockburger et al. [72], utilize Self-Sovereign Identities (SSI) for public transportation systems. This framework eliminates centralized authority but struggles with dynamic role assignment during emergencies, a key requirement for critical infrastructure. Finally, Zhang et al. [66] apply the abstract factory pattern to healthcare, inspiring similar modular designs in resource distribution networks, but without Multirole Access Control (MAC).

To the best of our knowledge, no existing solution integrates a hierarchical factory deployment pattern with a multirole access control mechanism that enforces role consistency across all levels of the hierarchy. This thesis addresses this gap in Chap. 4 by introducing a hierarchical factory pattern with a tightly coupled multirole authentication and authorization mechanism, specifically designed for complex, decentralized applications.

3.2 Decentralized Storage and Data Privacy

The integration of IPFS and blockchain technology offers a significant advancement in distributed data storage and security. This section reviews existing research on blockchain-based data storage, using IPFS, encryption, and hashing techniques to ensure data security and privacy. Blockchain technology is garnering significant attention for its potential to provide secure, decentralized data storage. Studies on blockchain have primarily explored its immutable ledger system for data storage; however, they often emphasize the limitations of the high costs and inefficiencies of storing large volumes of raw data directly within the blockchain. Monrat et al. [3] propose an in-depth analysis of these challenges, emphasizing the need for more effective data storage solutions within the blockchain framework. In this regard, IPFS is gaining popularity as a scalable data storage solution thanks to its decentralized, efficient design. Leveraging blockchain and IPFS in decentralized storage systems has revolutionized data storage and retrieval, particularly in sectors that demand high integrity and efficiency.

In [73], Haque et al. employ blockchain technology to transmit data in a peer-to-peer network and utilize IPFS as a data-sharing infrastructure to share pre-trained deep learning models with stakeholders. Similarly, in [74], Shah et al. propose a blockchain and cloud-based decentralized secure storage system to ensure data availability, privacy, and efficient resource utilization. In [75], Hao et al. focus on agricultural product traceability, utilizing IPFS to store multiple data types and blockchain to secure IPFS hash addresses, thereby improving query efficiency and data authenticity. However, the sensor data is collected and processed on a centralized private server before being stored on IPFS. Therefore, this approach may pose security risks, including a single point of failure, data loss, or a compromised server. In [76], Arer et al. integrate blockchain, IPFS, and Elasticsearch to address the challenges of big data storage in distributed systems, resulting in reduced storage overhead, improved search latency, and enhanced access control. The proposed approach relies on blockchain's inherent security features. Moreover, the Elasticsearch approach focuses primarily on search latency and precision, without considering the security implications of its query-handling mechanisms. Hasan et al. in [77] propose a secure and decentralized framework for managing cloud data provenance by integrating blockchain technology with the IPFS. The proposed solution ensures the availability, security, and integrity of provenance data by utilizing decentralized storage and blockchain for immutability. The method for verifying data integrity is secure; however, it may be computationally intensive, particularly when frequent validation of large datasets is required. Zheng et al. in [78] propose integrating blockchain and IPFS, demonstrating the potential for secure data retrieval and storage. The proposed approach leverages different privacy modes for data privacy and security: storing nonsensitive data on IPFS without encryption, and sensitive data with two-layered encryption. This approach mitigates the risk of a single point of failure and data tampering. However, data retrieval and queries require decryption of data stored on IPFS, which may introduce additional security and privacy risks by exposing private keys.

Data privacy has been a significant challenge in decentralized systems. En-

cryption and hashing have been pivotal in securing data within the blockchain and IPFS frameworks. The literature on various encryption algorithms and hashing techniques, as studied by Huang et al. [79], demonstrates the advancement of these methods in maintaining data integrity and security. Furthermore, Hassan et al. in [80] focus on the role of these techniques in preserving data privacy in decentralized systems and on the importance of balancing accessibility and security. Satybaldy et al. in [81] delve into privacy-preserving approaches, including ZKPs and homomorphic encryption. These studies emphasize the development of techniques that secure data without compromising privacy. However, these techniques can address data security and privacy but often fall short in balancing data security with system efficiency, such as computation overhead, scalability, and handling dynamic data. Additionally, relying on encryption inadvertently exposes private keys during decryption processes, posing a substantial security risk. Wang et al. [82] propose a blockchain-based framework for preserving privacy and ensuring secure access control in cloud storage. The proposed scheme integrates the Ethereum blockchain and ciphertext-policy attribute-based encryption (CP-ABE) to enhance user data security and privacy. However, due to privacy issues and data leakage, the cloud might not be a trustworthy source for data storage.

Despite these advancements, existing solutions rely on decryption during data retrieval, which exposes private keys and introduces security risks [25]. To the best of our knowledge, no existing solution eliminates decryption from the retrieval process while maintaining both privacy and auditability. The proposed solution, described in Chap. 5, addresses this gap by storing only the Content Identifier (CID) on-chain and distributing encrypted data on IPFS, removing private key exposure from the retrieval process entirely.

3.3 Zero-Knowledge Proof in Blockchain Applications

Traditional certificate validation typically relies on a single central authority, such as a government agency or the issuing institution, to verify the authenticity of the information presented. This centralized strategy establishes a clear trust anchor, but it also comes with significant challenges. Verification is often manual and time-consuming because it requires direct communication with the issuer or querying a centralized portal, and it involves waiting for back-office checks, which are prone to human error or tampering [83]. For instance, in the University domain, verifiers often need to contact the issuer to confirm educational credentials, which delays recruitment and is prone to fake credentials. Furthermore, the centralized model presents a single point of failure. The validity and availability of certificates depend on the continuous existence and integrity of records in the central authority. If the central authority is compromised or unavailable, independent verification of certificates becomes challenging [84]. The centralized system also lacks transparency for third-party verifiers, since it must fully trust the issuer. Furthermore, interoperability across different channels is also a challenge. At its core, while a centralized system ensures verification by official issuers, it is vulnerable to fraud and serves as a single point of failure, motivating the development of more robust solutions.

To overcome the constraints of centralized systems, several blockchain-based

frameworks have been proposed for certification verification [85]. Blockchain provides a decentralized trust infrastructure. Once a certificate (or its cryptographic fingerprint) is recorded in the ledger, it becomes tamper-proof and independently verifiable by any party [84]. For instance, as shown on the MIT Blockcert platform, academic certificates can be issued on a blockchain with a digital signature, allowing anyone to verify the certificates independently without relying on the university or issuer [26]. The main advantage of these techniques is transparency and immutability. The data stored on the blockchain cannot be tampered with or modified, and its distributed nature eliminates the need for third-party verification. Several studies and prototypes have demonstrated the advantages of using blockchain to log degree information or professional licenses, securing credential storage and automated verification through smart contracts [86, 87, 88]. These advancements underscore how blockchain frameworks can enhance trust and efficiency in credential management by eliminating the need for central authorities.

However, despite these advantages in tamper-resistance and the introduction of decentralized trust, blockchain-based certificate solutions also face challenges in privacy and scalability. For instance, Blockcert or recent implementations [89, 90, 91] do not protect sensitive certificate data or metadata, which is visible to everyone on the blockchain network, raising privacy concerns. Moreover, performance and cost are major issues, particularly for public blockchains such as Ethereum, which can lead to high transaction fees and latency in certificate verification, thereby hindering large-scale adoption [92]. Some solutions proposed permissioned or hybrid ledgers to mitigate costs [93]; however, they can reintroduce the centralization problem. These highlighted issues led to the development of techniques for improving privacy and compliance in a decentralized environment.

One prominent solution is to integrate ZKP to enhance privacy and security. ZKP enables a party to prove a statement to a verifier without disclosing sensitive information. This is especially crucial in a certificate verification system where sensitive information must remain protected. Partala et al. [7] survey non-interactive ZKP protocols and emphasize their significance for blockchain applications that verify facts without revealing secrets. Berrios Moya et al. in [94] introduced a dual-blockchain academic credential system using zk-SNARK proofs to verify a student's degree authenticity without exposing any student records. In the healthcare domain, Bai et al. [95] proposed a blockchain-based identity management system (Health-zkIDM) that uses ZKPs to protect patient information during credential verification. However, integrating ZKP with certificate verification presents its own set of challenges. ZKP protocols (zk-SNARKs or zk-STARKs) can introduce computational and implementation complexity. Proof generation and verification can significantly impact system performance and require specialized cryptographic libraries, especially when handling large datasets or multiple real-time verifications.

In light of the aforementioned advantages and despite potential difficulties, integrating blockchain with ZKP remains a promising solution to address the transparency-privacy paradox in public ledgers. The solution proposed in this thesis is the first to combine a Merkle tree-based selective disclosure mechanism with on-chain ZKP verification. This integration provides a complete certificate management system covering issuance, validation, and revocation, while preserving privacy throughout.

Unlike existing solutions, which either lack fine-grained selective disclosure or do not combine ZKP verification with on-chain revocation management, the proposed framework addresses all three requirements in a single design.

3.4 Blockchain and GDPR Compliance

Blockchain is frequently mentioned in the literature as a valid support for data protection regulations, like the GDPR, due to its immutability, traceability, and availability. However, integrating blockchain with full GDPR compliance presents several challenges. One major issue concerns the immutability of blockchain, which conflicts with certain GDPR rights for data subjects, such as the "Right to be forgotten". Furthermore, the decentralized nature of blockchain makes it difficult to identify and attribute responsibility to a specific data controller, complicating the enforcement of GDPR compliance. A preliminary investigation into the use of blockchain for restoring user ownership of their data and ensuring ethical data acquisition and provisioning has been conducted by Migliorini et al. in [96]. The paper theorizes how such technology could be decisive in giving users back ownership of their data, but does not provide any implementation of the introduced concepts. Truong et al. in [28] proposed a blockchain-based solution to manage personal data in compliance with the GDPR, utilizing blockchain and smart contracts. This approach enables users and service providers to maintain decentralized control over their personal data, while ensuring transparency and traceability. This study implements decentralized authentication, automated access controls, decentralized access tokens, and immutable logs of all data activities using Hyperledger Fabric, a permissioned blockchain framework. However, using Hyperledger Fabric, which is not a public blockchain, imposes certain limitations, such as reliance on a small number of nodes, which can restrict scalability and introduce centralization concerns, making it more challenging to achieve true decentralization.

Daudén-Esmel et al. in [27] proposed a blockchain-based access control system designed specifically for GDPR-compliant management of personal data. This approach focuses on providing transparent, immutable records of agreements between data subjects and service providers, thereby ensuring accountability and trust. This study proposes using smart contracts to record consent metadata and manage permissions transparently, while actual data resides off-chain. Off-chain storage enhances scalability, while consent, access, and disclosure records remain securely stored on the blockchain. However, data erasure and rectification happen off-chain without complete control over them. Ahmad et al. in [97] proposed a user-centric blockchain-based approach to verify the GDPR compliance in a multi-cloud environment. However, managing the extensive data operations across multiple cloud providers limits the scalability of the approach. Similarly, Barati et al. in [98] developed a blockchain-based model to verify GDPR compliance, particularly for IoT applications.

Existing blockchain-based solutions addressing GDPR compliance mainly rely on off-chain mechanisms to support data subject rights such as rectification and erasure, or require partially trusted infrastructures. While these approaches preserve blockchain immutability and traceability, they provide limited technical control over

data deletion and updates and often shift critical compliance guarantees outside the blockchain. No existing solution provides a blockchain-based framework that simultaneously preserves the audit trail, supports the GDPR right to erasure, and enforces these guarantees through verifiable on-chain mechanisms. In contrast, this thesis integrates smart contracts, off-chain storage, and cryptographic techniques to protect GDPR rights through verifiable, auditable mechanisms, without compromising the core blockchain properties.

3.5 Layer 2 Scalability and Data Availability

In the context of rollups, data availability refers to the guarantee that each user can access transaction data to verify the L2 chain's state updates and independently reconstruct the rollup's current state. Publishing proof of computation or hash on L1 ensures the system's integrity and confirms that the L2 state has not been tampered with. Conversely, posting complete transaction data on L1 ensures that all participants can independently validate state transitions and generate fraud proofs if necessary. This approach leverages the security and decentralization inherent in L1, while ensuring data availability even when L2 becomes compromised or unavailable. In particular, L2 transactions are considered finalized only when the corresponding transaction data is posted in L1 as part of a batch. This maintains a balance between scalability and security, enabling rollups to achieve higher transaction throughput without compromising the security that Layer 1 (L1) promises. When L1 is Ethereum, rollup data is typically permanently stored inside *calldata*, a space within transactions originally designed to store function call arguments. However, this solution is expensive and does not scale effectively; indeed, calldata costs 16 units of gas for a non-zero byte and four units of gas for a zero byte [99]. Moreover, due to fluctuations in the ETH price, posting rollup data to L1 is often postponed until the price becomes more favorable. Therefore, in this scenario, the data posting cost from L2 to L1 is twofold: there is an inherent cost for posting and an indirect cost for the possible delay. The former can be directly measured when batches are posted to L1 and depends on network congestion and transaction complexity; the latter is not directly measurable but results in degraded L2 transaction finality, security, and usability. A. Mamageishvili et al. in [100] design an effective and robust algorithm based on Markov decision processes to efficiently determine, based on current L1 prices, when it is convenient to post transaction batches to L1. In particular, it relates the cost of posting to the cost of delaying, trying to avoid posting data when the price is too high, while minimizing delays. This solution was proposed before the introduction of Ethereum's blobs, so it assumes that rollup data is stored solely within calldata. Similarly, Y. Bar-On et al. in [101] propose a general model to determine when to publish L2 transaction data on L1, generalizing the previous approach by introducing additional cost functions suitable for a broader range of L2 solutions.

Ethereum Improvement Proposal 4844 (EIP-4844), introduced in March 2024, aimed to overcome the limitations of calldata for storing rollup information. It introduced a new data structure called *blob* [102]. Like calldata, blobs are also stored in the blockchain. However, they are automatically deleted after 18 days (4096 epochs),

which reduces Ethereum's processing fees. An EIP-4844 blob consists of 4096 elements, each 32 bytes long, for a fixed total storage space of 128 KByte. D. Crapis et al. in [31] proposed a solution in which rollups are stored, alternatively, on calldata or blobs, depending on the contingent situation. They identify that posting rollups in calldata is better than posting in blobs by using Nash's economic theory to analyze the problem from a game-theoretical perspective. Conversely, S. Park et al. in [103] conduct an empirical analysis of the effect of blobs on four dimensions: consensus security, ETH cost, user delays, and blob gas fee market. They highlight the increase in fork rates and block delays correlated with the quantity of blob data per slot, which may have implications for the network's security. S. Lee in [30] suggests a solution to the fixed blob size issue by introducing the concept of *shared blob*, which holds multiple rollups, and analyzes its potential impact on reducing the overall rollup cost. This approach always posts a full blob to avoid wasting resources. However, if the transaction arrival rate is low, filling a blob can cause delays. Moreover, the solution has not been implemented, and the related actual details have not been studied.

To the best of our knowledge, no existing solution addresses blob space underutilization through an adaptive allocation mechanism that accounts for rollup size, data volume, and submission frequency simultaneously. The proposed solution builds on the survey in [29] and introduces a shared blob approach combined with an Adaptive Multi-Factor Scoring mechanism, described in detail in Chap. 8.

3.6 Summary and Comparison with the Thesis Contributions

Tab. 3.1 summarizes the main existing approaches reviewed in this chapter, their limitations, and the corresponding thesis contribution for each identified problem. As shown in Tab. 3.1, the five areas share a common gap: existing solutions address governance, storage security, privacy, compliance, and scalability as separate problems, each in isolation. The mechanisms used to close these gaps, in particular ZKP-based verification and on-chain/off-chain data integration, are not exclusive to blockchain; both apply to any distributed system that requires verifiable claims about data without full disclosure [7]. The gaps identified in this chapter directly motivate the five contributions developed in Chap. 4–8.

Table 3.1: Summary of related work, key limitations, and thesis contributions.

Research Area	**Representative Literature**	**Key Limitations**	**Thesis Contributions**
Smart contract design and access control	[8, 22, 23, 67]	Limited to flat contract instantiation; RBAC is not integrated with factory-based deployment; no role consistency is enforced across contract hierarchies	A hierarchical factory pattern with multi-role access control (Chap. 4)
Decentralized storage and data privacy	[73, 77, 78, 82]	Data retrieval depends on decryption, which may expose private keys [25]; data may become unavailable when IPFS nodes go offline	A blockchain–IPFS framework that avoids decryption-based retrieval (Chap. 5)
ZKP-based digital certification	[26, 94, 95]	Existing approaches do not jointly provide selective disclosure, on-chain ZKP verification, and revocation management within a unified framework	A Merkle-tree-based selective disclosure framework with on-chain ZKP verification and revocation support (Chap. 6)
Blockchain and GDPR compliance	[27, 28, 97]	Off-chain compliance shifts trust guarantees outside the blockchain and lacks verifiable on-chain support for rectification and erasure	An on-chain/off-chain architecture that supports GDPR rights through verifiable mechanisms (Chap. 7)
Layer 2 scalability and data availability	[29, 30, 31, 100]	Fixed blob sizes waste resources for small rollups; existing approaches lack adaptive allocation strategies for heterogeneous rollup workloads	A shared-blob approach with an Adaptive Multi-Factor Scoring mechanism (Chap. 8)

Chapter 4

Smart-Contract Design Patterns and Access Control Mechanisms

Smart contracts represent the foundation of decentralized applications by automating agreements in a transparent and immutable way. However, their design and deployment are still constrained by several challenges that limit their applicability in real-world infrastructures. These challenges primarily concern scalability, modularity, security, and manageability [104]. While scalability will be discussed in Chap. 8, this chapter concentrates on the lack of modularity in smart contracts. As applications expand, contracts become more complex and interdependent, making their maintenance and reuse more difficult. Without proper separation of concerns, developers struggle to isolate functionality, adapt contracts to new requirements, or extend their logic without introducing vulnerabilities. This problem becomes even more evident in hierarchical domains such as private or public company scenarios, where entities are naturally organized at multiple levels and require scalable contract families to reflect such structures. Moreover, in these scenarios, it is crucial to ensure that only authorized roles can perform sensitive operations. Traditional role-based access control (RBAC) models, developed for centralized systems, cannot be directly applied in blockchain contexts. Token-based approaches have been proposed, but they remain inflexible and unsuitable for scenarios requiring dynamic role management. Without robust Multirole Access Control (MAC) mechanisms, contracts are vulnerable to signature replay attacks, privilege escalation, and unauthorized access, which significantly reduces trust in the system [23].

The absence of design methodologies to efficiently manage large families of contracts or to support the dynamic reassignment of roles increases the gap between practical requirements and existing solutions [105]. This thesis addresses the need for new design patterns and access control mechanisms that support complex, hierarchical applications while preserving the decentralization, immutability, and transparency of blockchain technology.

4.1 Motivating Example

This section presents a motivating example derived from the water management domain to illustrate the practical relevance of the proposed approach. Water manage-

ment is a critical aspect of modern infrastructure, requiring continuous monitoring and control of water distribution and quality at multiple administrative levels. A water management system can be organized in a hierarchical structure, similar to that shown in Fig. 4.1. At the highest level, a water distribution authority serves as the super administrator, managing permissions and configurations across multiple cities to ensure that water is extracted, processed, and distributed efficiently while adhering to regulatory policies on conservation and quality standards. This authority establishes baseline permissions and oversees the creation of subordinate entities represented by the set of supervised cities. Each city is responsible for managing its own water distribution network, which typically comprises multiple districts. At the same time, each district oversees a set of physical water treatment or pumping facilities. At the lowest level, each station comprises multiple water pumps and sensors that monitor essential parameters, including flow rate, pressure, pH levels, and turbidity.

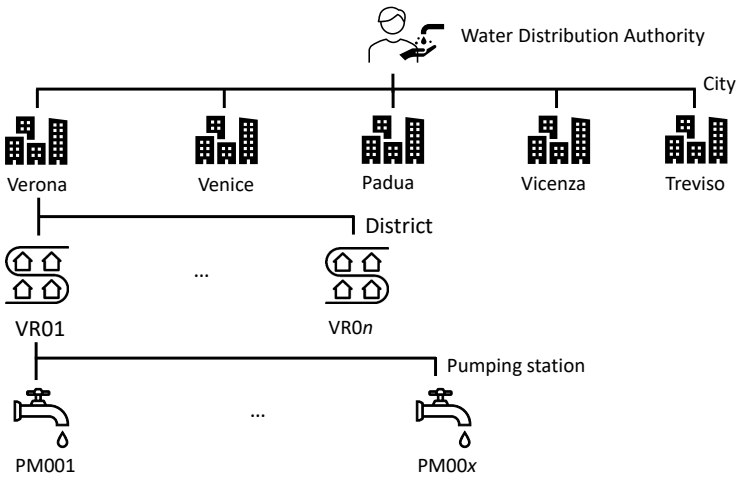


Figure 4.1: Hierarchy of actors inside a water management system.

Blockchain technology and smart contracts could be successfully implemented in such a water management system. By combining blockchain technology with IoT sensor data, the system creates an immutable, transparent audit trail that supports operational efficiency and regulatory compliance, ultimately establishing a resilient framework for modern water management. In particular, while IoT sensors continuously collect data stored off-chain on distributed storage systems such as IPFS, secure records of the hashes of such data are stored in the blockchain to ensure transparency and regulatory compliance. At the same time, when implementing a water management system through a dApp, it becomes clear that each level of the hierarchy is responsible for deploying and managing its own smart contracts while remaining dependent on the parent level. Moreover, the super administrator, also known as the Water Management Authority, grants and revokes permissions to ensure that only authorized roles manage water distribution and access water records. The city water departments are key client factories, coordinating their activities with the central authority and local subdivisions. They handle local water delivery and quality monitoring while implementing additional smart contracts to manage different districts. Finally, the district stations at the lowest level directly interact with water treatment

facilities and sensor networks. This real-world scenario highlights the limitations of a traditional factory pattern and suggests the need for a more sophisticated hierarchical factory pattern, as proposed in this thesis. In particular, its application facilitates efficient water management across different administrative levels and ensures that each system component can be independently updated and secured. The multirole authentication and authorization mechanism enforces strict access control, ensuring that operations are conducted only by entities with appropriate permissions.

The following sections provide a detailed discussion of how this real-world scenario can be implemented using a hierarchical factory pattern integrated with multirole authentication and authorization mechanisms.

4.2 Factory Pattern and Its Limitations

The factory pattern is one of the most widely recognized design patterns for smart contract development [8]. Its principle is to deploy a single factory contract that can generate multiple instances of a predefined contract template, simplifying their management and tracking. This reduces deployment costs, as the factory contract is deployed only once, while subsequent child contracts are created without requiring the redeployment of the entire bytecode. The pattern also contributes to modularity and security by centralizing contract instantiation and reducing the risk of duplicated vulnerabilities [22]. These characteristics make the factory pattern particularly useful in decentralized applications that require consistent behaviors across multiple contract instances. By promoting reusability and reducing redundancy, the pattern addresses some of the fundamental scalability challenges of smart-contract deployment, particularly the limitations of the Ethereum Virtual Machine (EVM) on the maximum bytecode size of deployed contracts, which traditionally discourage sophisticated designs and restrict extensibility.

Nevertheless, the traditional factory pattern presents several limitations when applied to real-world infrastructures. Its structure allows only one family of contracts to be instantiated, limiting flexibility in applications that require different variants or hierarchically related contracts. For example, while it is possible to replicate many instances of the same contract type, the pattern cannot support the creation of a coordinated family of heterogeneous contracts that mirror the hierarchical organization of domains such as the water management systems described in Sect. 4.1 [106]. This limitation is critical because many blockchain-based infrastructures operate in multi-level environments, where contracts must not only replicate functionality but also interact across layers of authority and responsibility.

Another challenge in the traditional factory pattern is its lack of integrated access control. Factories and their children are created without role-aware interaction between levels, which can expose them to unauthorized operations and limit the ability to enforce differentiated responsibilities. In scenarios where parent contracts must govern or restrict the actions of their child contracts, this limitation undermines security and manageability. Moreover, the absence of hierarchical governance leads to rigid deployments, where dependencies between contracts must be managed off-chain or through ad hoc mechanisms, thereby increasing complexity and operational risks. For these reasons, while the factory pattern successfully improves reusability

and cost efficiency, it is considered insufficient for complex decentralized applications. To address these shortcomings, the proposed solution extends the factory pattern to a hierarchical form, supporting multiple layers of factories and contracts, and integrates MAC mechanisms to ensure role-specific interactions at each level.

4.3 Hierarchical Factory Pattern Architecture

The hierarchical factory pattern proposed in this thesis extends the classical factory design by introducing a multi-layered structure for creating and managing smart contracts. Unlike the traditional approach, which is limited to instantiating a single contract type, the hierarchical factory enables the creation of factories that can act as parents for additional factories or contracts. This recursive structure yields a tree-like hierarchy in which each node can generate and govern subordinate elements. This layered architecture is particularly suited for decentralized applications that mirror real-world organizational models, where multiple administrative levels must collaborate while retaining independent responsibilities. Fig. 4.2 shows the general structure of the hierarchy: the mandatory parts of the tree have been highlighted in yellow. The white boxes correspond to optional levels that can appear in any number greater than or equal to zero, depending on the specific characteristics of the application domain.

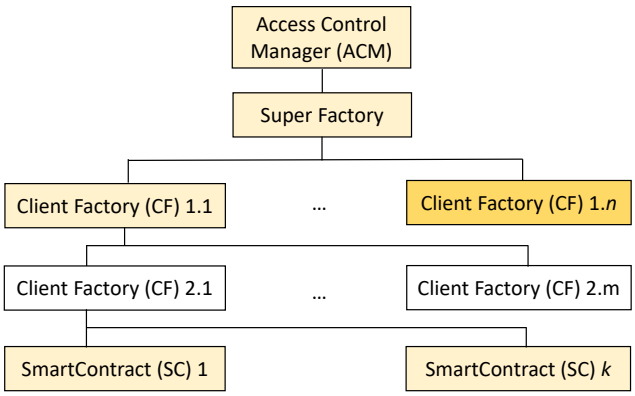


Figure 4.2: The Hierarchical Factory Pattern.

The root of the tree hierarchy is the *access control manager* ACM smart contract, which plays a crucial role in managing permissions, roles, and access levels for the various children. It is responsible for ensuring authorized interaction with factories and smart contracts. The second level of the hierarchy is represented by the SuperFactory contract. This entity serves as a central hub for deploying client-specific sub-factories within the hierarchy and manages connections to the ACM to acquire permissions and roles for these sub-factories. The ACM and the SuperFactory can be considered a single component in the hierarchy, since they manage the hierarchical generation of factories and smart contracts that are enhanced with authentication and authorization capabilities. They have been split into two smart contracts here to increase modularity: the SuperFactory can be customized for the application domain, thereby reducing the bytecode size of the deployed contracts.

More specifically, with reference to the motivating example introduced in Sect. 4.1, besides the ACM contract that will be described in more detail in the following, the water distribution authority deploys the central WMSFactory smart contract, which customizes the SuperFactory and establishes baseline permissions and oversees the creation of subordinate entities by using a hierarchical factory pattern. At the city level, a CityFactory deploys and tracks smart contracts that govern district-level operations. Meanwhile, each district oversees a set of physical water-treatment or pumping facilities via a DistrictFactory. Finally, the last level involves smart contracts that ensure the information collected from IoT devices is immutable. The contextualization of the hierarchy in Fig. 4.2 to this use case is depicted in Fig. 4.3.

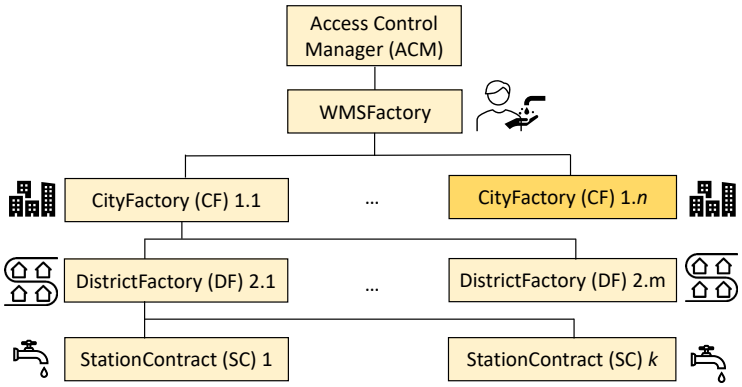


Figure 4.3: Hierarchical Factory Pattern contextualized to the water management scenario.

4.3.1 Access Control Manager

The structure of the ACM contract is reported in Alg. 1. At the beginning of this contract, it is necessary to specify the list of permission roles (lines 2-4) and the set of contract types composing the hierarchy in Fig. 4.2 (line 5). Depending on the specific application requirements, the number and types of both roles and contract types can be adjusted straightforwardly. However, the ADMIN role must be defined and automatically assigned to the deployer of the ACM smart contract. This role assignment is a foundational security measure to ensure that the primary administrative control remains with the entity responsible for contract creation. In the water management system, three roles are defined: the ADMIN, WATER_MGR, and VERIFIER. The ADMIN has complete access to all functions across all smart contracts, including the creation, modification, deletion, and viewing of data at every level of the hierarchy. This role also has the authority to create any new sub-factory or smart contract within the hierarchy using the functions in the WMSFactory module. This role resembles the administrative user in a traditional information system, which has access to all the system's functionalities and can also manage other users' roles by revoking and assigning grants. The WATER_MGR role identifies users responsible for overseeing water distribution at each level of the hierarchy. In particular, a user assigned to this role for a CityFactory contract can add a new district factory for that city. Similarly, a user assigned to this role inside a DistrictFactory contract can create new station contracts to monitor the activities of that specific district. A WATER_-

MGR role also gives assigned users the right to access reading and writing functions inside the corresponding smart contract. On the other hand, the VERIFIER role has been introduced to grant read-only access to data stored on the blockchain. A user assigned to this role can audit IoT sensor data for verification purposes without altering it by calling the appropriate read-only function. Regarding the layers, the system is organized into four mandatory layers, as depicted in Fig. 4.3, each corresponding to a different element in the ContractType enumeration, thereby reinforcing a clear separation of responsibilities and access privileges throughout the hierarchy.

It is essential to note that both roles and layers must be predefined at design time in an ACM contract and cannot be modified after its deployment. This aligns with conventional RBAC systems commonly found in traditional information systems, where roles and their associated permissions are identified during the requirements analysis. Moreover, this static definition of roles and layers is inherently tied to the immutable nature of smart contracts, which offer users greater trust and security, as the underlying logic and assigned roles remain immutable after deployment. At the same time, even if roles and levels in the hierarchy are predefined, the system allows dynamic reassignment of individual user addresses to the identified roles, maintaining flexibility in user management while preserving the structural integrity and security guarantees inherent to the contract design. Additional details about authentication and authorization functions will be provided in Sect. 4.4.

Given that, the ACM contract maintains two main properties: a reference *_adminOwner* to the owner of the contract, which is a general administrator for the entire hierarchy (line 11), and a structured mapping called *_accessControl* (line 12). This map maintains, for each role r and user address u, the list of contracts that u can access with the role r. Each contract is represented through a data structure *ContractData* (lines 6-10), to maintain the information related to the contract address, the address of the contract owner, the address of the parent contract, i.e. the factory that generates it, and the type of the contract, i.e. the level of the hierarchy to which it belongs. Additional information can be added as needed, depending on the specific application requirements. Using this data structure is essential for implementing dynamic assignments between users and roles, as well as for revoking rules from compromised or outdated users and granting roles to new users. The last maintained variable is *_executedNonce*, whose use and importance will be described in more detail in Sect. 4.5.3 since it is related to the prevention of signature replay attacks [107].

The ACM constructor (lines 14-19) is responsible for three important aspects: (i) it registers the current ACM contract inside the *_accessControl* map and associates it with the ADMIN role for the *msg.sender* address. This means that the owner of the ACM contract, namely the address that deployed it, will act as an ADMIN user for the entire system, managing and revoking rights and roles as needed. (ii) It creates a new instance of the WMSFactory and registers it inside the *_accessControl* map with the same role and owner, making the same address also the owner of the WMSFactory contract with ADMIN grants. Therefore, the message sender will be able to manage rights and roles and deploy any factory or contract across the entire hierarchy. (iii) Finally, it stores the address of the message sender in the *_adminOwner* variable. The updates to the *_accessControl* map are performed by the function *cre-*

Algorithm 1: Structure of the access control manager (ACM) contract.

```
1  contract ACM :
2      public constant ADMIN;
3      public constant WATER_MGR;
4      public constant VERIFIER;
5      enum ContractType {ACM, WMSF, CF, DF, SC};
6      struct ContractData :
7          ContractType _contractType;
8          address _contractAddress;
9          address _ownerAddress;
10         address _parentContractAddress;
11     private address _adminOwner;
12     private mapping _accessControl;
13     private mapping _executedNonce;
14     constructor :
15         s ← msg.sender;
16         a ← address(this);
17         createNewContract(ADMIN, s, a, s, a, ACM);
18         createNewContract(ADMIN, s, address(new WMSF(a)), s, a,
             WMSF);
19         _adminOwner ← s;
20     function createNewContract(role, user, contr, owner, par, type) :
21         ContractData c ← new ContractData(type, contr, owner, par);
22         _accessControl[role][user].add(c);
23     function authnz(role, data, sign, nonce, contract) returns address :
24         // authentication and authorization
25     function addContractOnBehalfOf(contr, own, par, data, sign, nonce,
         type) :
26         address signer ← authnz(ADMIN, data, sign, nonce);
27         createNewContract(WATER_MGR, own, contr, own, par, type);
28         createNewContract(ADMIN, signer, contr, own, par, type);
29     function addContract(contr, parent, data, sign, nonce, type) :
30         address signer ← authnz(WATER_MGR, data, sign, nonce);
31         createNewContract(WATER_MGR, signer, contr, signer, parent,
             type);
32         createNewContract(ADMIN, _adminOwner, contr, signer, parent,
             type);
```

ateNewContract() (lines 20-22), which essentially creates a new instance of the data structure and stores it inside the map.

The function *authnz*() will be described in more detail in Sect. 4.4. It authenticates the user who has signed *data* with the signature *sign* and checks if this user has the required *role* for the *contract*. On success, it returns the *user* 's address.

This function is used in the last two functions, which assign role permissions to contracts deployed via factories within the hierarchy. In particular, the function *addContractOnBehalfOf*() could be invoked by the WMSFactory to deploy any contract instance anywhere in the hierarchy, both sub-factories and smart contracts, on behalf of the WATER_MGR role. Indeed, after authenticating the current transaction signer and checking that it has an ADMIN role, the function registers the given contract inside the *_accessControl* mapping by assigning administrative permissions to the signer (i.e., the WMSFactory calling the function) and WATER_MGR role to the contract owner specified as the parameter. This function enables administrators to deploy and manage contracts at any level of the hierarchy, assigning ownership and correct rights to the appropriate role. Conversely, each subfactory inside the hierarchy could invoke the function *addContract*() to deploy a DistrictFactory or *StationContract*, respectively. In this case, the signer is checked against the required role, and if both authentication and authorization succeed, the signer is registered as the contract owner. The ADMIN role is then assigned to the ACM owner. This function enables members of the hierarchy to deploy contracts under their responsibility, while assigning the administrative role to the ACM owner.

4.3.2 Water Management System Factory

The skeleton of the WMSFactory contract is illustrated in Alg. 2. In this implementation, the contract maintains a reference to the connected ACM contract and defines a modifier, *checkAdmin()* (lines 5-8) which invokes the ACM verification routine to ensure that only an entity with the ADMIN role can call sensitive functions. This security measure guarantees, in particular, that only an ADMIN user can invoke the *createClientContract*() function (lines 9-19). Based on the parameter *type* given as input and referring to the ContractType enumeration in line 5 of Alg. 1, the function can create any factory or smart contract within the hierarchy, returning the corresponding contract address. Finally, the new contract is registered in the ACM *_accessControl* mapping on behalf of the instantiated factory (line 19) by calling the function *addContractOnBehalfOf()* described previously. In the context of the water management system illustrated in Fig. 4.3, this design enables the ADMIN to deploy subordinate factories, such as CityFactory and DistrictFactory, as well as *StationContract*, reflecting the organizational structure of a water distribution system in which the distribution authority can manage all subordinate organization levels. The WMSFactory not only streamlines contract creation but also ensures that the hierarchical deployment of smart contracts follows the access control criteria outlined in earlier sections. Notice that the reference to the ACM contract is set by passing its address in the WMSFactory constructor and performing a cast. This operation is secure and does not introduce vulnerabilities in the contract since the constructor is invoked only once during the factory deployment by the ACM itself. Furthermore, the immutability of this reference is guaranteed by the *private immutable* variable, which prevents external contracts from altering it after deployment.

Algorithm 2: Structure of the WMSFactory smart contract.

```
1  contract WMSFactory :
2      private ACM _acm;
3      constructor (acm) :
4          _acm ← ACM(acm);
5      modifier checkAdmin() :
6          if !(_acm.hasRole(ADMIN, msg.sender, this)) then
7              revert NotAdmin();
8              _;
9      function createChildren(owner, par, data, sign, nonce, ty)
         checkAdmin() :
10         address c;
11         if ty = ACM.CF then
12             c ← address(new CityFactory(address(_acm)));
13         else if ty = ACM.DF then
14             c ← address(new DistrictFactory(address(_acm)));
15         else if ty = ACM.SC then
16             c ← address(new StationContract(address(_acm)));
17         else
18             revert IncorrectType();
19         _acm.addContractOnBehalfOf(c, owner, par, data, sign, nonce, ty);
```

4.3.3 City Factory

Further levels in the hierarchy are implemented through domain-specific factories that deploy and manage smart contracts, following the organizational structure of the water management system. At the city level, the CityFactory (see Alg. 3) is responsible for the implementation and tracking of smart contracts for district-level operations and the management of city data. In particular, each CityFactory defines a mapping (line 5) that records aggregated data collected from all city districts under its responsibility each year, using such information as the key. While function *addCityData*() could be invoked only by the user having WATER_MGR role for this specific contract, the corresponding reading function *getCityData*() could also be invoked by users registered with a VERIFIER role for it. This is obtained by the definition of two modifiers: *checkManager*() to restrict function calls to WATER_MGR role and *checkManagerOrVerifier*(), which allows users with the VERIFIER or WATER_MGR role to invoke read-only functions. To simplify the notation, the aggregated city-level data have been summarized as *cityData*. However, it can be extended and detailed to meet specific needs. Finally, the CityFactory can instantiate subordinate DistrictFactory through the function *createContract*(). This function deploys a new DistrictFactory and registers it in the ACM mapping through function *addContract*().

Algorithm 3: Structure of the CityFactory contract.

```
1  Contract CityFactory :
2      private ACM _acm;
3      constructor (acm) :
4          _acm ← ACM(acm);
5      private mapping _aggregatedData;
6      modifier checkManager() :
7          if !(_acm.hasRole(WATER_MGR, msg.sender, this)) then
8              revert NotAuthorized();
9              _;
10     modifier checkManagerOrVerifier() :
11         if !(_acm.hasRole(WATER_MGR, msg.sender, this) ∨
            _acm.hasRole(VERIFIER, msg.sender, this)) then
12             revert NotAuthorized();
13             _;
14     function createContract(parentAddr, data, sign, nonce)
        checkManager() :
15         _acm.addContract(address(new DistrictFactory(address(_acm))),
            parentAddr, data, sign, nonce, DF);
16     function addCityData(year, cityData) checkManager() :
17         _aggregatedData[year] ← cityData;
18     function getCityData(year) checkManagerOrVerifier() :
19         return _aggregatedData[year];
```

4.3.4 District and Station Contract

The DistrictFactory described in Algo. 4 represents the subsequent layer in the hierarchy, and follows a similar design. In particular, the differences reside in the *createContract*() function, which this time creates an instance of a *StationContract*, as well as in the type and frequency of data collection. In particular, the *districtData* mapping collects aggregated monthly data from the district's water stations. DistrictFactory employs the *checkManager*() modifier to restrict contract creation to users with the WATER_MGR role, while functions to retrieve stored data are protected by the *checkManagerOrVerifier*() modifier. Overall, CityFactory and DistrictFactory extend the hierarchical factory pattern by ensuring that smart contracts at each level comply with the overall access control policy and manage water data at the appropriate level of granularity. Therefore, the proposed hierarchical factory pattern offers much greater flexibility than traditional smart contract factories in the literature, which can create only new instances of the same smart contract type.

To conclude, a prototypical smart contract belonging to the leaf level is presented in Alg. 5. *StationContract* is responsible for interacting with sensor networks and physical water treatment facilities. Since it represents the last level of the hierarchy, it does not present any function for creating child contracts or factories. However, it

Algorithm 4: Structure of the DistrictFactory contract.

```
1  Contract DistrictFactory :
2      private ACM _acm;
3      private mapping _districtData;
4      constructor (acm) :
5          _acm = ACM(acm);
6      modifier checkManager() :
7          if !(_acm.hasRole(WATER_MGR, msg.sender, this)) then
8              revert NotAuthorized();
9              _;
10     modifier checkManagerOrVerifier() :
11         if !(_acm.hasRole(WATER_MGR, msg.sender, this) ∨
               _acm.hasRole(VERIFIER, msg.sender, this)) then
12             revert NotAuthorized();
13             _;
14     function createContract(parentAddr, data, sign, nonce)
         checkManager() :
15         _acm.addContract(address(new StationContract(address(_acm))),
             parentAddr, data, sign, nonce, SC);
16     function addDistrictData(date, districtData) checkManager() :
17         _districtData[date] ← districtData;
18     function getDistrictData(date) checkManagerOrVerifier() :
19         return _districtData[date];
```

only includes functions for collecting and retrieving data. In line with the previous contracts, access to these functions is controlled by role-specific modifiers. Due to the high volume of data produced by each station, which can be very fine-grained, the raw data is stored in IPFS and made immutable by including a content identifier (CID) and a hash. Further details about this are out of the scope of this work, and so are omitted.

The proposed design pattern offers a more flexible structure than the traditional factory pattern, as it can accommodate multiple layers, each handling specific functions and tasks. The proposed design pattern is scalable, since deploying a new contract instance requires adding a new reference only in the appropriate parent factory contract, without altering the rest of the smart contract hierarchy. In other words, newly created instances are distributed among many parent factories without loading any contract. The other important aspect which will be discussed in more detail in the following section, is related to the separation of roles and grants, that mimic the physical and administrative structure of a water management system: the ADMIN deploys the factories, the WATER_MGR can both create subordinate contract instances and update critical off-chain references, and the VERIFIER role allows secure, read-only access to ensure data integrity. This approach enhances modular

Algorithm 5: Structure of the StationContract.

```
1  Contract StationContract :
2      private ACM _acm;
3      private mapping _stationData;
4      constructor (acm) :
5          _acm = ACM(acm);
6      modifier checkManager() :
7          if !(_acm.hasRole(WATER_MGR, msg.sender, this)) then
8              revert NotAuthorized();
9              _;
10     modifier checkManagerOrVerifier() :
11         if !(_acm.hasRole(WATER_MGR, msg.sender, this) ∨
             _acm.hasRole(VERIFIER, msg.sender, this)) then
12             revert NotAuthorized();
13             _;
14     function addStationData(date, dataCID, dataHash) checkManager() :
15         _stationData[date] ← ⟨dataCID, dataHash⟩;
16     function getStationData(date) checkManagerOrVerifier() :
17         return _stationData[date];
```

deployment and granular access control while also establishing a verifiable relation between on-chain contracts and off-chain sensor data.

4.4 Integrated Multirole Access Control

This section addresses user authentication and authorization within the hierarchical structure outlined in the previous section. In particular, regarding authentication, the emergence of blockchain technology has brought about a paradigm shift in the digital security domain, moving from traditional centralized authentication and authorization systems to decentralized mechanisms. In fact, traditional information systems that rely on RBAC often use centralized authorities or servers to verify user identity, thereby exposing the system to a single point of failure, data breaches, and unauthorized access. Conversely, in the blockchain domain, there is no central authority, and robust, decentralized multirole authentication and authorization mechanisms are needed to distribute authentication information across a network of nodes, thereby making it difficult for malicious actors to undermine system integrity.

As already mentioned in Sect. 4.3, the proposed solution implements the multirole authentication and authorization mechanism in the ACM contract. In particular, ACM uses the *authnz*() function to authenticate users and check their roles through the *_accessControl* private mapping. The details of this mechanism and functionalities are reported in Alg. 6 and illustrated in Fig. 4.4. Before proceeding with the description of *authnz*(), it is necessary to note that, in the absence of a central server,

Algorithm 6: Structure of the access control manager ACM contract.

```
1  contract ACM :
2      function verify(data, sign) returns address :
3          return data.toEthSignMessageHash().recover(sign);
4      function hasRole(role, user, contract) returns bool :
5          ContractData[] cs ← _accessControl[role][user];
6          return cs.contains(contract);
7      function authnz(role, data, sign, nonce, contract) returns address :
8          byte32 hash ← keccak256(data, nonce);
9          if (_executedNonce[hash]) then
10             revert AlreadyExecuted();
11         address signer ← verify(hash, sign);
12         if (!hasRole(role, signer, contract)) then
13             revert NotAuthorized();
14         _executedNonce[hash] ← true;
15         return signer;
16     function revokeRole(role, user) onlyAdmin() :
17         if (user = address(0)) then
18             revert InvalidAddress();
19         if (user = _adminOwner) then
20             revert NotAuthorized();
21         _accessControl.delete(⟨role, user⟩);
22     function transferOwnership(address newAdmin) onlyAdmin() :
23         if (newAdmin == address(0)) then
24             revert InvalidAddress();
25         if (newAdmin = _adminOwner) then
26             revert NewAdminSameAsCurrent();
27         ContractData[] prev ← _accessControl[ADMIN][_adminOwner];
28         ContractData[] curr ← _accessControl[ADMIN][newAdmin];
29         for (i = 0; i < prev.length; i++) do
30             curr.push(prev[i]);
31         _accessControl.delete(⟨ADMIN, _adminOwner⟩);
32         _adminOwner ← curr;
33     function addVerifier(verifier, contract) onlyAdmin() :
34         if (verifier = address(0)) then
35             revert InvalidAddress();
36         _accessControl[VERIFIER][verifier].add(contract);
```

a blockchain-based system can exploit the transaction signature mechanism to provide authentication. Namely, a simple authentication mechanism within the wallet application is insufficient to provide strong and secure authentication, as a wallet is a client tool that a malicious user can compromise. Therefore, it is necessary to implement a mechanism that combines smart contract functionality and wallet signature capabilities to establish a robust, decentralized authentication system that involves network nodes. This approach requires users to actively sign a message with their wallet's private key, thereby generating cryptographic proof of their identity.

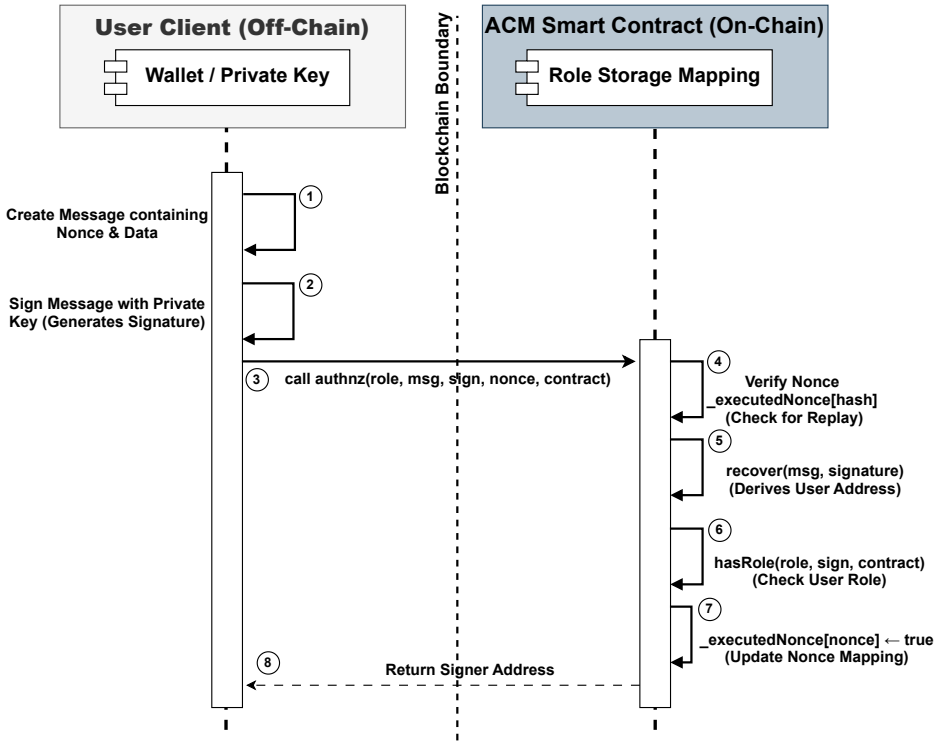


Figure 4.4: Execution flow of the user authentication and authorization using *authnz*().

The signed message is then verified with the *verify*() function illustrated in lines 2-3 of Alg. 6. This function takes two inputs: a hashed message *data* and a signature *sign*, from which it can retrieve the address of the user signing the message *data*. In order to do that, the proposed solution exploits the *recover*() function provided by OpenZeppelin's Elliptic Curve Digital Signature Algorithm (ECDSA) library[1]. This method allows the retrieval of the address that signs a message and then confirms that the message comes from an expected address. Indeed, without this authentication procedure, a malicious user can forge the signature, resulting in a mismatch between the *msg.sender* and the address that signs the message. In other words, the message was breached and not sent by the person from whom it was expected. Conversely, the success of this check requires that the transaction's sender is indeed the individual who signed the message hash. Although the current design leverages OpenZeppelin's ECDSA-based recovery due to its optimized implementation via

[1]https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/ECDSA.sol

the use of EVM precompiled *ecrecover*, future extensions of the proposed solution may consider alternative signature schemes, such as the use of the EIP-2098 short signature format or other cost-effective cryptographic methods, to further reduce gas consumption without compromising security.

The *verify*() function is used by *authnz*(), which performs both authentication and authorization (lines 7-15). Before safely retrieving the signer address (line 11), the function uses a nonce to overcome a potential signature replay attack (line 8-10). This mechanism will be explained in more detail in Sect. 4.5.3. After that, the *authnz*() function checks if the signer has the required role through the *hasRole*() function (lines 4-6), by accessing the content of the *_accessControl* mapping. As discussed in the previous section, the proposed solution employs a hierarchical access control mechanism rather than a flat permission model that maps user addresses and roles to specific contract addresses. This defines clear access boundaries for roles at each level of the hierarchy. When a user attempts to access a contract, the system verifies the user's role and determines whether the user is authorized to access the specific contract within the hierarchy.

The *revokeRole*() function (lines 16-21) can dynamically revoke roles from users and could be invoked only by the ADMIN to remove any role different from itself. This limitation is necessary to avoid losing control over the hierarchy. The function accepts two parameters: a role *r* and a user address *u*, and revokes the role privilege *r* for the previously assigned smart contracts. The function initially verifies that the user's address is valid (i.e., nonzero) and not the administrator's address (lines 18-20). Then, it removes role *r* from user *u* in the access control mapping. This function provides an additional layer of control for ADMIN, enabling effective role management and maintaining the integrity of ACM in a dynamic environment where user permissions may need to be updated frequently.

The ACM incorporates also a function to transfer its ownership, namely the transfer of the ADMIN role. The *transferOwnership*() function can only be invoked by the current ADMIN and transfers the ownership to a new ADMIN (lines 22-32). That function initially verifies if the new ADMIN address is valid (nonzero) to ensure that ownership is transferred to a valid address and that the new address differs from the previous one. Then, all the contracts associated with the old ADMIN role are transferred to the new ADMIN in the *_accessControl* mapping. The *transferOwnership*() function is critical for ensuring the security of ACM, particularly in the case of a compromised or exposed ADMIN private key. Indeed, in this case, an attacker could take control of the entire hierarchy and act on any contract or factory within the tree. Therefore, its usage could be carefully evaluated and eventually excluded from the ACM implementation. Finally, the *addVerifier*() function allows the ACM to add read-only permissions to external contracts by managing the VERIFIER roles associated with a specific contract.

The multirole authentication and authorization framework not only ensures secure access to smart contracts within the hierarchy and role-specific permissions but also preserves the system's decentralized, trustless nature, which is crucial for sensitive infrastructure, such as water management applications. The next section will discuss the implementation details of the proposed solution and evaluate it from both performance and security perspectives.

4.5 Implementation and Evaluation

In this study, the Ethereum Sepolia test network is used to evaluate the proposed Hierarchical Factory Pattern in a more realistic environment, using testnet Ethers. Smart contracts are developed in Solidity 0.8.40 using the Hardhat deployment environment. The MetaMask wallet is used to interact with the blockchain, deploy smart contracts, and execute transactions. Finally, the DApp layer has been developed using React.js and Ether.js libraries. Tab. 4.1 summarizes all the tools and technologies used; the source code is available in a GitHub repository[2].

Table 4.1: Tools and Technologies Used in the Experimental Setup

Technology	**Version/Network**
Ethereum Test Network	Sepolia
Solidity	0.8.40
Contract Deployment	Hardhat
Blockchain Interaction	MetaMask Wallet
DApp Development	Ether.js
Frontend Framework	React TypeScript
Bytecode Optimizer	true

The following section discusses the implementation of the proposed hierarchical factory pattern from different perspective: the optimization of the developed smart contracts (Sect. 4.5.1), the evaluation of their performances with respect to bytecode size and gas costs (Sect. 4.5.2), and the ability of the solution to overcome two well-known security attacks: the signature replay attack, and the dynamic revocation and transfer of ownership (Sect. 4.5.3).

4.5.1 Optimization of Smart Contracts

Optimizing smart contract code is a critical aspect of any real-world decentralized application, as it affects both deployment gas costs and the overall bytecode size, which must comply with predefined limits. Indeed, the gas cost G_i for the deployment of a contract c_i could be computed as $G_i = G_{\text{base}} + n_i \times G_{\text{byte}}$, where G_{base} is the base transaction cost, n_i is the bytecode size of c_i, and G_{byte} is the cost per byte. Therefore, the bytecode size n_i significantly affects G_i, and smart contracts must be properly designed to reduce it.

The proposed solution addressed this problem from several perspectives. Firstly, the proposed hierarchical factory pattern itself reduces bytecode size and increases scalability by isolating common parts within the factory and allowing them to be called by all other factories or smart contracts in the hierarchy. Sect. 4.5.2 will experimentally analyze this aspect by comparing the gas costs induced by the proposed

[2]https://github.com/MuhammadBinSaif/Extension-Hierarchical-Factory-Pattern-in-Smart-Contracts

solution with those of a flat solution that does not use factories as the total number of deployed contracts increases.

Additionally, an efficient approach is to use libraries to isolate and reuse behaviors. In particular, the WMSFactory in Alg. 2 could exploit methods included in some libraries to create the various subfactories within the *createClientContract*() function. Specifically, when compiled monolithically, the WMSFactory bytecode exceeds the 24 KB (24576 bytes) contract size limit imposed by Ethereum to protect against DoS attacks. Since the library code is stored in a separate contract, the size of the WMSFactory is significantly reduced below the 24 KB limit, decreasing the cost of deploying and interacting with the contract.

The code has also been optimized by replacing standard *require* statements with custom error handling introduced in Solidity version 0.8.4. The *require* statement, which stores the entire revert reason as a string, increases bytecode size and gas consumption. Custom errors enable the definition of typed error codes that consume significantly less gas when reverting. This minimizes the size of the deployed bytecode and the cost of transaction reverts, particularly in functions secured by role checks, which are frequently triggered in hierarchical contracts. In implementing the proposed solution, the *require* keyword is replaced with *custom errors* in all access checks, reducing the deployed bytecode by approximately 134-446 bytes per contract. This corresponds to a decrease of 4.3% to 6.3%, resulting in direct gas savings during both deployment and reversion operations. These reductions are particularly important for role-protected modifiers and other security-critical checks, which are frequently used in hierarchical contracts.

Finally, in smart contract optimization, strategic ordering of state variables can significantly reduce gas costs. The EVM allocates state variables in storage slots, where each slot occupies a 32-byte. Moreover, it sometimes inserts padding after large types to maintain consistent alignment, especially when a small type (such as an enumeration, which is 1 byte) follows multiple large types (such as strings, which are 20 bytes each). It follows that with the organization of the *ContractData* structure as presented in Alg. 7 for our specific application, the structure will occupy 32 bytes $\times$ 3 for storing the first three addresses and an additional slot of 32 bytes is needed to store the last enumeration attribute, even if it requires only 1 byte, for a total of 128 bytes.

Algorithm 7: Contract Data structure

```
1 struct ContractData :
2     address _contractAddress;
3     address _ownerAddress;
4     address _parentContractAddress;
5     ContractType _contractType;
```

4.5.2 Performance Evaluation

The performance of the proposed solution is evaluated with a particular focus on two critical aspects: the bytecode size of the deployed contracts and the gas cost

required for contract execution and deployment. These two metrics are crucial for evaluating the efficiency and resource allocation of the proposed pattern within an Ethereum network. In particular, bytecode size measures the contract's complexity and the amount of code processed by the EVM, while gas consumption reflects the computational resources required for contract deployment and execution. This evaluation compares the proposed hierarchical solution with a traditional factory pattern. Moreover, both the traditional L1 Ethereum EVM and an L2 solution, represented by the Polygon zkEVM, are considered to evaluate the solution's feasibility in real-world application domains.

Table 4.2: Bytecode Size and Estimated Gas Cost of Smart Contract.

Smart Contract	**Bytecode size (bytes)**	**Estimated Gas used (gwei)**
ACM	7,452	1,619,208
WMS Factory	1,607	374,400
City Factory	7,373	1,530,644
District Factory	4,789	1,291,439
Station Contract	2,003	932,240

To better study the effect of the hierarchical structure, the structure of a water management system serving the municipality of Verona, a city in northern Italy, is developed. In the hierarchy, the top-level CityFactory oversees 12 *DistrictFactories*, each of which handles 7 to 15 *StationContracts*, totaling 124 stations. A three-tier structure (i.e., city, district, station) is developed based on the roles described in Sect. 4.1, focusing on the key operations of role checks, contract deployment, and data aggregation. The full hierarchy is deployed on the Sepolia testnet and measures bytecode size and gas for deployment and typical management operations (role assignment, station registration, yearly and monthly data uploads). Tab. 4.2 illustrates the gas consumed during deployment and the bytecode size of each factory in the hierarchy, including the ACM, the WMS Factory, and the three levels: City, District, and Station. It can be observed that the bytecode size of all proposed smart contracts is below the EVM limit of 24 KB, set to mitigate security attacks. Additionally, the size of each factory decreases as the depth in the hierarchy increases.

As shown in Fig. 4.5, the traditional factory pattern generates an identical bytecode size of 7,711 bytes for each level CityFactory, DistrictFactory, and StationContract. This is because each contract independently includes the full logic required for contract creation and access control. In contrast, the hierarchical factory pattern concentrates the main logic at the highest level, resulting in smaller bytecode sizes at lower levels. Particularly, CityFactory remains relatively large at 7,373 bytes since it contains references to all subordinate contracts, while DistrictFactory and StationContract sizes reduce to 4,789 bytes and 2,003 bytes, respectively. Therefore, by centralizing common functionality at the root and minimizing redundancy, the hierarchical approach reduces overall bytecode size compared to the traditional factory approach.

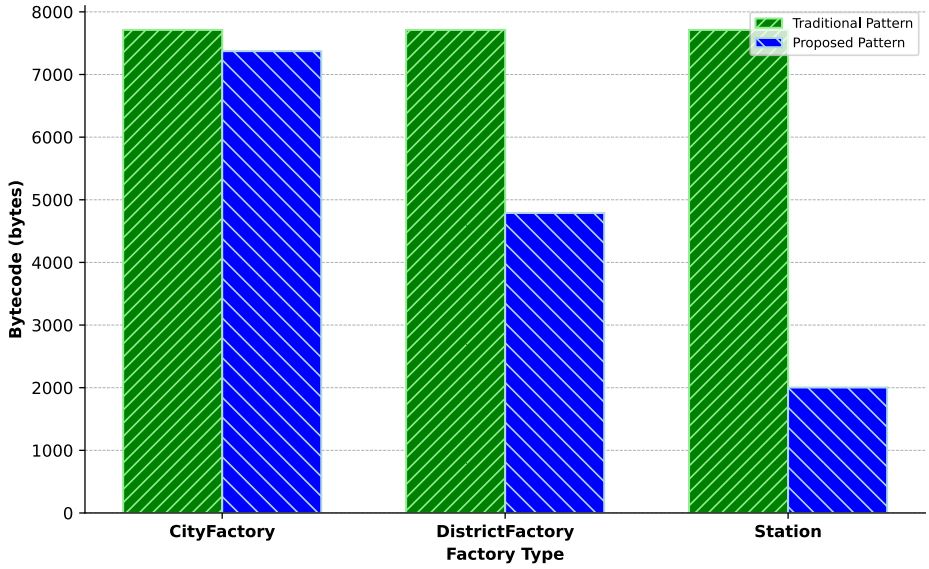


Figure 4.5: Bytecode comparison of the proposed vs the traditional factory pattern.

The cost in currency for different operations is measured by converting the amount of *GasUsed* into USD using the unit Gas price for two networks: the Ethereum mainnet and the Polygon zkEVM, a Layer 2 solution. At the time of the experiment, the median base-fee for Ethereum mainnet was 1.382 gwei at an exchange rate of 1 ETH = 1733.68 USD, and for Polygon zkEVM, the base-fee was 513 gwei and 1 POL = 0.2257 USD. The conversion is performed with the following formula:

$$\text{Cost}_{\text{USD}} = \mathit{GasUsed} \times \mathit{GasPrice}_{\text{gwei}} \times 10^{-9} \times \mathit{tokenUSD}$$

Fig. 4.6 shows the deployment cost of our three-layer hierarchy (i.e., CityFactory, DistrictFactory, and StationContract), and compares it with the deployment cost induced by a conventional single-factory pattern, on both the Ethereum net and the Polygon one. As can be observed, the proposed solution makes the CityFactory deployment approximately 8% more expensive than a traditional factory, since the contract includes both access-control logic and shared libraries. However, this extra cost decreases at the DistrictFactory level, where our approach is 10% cheaper, and at the StationFactory level, which accounts for most of our contracts, it is 34% cheaper. Indeed, the traditional pattern requires redeploying a whole factory (about 3.0$ on the Ethereum mainnet) for each new district or station, repeating byte-code and role checks. In contrast, the hierarchical factory pattern reuses parent logic and deploys only a light child contract. Since the DistrictFactory and the StationContracts are the most frequently added or interacted with, a one-third cost reduction at this level directly reduces operational costs for water network operators. Regarding the comparison between the costs on the Ethereum and Polygon networks, we can observe that a significant cost reduction can be achieved on the former, where general operational costs are higher, and achieving such a reduction is particularly important.

Fig. 4.7 compares the USD cost of four function calls *addVerifier*(), *revokeUser*(), *transferOwnership*(), and *addData*() on both the Ethereum mainnet and the Polygon

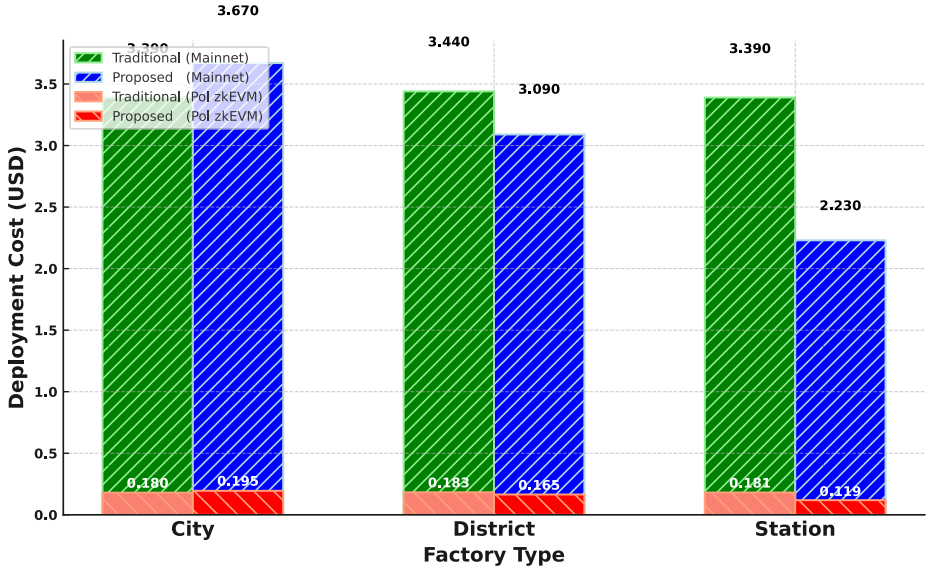


Figure 4.6: Comparison of the USD costs needed for the contract deployment in the proposed hierarchical factory pattern approach vs the traditional factory pattern.

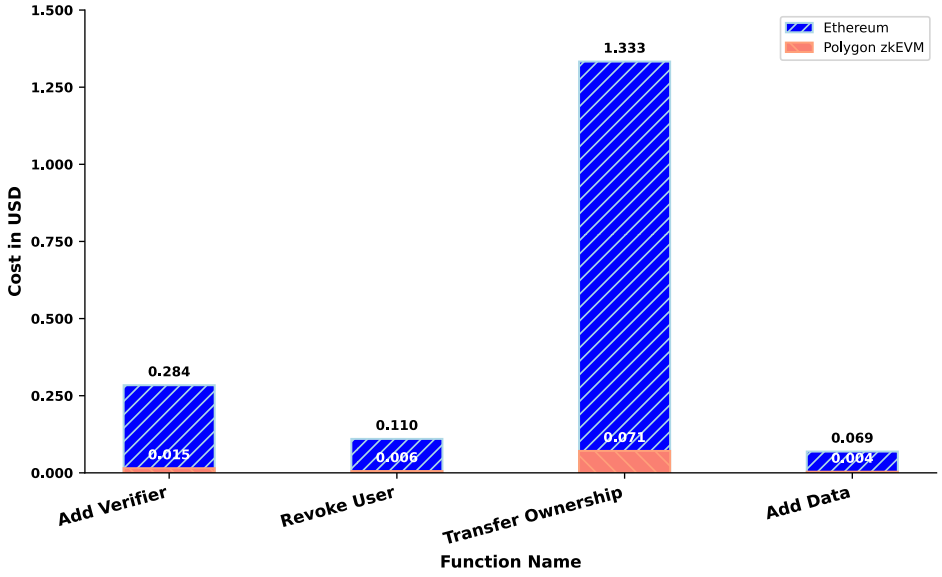


Figure 4.7: USD cost of function calls.

zkEVM. On Ethereum mainnet, the *transferOwnership*() function is the most expensive operation, costing 1.33$, while the simple data upload (i.e., *addData*()) requires only 0.07$. The two remaining access-control calls cost 0.28$ for *addVerifier*() and 0.11$ for *revokeUser*(). Executing the same functions on Polygon zkEVM significantly reduces cost, such as *transferOwnership*() drops to 0.07$, *addVerifier*() to 0.02$, *revokeUser*() to 0.006$, and *addData*() to a negligible 0.004$. It can also be observed that while the first three functions are rarely called, since they deal with the grant and revoke of user roles and permissions, the last one (i.e., *addData*()) is the most frequently used in a real-world application, since it is used to store the water

management data. The cost of its invocation is much lower than that of the other three, and when deployed in an L2 solution, the fees are also economically feasible for frequent invocations.

Finally, Fig. 4.8 presents a detailed comparison of gas costs for user management operations with the ACM smart contract with respect to the results obtained by applying the solutions in [108] and [23]. In this case, two types of operations have been considered: adding a new user and removing an existing one. For the first operation, different amounts of informative data were passed to the method. The results show that the proposed approach outperforms both baselines even if the gas cost increases with the amount of data processed by *addUser*(). These results confirm the efficient behavior of the proposed approach for the *removeUser*() method as well.

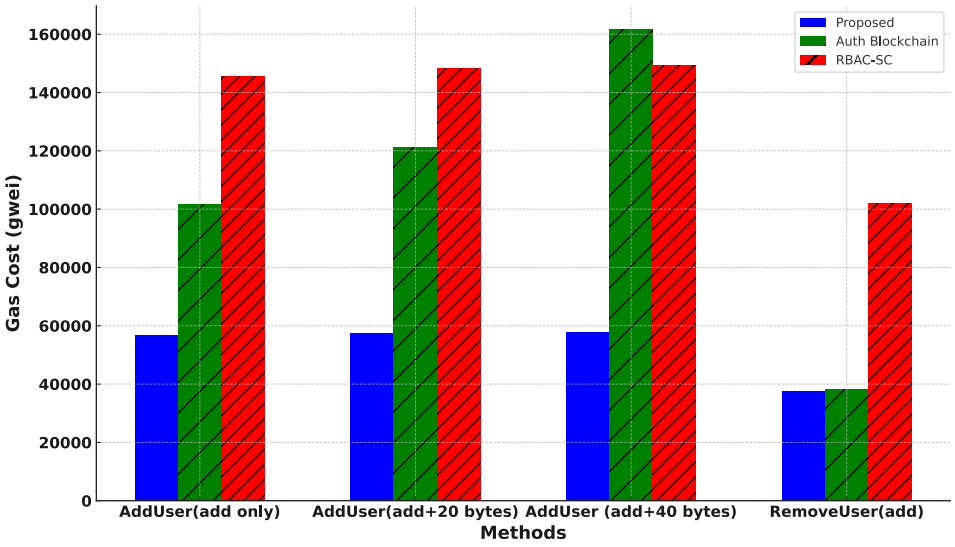


Figure 4.8: Gas cost comparison of the proposed approach vs [108] and [23].

4.5.3 Security Evaluation

The security evaluation of the proposed solution is grounded in a defined threat model. We consider three adversaries: an *external unauthorized user*, who attempts to invoke restricted functions without the required role; a *compromised role holder*, whose exposed private key may enable privilege escalation or unauthorized contract instantiation; and a *malicious factory interaction*, in which a rogue contract is deployed or registered outside the intended ACM-governed hierarchy. The assets to protect are the integrity of contract deployment, the consistency of role propagation, and the ownership of privileged contracts. We assume that the execution of the blockchain and the underlying cryptographic primitives are correct, while external users and non-administrative keys may be adversarial. The following discussion evaluates how the proposed design mitigates these threats.

In the ***signature replay attack*** [107], a malicious user reuses a previously valid signature to illegitimately re-execute an operation. This kind of attack can affect the *authnz*() function of the ACM (see Alg. 6, lines 7-15) which is responsible for user

authentication and authorization. To prevent this form of attack, the proposed system incorporates a nonce-based security approach into the authentication process. In particular, each transaction invoking *authnz*() passes to it a nonce value, which is combined with the signed data to generate a unique hash (lines 9). The resulting hash is then checked against the *executedNonce* mapping, which records previously used signature hashes. The transaction is reverted immediately if the hash already exists, preventing signature reuse. This approach ensures that every signed message is valid for a single execution, preventing it from being replayed in function calls or contract contexts. Furthermore, the integration of the nonce argument in factory deployment and access control functions, particularly the *addContractOnBehalfOf*() and *addContract*() functions in Alg. 1, guarantees the constant mitigation of replay attacks in all tiers of the hierarchical factory pattern. This is crucial for the security of critical infrastructure applications, such as water management systems, since unauthorized access could compromise the integrity and confidence of the entire operating hierarchy.

Regarding the ***dynamic revocation and transfer of ownership*** attack, it can be observed that the integrity and security of the entire hierarchy depend entirely on the validity and security of the accounts authorized to manage contracts within it. However, in many real-world applications, users may leave an organization or change their role within it, or in some cases, their account could be compromised, requiring new authentication credentials to be registered. To address both situations, the proposed solution provides a functionality for the ADMIN to dynamically revoke user roles, particularly in the case of a compromised account or wallet. In particular, the ADMIN can revoke any role assigned to a user in all previously authorized contracts by using the *revokeRole()* function. The system does not allow the revocation of the ADMIN role; however, if the private key associated with the ADMIN account is compromised, the owner can, as a last resort, invoke the *transferOwnership()* function to reassign administrative privileges to a new address securely. It can be seen that, in this case, if the ADMIN private key is stolen, a malicious user could potentially reassign all the hierarchy to himself before the administrative role is properly reassigned. To further enhance security, the ADMIN role can also be connected to a multisignature (multisig) wallet, thereby requiring multiple user approvals to authorize sensitive actions such as the transfer of ownership, without changing the structure of the proposed smart contract, while reducing the risk associated with a single compromised key. However, this solution requires more gas per ADMIN operation and can be chosen transparently based on the application domain. By enabling dynamic role revocation and ownership transfer, the system can adapt quickly to potential security threats or structural changes without redeploying the smart contract hierarchy. This ensures that the system remains robust and operational under various circumstances.

4.6 Chapter Summary

This chapter addressed the design challenges in developing and deploying smart contracts, focusing on scalability, modularity, security, and manageability. These challenges highlight the limitations of traditional approaches and the need for more

advanced design methodologies that align with the requirements of complex decentralized applications. The factory pattern has been introduced in the literature as a classical solution for improving modularity and reducing deployment costs. While effective at replicating multiple instances of a single contract, it proved insufficient for heterogeneous or hierarchically organized applications due to its structural rigidity and lack of integrated access control. This motivated the exploration of an extended design pattern capable of supporting layered organizations and dynamic governance. The hierarchical factory pattern has been presented in this chapter as an evolution of the classical approach. By enabling recursive instantiation of factories and contracts, this architecture mirrors the multi-level organization of real-world infrastructures, such as water management systems, and introduces scalability, modularity, and maintainability across multiple layers. To complement these structural improvements, integrated multirole authentication and authorization were introduced, ensuring that access control becomes an intrinsic component of the system rather than an external mechanism.

Implementation of the proposed model in Solidity confirmed its feasibility and efficiency. Empirical evaluation demonstrated that while initial deployment at the upper layers introduces a small overhead, cumulative savings at the lower layers lead to significant cost reductions, particularly when combined with L2 solutions. Security testing further validated the model's resilience against replay attacks, privilege escalation, and unauthorized access. In conclusion, the hierarchical factory pattern with integrated MAC offers a structured and secure approach to smart contract design. It not only addresses the inherent challenges of scalability, modularity, and security but also demonstrates practical efficiency and robustness when deployed in real-world infrastructures. While water management represents the primary evaluation scenario in this chapter, the hierarchical factory pattern is not restricted to this domain. The proposed architecture applies to any multi-level organizational structure that requires coordinated contract deployment and role-differentiated access control. In supply chain management, for instance, tiers of manufacturers, distributors, and retailers follow a natural hierarchy in which parent entities govern and audit the operations of subordinates. In healthcare, data access rights are typically organized across hospitals, departments, and individual practitioners in a similar layered design. Identity management infrastructures, where a root authority delegates credential issuance to regional registrars, represent another direct application of the ACM-factory model. In each of these cases, the ability to configure roles and hierarchy depth at design time while preserving dynamic user assignment at runtime makes the proposed pattern applicable to a broad class of hierarchical decentralized applications. This chapter has therefore established a solid foundation for exploring how advanced design patterns and access-control mechanisms can strengthen decentralized applications, paving the way for the subsequent chapters of this thesis.

Chapter 5

Privacy Preserving Decentralized Data Storage and Retrieval

To address the limitations of blockchain for managing large volumes of data, this thesis develops a decentralized storage framework that integrates blockchain, the IPFS, and encryption. While blockchain ensures immutability and traceability, it is unsuitable for storing raw information at scale due to high costs and limited throughput. In the proposed approach, only cryptographic references are stored on-chain, while the actual data is distributed across IPFS nodes, providing a more scalable storage architecture. However, using IPFS alone does not guarantee confidentiality or controlled access. To overcome this challenge, the framework integrates encryption and hashing mechanisms that preserve privacy and integrity. Encryption protects sensitive data stored off-chain, while hashing ensures tamper detection without exposing the original content. This dual-layer approach maintains confidentiality during retrieval and querying while enabling transparent verification through blockchain anchors. By integrating blockchain, IPFS, and encryption, this contribution provides a practical solution that guarantees immutability, availability, traceability, and privacy in distributed storage systems. Based on this contribution, the chapter first contextualizes the problem with reference to the motivating example introduced in Chap. 4, then formalizes the storage problem and desired properties, describes the system architecture and the combined hashing and encryption workflow, and finally evaluates the solution in terms of efficiency, scalability, and security.

5.1 Motivating example

Let us consider the motivating example introduced in Sect. 4.1 about the development of a blockchain-based water management information system.

As described in Chap. 4, a water management system usually divides a city into multiple administrative districts in a hierarchical manner, as illustrated in Fig. 5.1. Each district contains several monitoring stations, each operating independently and consisting of multiple water pumps to extract water from the ground, sensors to monitor water quality (e.g., pH, turbidity, etc.), and quantity (e.g., extraction rate, loss). Indeed, while a pump extracts water from the ground, several onsite IoT sensors collect essential data on water quality and quantity. These pieces of data are

collected through a Message Queuing Telemetry Transport (MQTT) broker [109], which stores the received data in a centralized storage, such as a relational database or a set of log files. The initial storage of data produced by IoT devices in a centralized system is driven by performance and scalability considerations, as water monitoring stations continuously generate high-frequency data streams, which would be inefficient and costly to process or store directly on the blockchain. In this scenario, centralized storage allows for efficient data ingestion and temporary buffering. Moreover, the collected data contains sensitive information that cannot be stored directly on public decentralized networks like IPFS without compromising privacy. Therefore, the actual sensor data must remain off-chain under controlled access, while only cryptographic references and verification metadata are anchored on the blockchain to ensure integrity, traceability, and auditability. Consequently, the fundamental challenge in this scenario is to implement this architecture to make the information immutable and tamper-proof, while ensuring its availability and strictly maintaining the privacy of sensitive information.

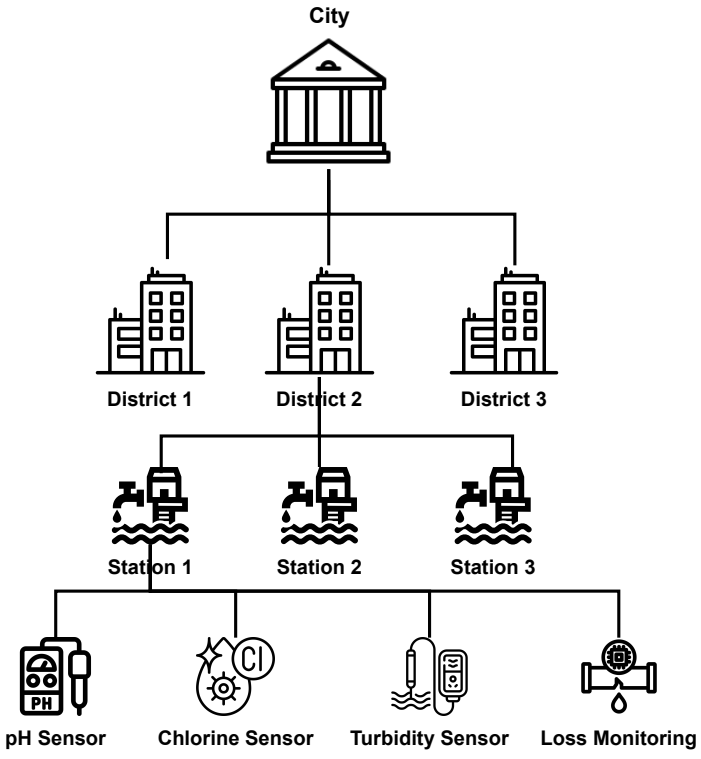


Figure 5.1: Hierarchical organization of a typical water management system.

The following section formalizes the desired properties while Sect. 5.3 illustrates the proposed solution, which integrates blockchain, smart contract technology, and the IPFS decentralized storage.

5.2 Problem Statement and Formalization

In a typical IoT scenario, such as the one introduced in the motivating example of Sect. 5.1, several IoT devices or sensors generate a set of data daily that needs to be stored and certified to ensure immutability, traceability, and tamper-proofing. In this scenario, it is crucial that the generated data be stored permanently, without any subsequent modification or deletion. Given such premises, it can be formalized that the following properties are what one wants to ensure by using blockchain and smart contract technology in conjunction with the IPFS protocol.

Proposition 5.1 (Data immutability)**.** *Given a piece of data d stored in a database or file system, let's assume d is immutable if it cannot be modified after storage, or that any subsequent modification of d can be easily identified.*

Blockchain technology ensures the immutability of data stored inside its blocks. Indeed, block confirmation ensures that no further data modifications are made to the block. Typically, the amount of data stored on the blockchain is minimal due to technological constraints and cost considerations. However, blockchain can ensure the immutability of data stored off-chain by essentially storing a fingerprint (or hash) of the data. Such a fingerprint is enough to determine whether information stored outside the blockchain has been modified. In many real-world scenarios, immutability can be considered too strict, as data may need to be updated or rectified due to missing or erroneous parts. Therefore, immutability, together with another property called traceability, is considered.

Proposition 5.2 (Data traceability)**.** *Data traceability is the ability to trace data transformations back to their origins, verifying the authenticity and integrity of the data. It can also be interpreted as the degree to which a system or data provider can accurately record the changes made to the data.*

Similar to the previous property, this one can also be achieved by properly implementing a set of functionalities using smart contracts and storing fingerprints or hashes of data, enabling easy, optimal integrity checks. Combined with immutability, traceability can lead to a more robust system in which each stage of the data life cycle is stored immutably. Instead of preventing data modification, these two properties together allow the registration of multiple subsequent data evolutions, similar to a versioning system. Blockchain has emerged as a potential solution for implementing data traceability by creating an information trail that ensures security and data availability [110].

Besides these two properties, which concern data quality, the next property concerns data availability and the information that certifies data quality. It ensures stakeholders that the information they are interested in will always remain accessible and valid without relying on the activity of a specific company or organization. The last property assumes a very different meaning depending on whether the data is stored on-chain or off-chain.

Proposition 5.3 (Data availability)**.** *Data availability refers to a user's confidence that the stored data and all the information required to verify specific properties of that (such as immutability) are available in a given period.*

The following section outlines a system architecture based on blockchain, smart contracts, and IPFS, which enables achieving all these properties in a secure and efficient manner.

5.3 System Architecture

Fig. 5.2 illustrates the general architecture of the proposed solution. It includes, at its core, a decentralized application (dApp) that interacts with traditional centralized storage to retrieve data to be persisted, making it immutable through the blockchain using a set of smart contracts and IPFS. It is essential to note that the primary objective is to store data efficiently on IPFS rather than directly in a blockchain, as this can be costly and inefficient. However, since the data is sensitive and subject to privacy concerns, it is encrypted before being stored on IPFS. At the same time, the data hashing technique generates a unique fingerprint for each row, enabling rapid detection of data manipulation without requiring decryption. Moreover, IPFS returns a CID upon storing data, and that CID is permanently stored in the blockchain for traceability. The CID is a unique identifier for the data and essentially a hash of the data stored on IPFS. In a centralized web service, content or data can be accessed via location-based uniform resource locators (URLs). In IPFS, content or data is stored and retrieved using hashes, which makes it more challenging to censor or manipulate. Moreover, there is no need for trust in IPFS data hosting entities.

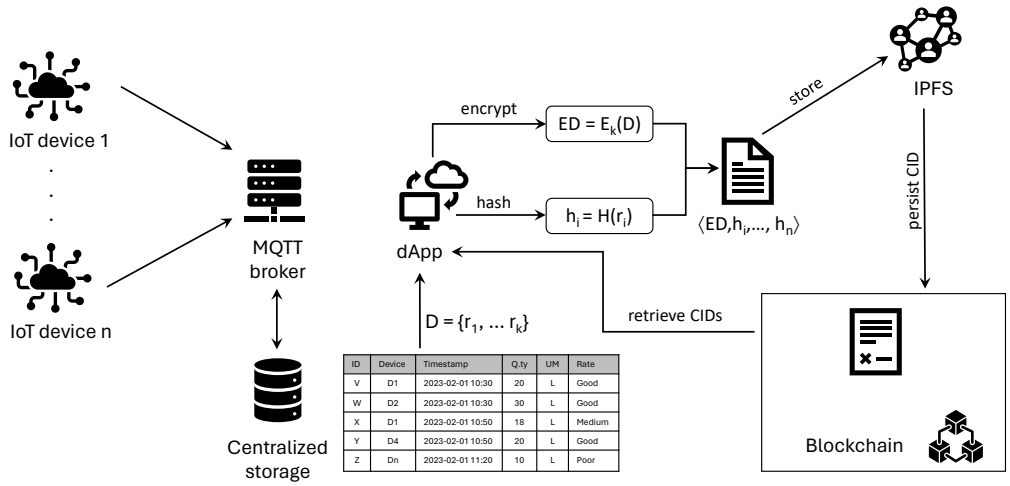


Figure 5.2: The architecture of the proposed solution.

In the future, any modification can be detected by simply comparing the previously stored hash on IPFS with the newly generated hashes of the modified data. Consequently, this approach eliminates the need for decryption, which often requires revealing private keys and poses security risks, and is therefore limited to cases in which previously stored data needs to be effectively restored. Moreover, this scheme allows for the easy separation of concerns, as an integrity check can be performed without revealing the content. Compared with existing blockchain and IPFS storage approaches, the novelty of the proposed solution does not lie in the isolated use of IPFS, encryption, or blockchain anchoring, as these components are already well

established in the literature. Instead, the novelty is in computing the hash of each record before encryption and storing it alongside the encrypted payload on IPFS. Any integrity check, therefore, relies on comparing newly computed hashes with the hash values retrieved from the CID-anchored on-chain data, without decrypting the protected data or exposing the secret key. As a result, decryption is limited to cases in which the original plaintext must be explicitly restored, reducing the attack surface associated with key exposure to an exceptional rather than a routine operation. In this way, the proposed solution achieves data privacy, integrity, and query optimization by properly integrating data hashing and encryption, as described in the following section.

5.4 Hashing and Encryption

Alg. 8 illustrates how the data from the centralized storage is prepared for storage in IPFS. The function CreateDataForIpfs receives as input a list of records corresponding to the database rows in Fig. 5.2. For each row, the function computes the hash of the row content (see line 4) and includes the tuples $\langle id_i, H(record_i)\rangle$ in a list, where $H()$ represents the application of the hashing function. After that, the function encrypts the overall content using the $E()$ function (see line 5). The content stored on the IPFS is the union of these two elements, as reported in line 6.

Algorithm 8: Creation of the data to be stored in IPFS

1 **Procedure** *CreateDataForIpfs*$(rawData = \{\langle id_i, record_i\rangle\}_{i=1}^{n})$ **:**
2 $\quad hashedData \leftarrow \{\}$;
3 $\quad$ **foreach** $d \in rawData$ **do**
4 $\quad\quad hashedData \leftarrow hashedData \cup \{\langle id_i, H(record_i)\rangle\}$;
5 $\quad encryptedData \leftarrow E(rawData)$;
6 $\quad$ **return** $combinedData = \{hashedData, encryptedData\}$;

The SHA-256 algorithm [111] is employed for hashing, which maps the given data to a 256-bit fixed-size cryptographic hash.

Definition 5.1 (Hashing)**.** Let D be the data entry and H be the hash function implementing the SHA-256 algorithm. The hashing process can be represented as:

$$H(D) = h \tag{5.1}$$

where h is the unique hash result.

The characteristic of a hash is that each unique input data produces a unique hash value. Even minor changes in the input data result in a significant change in the hash output. This property of hashing is known as the avalanche effect.

For encryption, the proposed solution employs the Password-Based Key Derivation Function 2 (PBKDF2) [112], a key derivation function that incurs a sliding computational cost to mitigate vulnerability to brute-force attacks. It generates a cryptographic key from a secret, eliminating the need to store the encryption secret in a centralized database or server and thereby enhancing security.

Definition 5.2 (Key Derivation)**.** Let P represent the user-provided secret and KDF the key derivation function. The key K used for the encryption and decryption process can be derived as follows:

$$K = KDF(P, s, c, dkLen) \tag{5.2}$$

where s is the salt, c is the number of iterations, and $dKLen$ is the desired key length.

One of the novelties of the proposed solution is that it does not store the encryption secret or keys. Instead, it encrypts a known string S with the secret key and stores the resulting encrypted string on IPFS. Alg. 9 illustrates the process of verifying a secret without storing the secret or keys anywhere. Given the secret P' provided by the user, the corresponding key K' is derived using the KDF function. Then, the encrypted string $E(S)$ retrieved from IPFS is decrypted by using K'. If the obtained decrypted string S' corresponds to the original string S, then the provided secret is correct, and the verification is successful; otherwise, the verification fails.

Algorithm 9: Verification of a secret S without revealing it

```
1 Procedure VerifySecret(P') :
2     Retrieve the encrypted string E(S) from IPFS;
3     K' ← KDF(P', s, c, dkLen);
4     S' ← D(E(S), K');
5     if S' = S then
6         the secret is correct;
7     else
8         the secret is incorrect;
```

If the secret verification is successful, the data is encrypted using the derived key K', as reported in line 5 of Alg. 8. The encrypted data is then combined with the hashed data inside a JSON object (see line 6 of Alg. 8).

Definition 5.3 (Encryption)**.** Let E represent the encryption function, K the key derived from the secret, and D the data. The encrypted data ED can be obtained as follows:

$$ED = E(K, D) \tag{5.3}$$

Once the *combinedData* is generated, it is stored on IPFS. IPFS returns a unique CID, which serves as a reference for the encrypted and hashed data stored on IPFS. The blockchain is the final step of the proposed architecture. After storing the encrypted data and the hashed data on IPFS and obtaining the corresponding CID, this CID is stored on the blockchain through a smart contract. Such a smart contract will maintain a list of CIDs related to the data stored on IPFS. The immutability of blockchain ensures that the CID is recorded and cannot be changed or removed, providing an auditable record of data integrity and traceability. The primary purpose of this smart contract is not only to store CIDs permanently but also to retrieve and verify the persisted data.

Alg. 10 illustrates the function DataVerification, which performs an integrity check about the current content of the centralized database and what has already been stored in the IPFS. After retrieving both contents, D_{api} and D_b, respectively, the function analyzes whether each row in the database is already contained in the blockchain and, in this case, whether the hashes match. The newly identified records are then stored in IPFS, as illustrated in Alg. 8, and a transaction is performed to store the corresponding new CID. Conversely, records whose hashes do not match are classified as corrupted or inconsistent. This consistency verification does not require decrypting the stored data; instead, it relies solely on comparing the hashes. However, based on the system's specific policies, in the event of corrupted or inconsistent data, the original version could be retrieved eventually through decryption of the encrypted data contained at the corresponding CID.

Algorithm 10: Data retrieval and verification

```
1  Procedure DataVerification :
2  D_b ← retrieve from IPFS the documents with the stored CIDs;
3  M_b ← map from the hashed data in D_b where key = id_i and value = H(r_i);
4  D_api ← retrieve from the API the data stored in the database;
5  foreach r ∈ D_api do
6      id ← identifier of the current record r;
7      h_new ← H(r);
8      h ← M_b.get(id);
9      if h is null then
10         the data entry is new and needs to be persisted in the blockchain;
11     if h == h_new then
12         the data entry has not been modified;
13     else
14         the data entry has been modified;
```

The proposed solution enables efficient, secure, continuous monitoring of data updates. The system's privacy and security are enhanced by not decrypting sensitive data; instead, it compares hashes. Furthermore, by employing blockchain as an immutable storage for comparisons, it ensures that the verification process is tamper-proof and auditable. The robust integration of cryptographic hashing and encryption, IPFS, and blockchain enables a sophisticated solution that ensures data integrity, enhances transparency, and maintains confidentiality in a public network. The next section describes the implementation of such a solution and evaluates it.

5.5 Implementation and Evaluation

This section presents the proposed solution implementation, focusing primarily on testing the system's performance and security across various data sizes and security vulnerabilities. The experiments were performed on a computer equipped with an

Intel Core i5 CPU at 1.00 GHz and 8 GB of RAM. A dApp has been developed in React[1] using the TypeScript[2] for cryptographic functionalities, and the ipfs-http-client (version 60.0.1) for IPFS decentralized storage interactions. Additionally, the AES algorithm is employed for encryption and decryption, utilizing a 256-bit key derived from a secret key. To ensure data integrity, the SHA-256 hashing algorithm is used. The proposed solution uses IPFS for data storage and retrieval, with Infura as the IPFS node provider and the Ethereum blockchain to store IPFS CIDs, ensuring robust, decentralized storage. Table 5.1 presents a detailed description of the experimental setup. The source code of the solutions has been made available through a GitHub repository [3].

Table 5.1: Experimental Setup Specifications

Specification	**Details**
Front-end Framework	React 18.2.0
Programming Language	TypeScript
Encryption Library	CryptoJS 4.2.0
IPFS Client	ipfs-http-client 60.0.1
Encryption Algorithm	AES
Key Generation Method	PBKDF2
Hashing Algorithm	SHA-256
IPFS Node Provider	Infura
Blockchain	Ethereum
Processor	Intel Core i5 CPU 1.00GHz
RAM	8.00 GB

Two different metrics are adopted to measure performance: throughput and total time. The aim is to demonstrate the impact of varying data size on system throughput and the time required to encrypt and decrypt the JSON data. The dummy JSON data (resembling real-life JSON data) is generated using the Faker library[4].

5.5.1 Encryption

The throughput T is measured as the amount of data D processed per unit time t during the encryption and decryption phases. In particular, given the size of the encrypted data $size(ED)$ and of the decrypted data $size(DD)$, the relative throughput can be computed as follows respectively:

$$T_E = \frac{size(ED)}{t} \qquad T_D = \frac{size(DD)}{t} \tag{5.4}$$

The throughput of the proposed solution is tested on a wide range of data sizes from 10 KB to 3000 KB. Fig. 5.3 illustrates that the system's throughput increases

[1] https://react.dev/

[2] https://www.typescriptlang.org/

[3] https://github.com/MuhammadBinSaif/Efficient-and-Secure-Distributed-Data-Storage-and-Retrieval-Using-IPFS-and-Blockchain

[4] https://fakerjs.dev/

as data size increases. Initially, the throughput was around 3.07 kilobytes per second (kb/s) for 10 kilobytes (KB) of data. However, with larger data sizes, such as 3,000 KB, the throughput has increased to over 307.12 kb/s. The throughput decreases for smaller amounts of data due to the significant time required for the initial encryption or decryption steps, as well as the inherent computational inefficiencies at smaller data scales. However, as the data volume increases, these initial steps become less significant in the overall process. These results demonstrate that AES is scalable and efficient in processing large datasets, making it suitable for applications that deal with large data volumes, resulting in secure and efficient data encryption and decryption.

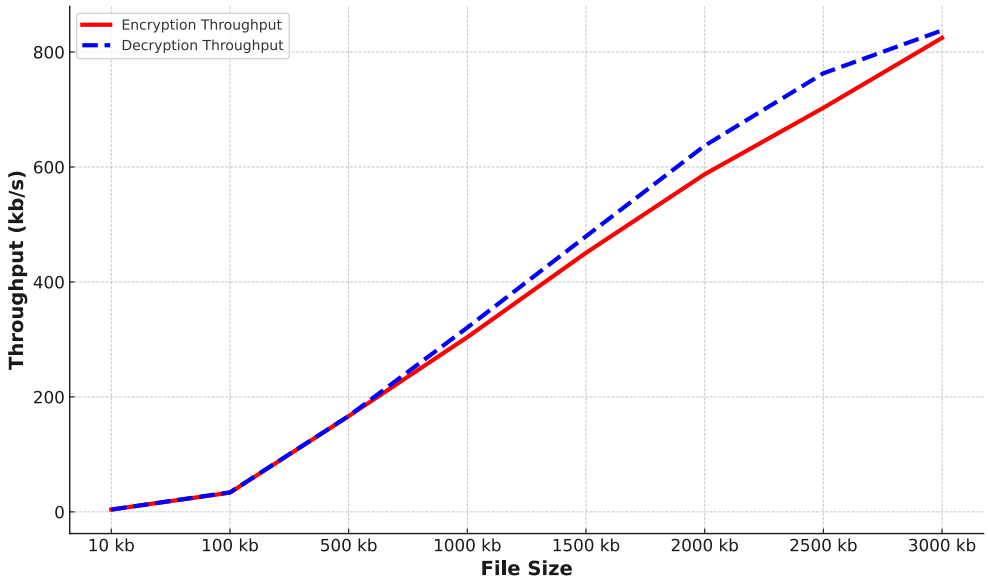


Figure 5.3: Throughput in kb/s.

5.5.2 Total Time

The total time $t = t_e - t_i$ is the difference between the initial time t_i when a process starts and the end time t_e when a process is completed. This section evaluates the total time required for the proposed solution to encrypt, decrypt, generate a hash, and store the encrypted and hashed data on IPFS for data sizes ranging from 10 KB to 3000 KB. Fig. 5.4 illustrates the total time for these processes. The results show that the time per operation increases with data size. The processing times for encryption, decryption, and hashing of 10 KB of data are approximately 2.472 ms, 2.511 ms, and 1 ms, respectively. In contrast, storing 10 KB of data on IPFS took 720 ms. As the data size increases to 3000 KB, an increase in operation times for encryption to 3647 ms, decryption to 3590 ms, hashing to 210 ms, and storing on IPFS to 6252 ms can be observed. The gradual increase in processing time with data size demonstrates the efficiency of the AES and SHA-256 algorithms. However, due to network latency, data propagation, and processing delays, IPFS takes longer to store data. However, this time could also be considered acceptable for real-world

applications, as this persistent activity will be reasonably performed offline, unlike other system functions.

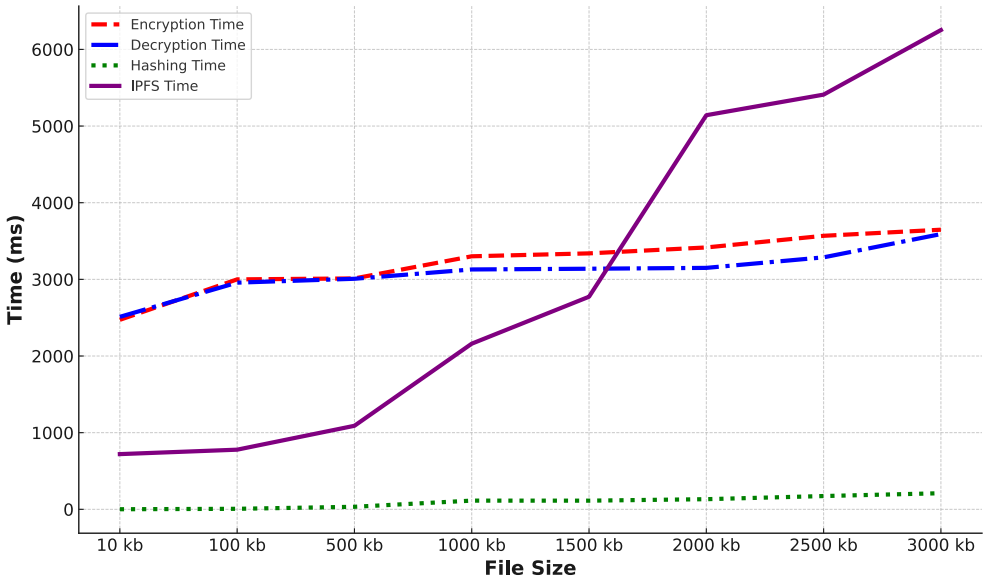


Figure 5.4: Total time in ms.

5.5.3 Scalability

Finally, the scalability of the proposed system is analyzed by measuring the overall throughput across all phases (not just the encryption phase) for data sizes ranging from 10 KB to 3000 KB. Fig. 5.5 shows the overall throughput, including data encryption, hashing, and storing the encrypted and hashed data on IPFS. As shown in the graph, the initial throughput was approximately 3 kb/s for 10 KB of data. However, as the data size increases to 3000 KB, the throughput impressively grows to over 307 kilobytes per second. This demonstrates the robustness of the proposed system in managing and processing large datasets, highlighting its scalability. Therefore, the proposed solution is suitable for adoption in data-intensive applications where scalable security is critical.

5.5.4 Security Evaluation

This section presents a security evaluation of the proposed system. The system's capability to verify data integrity and prevent unauthorized access is evaluated. The proposed system successfully detected both attacks. We consider two main classes of adversaries: an *unauthorized data requester*, who attempts to access encrypted content without the correct secret, and a *tampering adversary*, who modifies data stored off-chain or the corresponding metadata to invalidate integrity guarantees. The main assets to protect are therefore the confidentiality of off-chain data and the integrity of the stored content, together with its blockchain anchor. We assume that blockchain execution and the underlying cryptographic primitives are correct, while

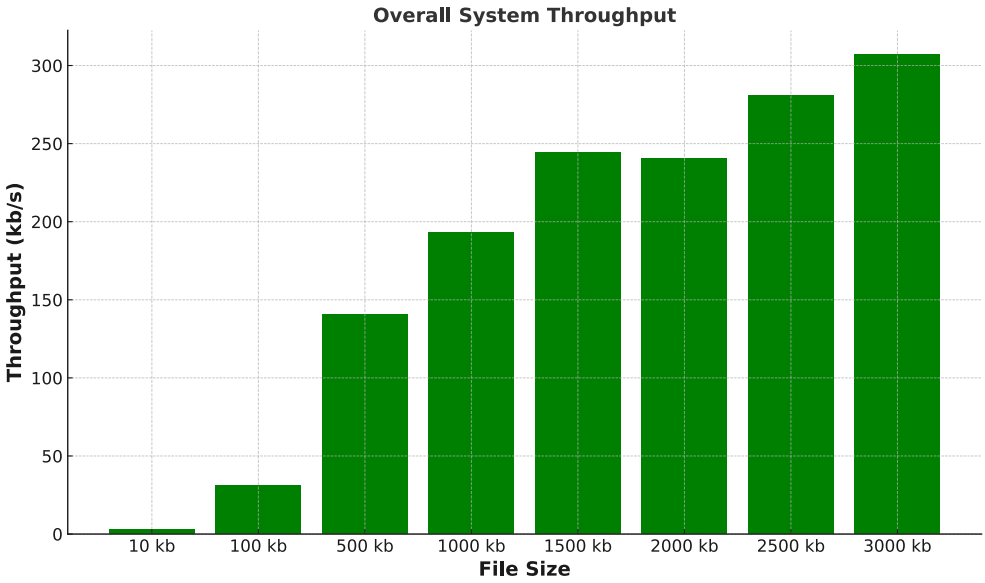


Figure 5.5: Overall throughput in kb/s.

off-chain storage nodes, external requesters, and communication channels may be adversarial. Under these assumptions, the following evaluation examines how the proposed design mitigates unauthorized access and data tampering.

Data Integrity

The primary focus of AES and similar encryption algorithms is to ensure confidentiality. However, these systems do not ensure data integrity, meaning they fail to detect changes to data during transmission or storage. An attacker could modify encrypted data, resulting in incorrect but successful decryption. Therefore, the AES algorithm requires an additional mechanism for data integrity. The proposed solution addresses this data integrity challenge by integrating a hashing mechanism with encryption. The proposed system computes the original data hash, a unique digital fingerprint of data. After decryption, it computes the hash of the decrypted data and compares it with the original hash. If both hashes match, no tampering occurred during the encryption and decryption. The proposed data integrity approach provides robust protection against unauthorized modifications, enhancing the security of stored or transmitted information.

Unauthorized Access

The unauthorized access attack involves accessing the content of encrypted data without a valid decryption key. This attack involves two steps: first, encrypting data securely with a valid secret key, and then attempting to decrypt it with a different, unauthorized secret key. This approach simulates a real-world attack scenario in which an attacker attempts to gain access to encrypted information. The system successfully identified the unauthorized attempt because it continuously failed to decrypt the data with the unauthorized key, producing either an empty output or an

error. These results demonstrate the robust security and efficiency of the solution in preventing unauthorized access, as well as the reliability of system encryption and key management schemes in protecting sensitive data.

5.6 Chapter Summary

This chapter introduced a decentralized and privacy-preserving framework for storing and retrieving data using blockchain, IPFS, and cryptographic techniques. It began by formalizing three core properties: immutability, traceability, and data availability, which are necessary for ensuring the trustworthy management of continuously generated IoT data. The proposed architecture integrates blockchain to anchor data fingerprints, IPFS to store encrypted payloads, and hashing mechanisms to maintain integrity throughout the data lifecycle. The implementation demonstrated that the system can efficiently encrypt, hash, and distribute data. The performance evaluation showed an increase in throughput with larger data sizes, confirming the solution's scalability. Moreover, security experiments validated the approach's robustness against integrity violations and unauthorized access attempts. Overall, this chapter establishes a secure, scalable, and verifiable method for decentralized data storage, providing a foundation for subsequent chapters that address scalability, regulatory compliance, and privacy-preserving verification mechanisms.

Chapter 6

Privacy Preserving Digital Certificate Verification and Validation

Certification of human, economic, and industrial activities is a necessary and sometimes legal obligation in various sectors. The primary objective of certification is to ensure the verifiability of the process and data integrity, while enhancing trust among individuals or organizations [113]. Physical certificates have existed in paper form for a while. However, the inherent operational complexities of physical certificate management, such as certificate creation, verification, and revocation, often result in inefficient, complex processes [114]. Therefore, as digital technologies continue to grow, the adoption of digital certificates has become crucial for industries such as e-commerce [115], health care [116], public resource management [117], education [118], and government services [119]. While digitization improves efficiency and accessibility, it also introduces new challenges, most notably the increased risk of certificate forgery or unauthorized modifications after issuance, posing a significant threat to trust. Additionally, traditional certificate systems rely on centralized authorities to issue, verify, and revoke certificates, ensuring their validity and acceptance. However, this centralized approach often suffers from transparency limitations [120]. Moreover, centralized solutions are prone to a single point of failure. The reliability and accessibility of certificates become entirely dependent on the availability and integrity of the central authority or designated custodian. Furthermore, the transition from paper to digital certificates introduces new challenges in protecting sensitive information. Privacy becomes a significant concern when managing and verifying certificates, as these processes may expose sensitive information and increase the risk of data exploitation [121].

A reliable digital certification system must address three core challenges: (a) ensure the immutability, reliability, and transparency of the provided information, (b) ensure the availability of certificates and the possibility of checking their validity, and (c) protect sensitive data by preventing unauthorized access during the issuance and verification processes, particularly in privacy-critical scenarios. Blockchain can inherently contribute to addressing the first two challenges (a) immutability and (b) availability. Namely, it replaces a traditional centralized system with a transparent, tamper-proof, and decentralized system that mitigates transparency issues and enhances trust [122]. Moreover, blockchain data is immutable, eliminating the risk of

forgery or unauthorized modification of certificates [123]. Additionally, the distributed nature of blockchain enhances system resilience by eliminating a single point of failure, ensuring that everyone can access and verify the stored information. Despite these advantages, blockchain technology has its limitations concerning the third challenge (c). Namely, there are concerns about privacy preservation in blockchain [124], since a blockchain ensures transparency by making all transactions public for all network participants. However, this same transparency exposes sensitive information in the certificates. For instance, metadata embedded in a digital certificate may reveal personal details that could compromise individual privacy or even pose risks to public security [125]. Therefore, a critical challenge in using blockchain technology for digital certificates is balancing privacy and transparency, particularly in healthcare or government services [126], where unauthorized access to sensitive information can have severe consequences. The problem of selective disclosure has also been studied in the Self-Sovereign Identity (SSI) ecosystem, particularly in approaches based on W3C Verifiable Credentials and Decentralized Identifiers (DIDs) [127]. In these systems, selective disclosure is often supported by privacy-preserving credential presentation mechanisms, such as BBS+ signatures [128]. However, such approaches are primarily designed for holder-controlled identity claims, whereas the scenario addressed in this chapter concerns publicly auditable compliance certification issued by a recognized authority. In many SSI-style architectures, blockchain is mainly used as a registry or status layer, while credential presentation and verification remain largely off-chain, typically within wallet-based workflows [129]. This differs from the proposed solution, in which any verifier must be able to validate a compliance certificate independently, without relying on issuer availability or wallet infrastructure. Moreover, the present use case requires proving that hidden measurements satisfy regulatory thresholds, which goes beyond selective disclosure of signed attributes and requires predicate proofs. Standard BBS+ selective-disclosure mechanisms alone do not provide this functionality without additional zero-knowledge proof machinery [128]. For this reason, the proposed solution uses blockchain not only as an anchoring layer, but also as the verification layer for Merkle-based attribute validation and ZKP-based compliance checking.

This chapter formalizes, designs, and experiments with an integrated solution that combines blockchain technology and ZKP to obtain a robust and efficient system for digital certificate management and verification, including a smart contract and Merkle tree-based structure for validating certificate attributes and enabling selective disclosure, and the application of the proposed privacy-aware, robust digital certificate system to the real-world scenario of public water management.

6.1 Motivating Example

Let us revisit the motivating example introduced in the previous chapters, which concerns the development of advanced water management systems capable of monitoring and controlling the quantity and quality of water. Besides collecting various environmental and operational parameters through sensors and IoT devices distributed across space and aggregated into districts, a water management system typically also requires issuing certificates that verify compliance with legally required quality and

quantity thresholds. In particular, while a pump extracts water from the ground, several onsite sensors collect essential data on water quality and quantity. Each station then transmits this data to its corresponding district authority, which aggregates the data from all its stations. For example, the district can quantify annual water extraction or loss at its stations, or analyze trends in contamination. The authorities can examine the district data to ensure compliance with policies such as water conservation or fair resource distribution.

Once the collected data has been properly aggregated and verified, the system issues a digital certificate to confirm compliance with the prescribed quality and quantity thresholds. An authorized entity typically signs the certificate, which must be made immutable and non-repudiable to ensure trust and integrity. Blockchain technology can help meet this requirement by enabling the publication and storage of certificates via smart contracts, thereby enhancing their immutability, traceability, and verifiability. However, not all information embedded in a certificate can be made public. For instance, the disclosure of raw operational data, such as extraction rates, losses, or sensor configurations, could expose sensitive operational techniques and lead to the exploitation of that data and to public misinterpretation. However, to ensure public trust and verify compliance with regularity requirements, such information should be publicly *verifiable*. To address these challenges, a privacy-preserving system should be able to demonstrate compliance without revealing the underlying operation or data, thereby enabling tamper-proof, transparent, and trustworthy verification.

Sect. 6.2 introduces a proposed solution, which combines blockchain technology, ZKP, and Merkle trees to let immutability and transparency coexist with privacy and selective data disclosure.

6.2 Proposed Solution

Starting from the motivating example in Sect. 6.1, this section proposes a layered solution for a digital certificate verification and validation system that integrates ZKP for compliance verification of sensitive data, a Merkle tree for checking integrity and validation of the certificates, and blockchain to guarantee privacy while, at the same time, enabling public verification of the entire process. The general architecture of the proposed solution is illustrated in Fig. 6.1. A set of sensors $\{s_1, \dots s_n\}$ located inside each station S_i continuously collect different environmental and operational data (i.e., $m_1 \dots m_n$ in the figure). Each station S_i collects the data and periodically transmits an aggregation D_i of it to its corresponding district. In particular, with reference to the figure, each value $d_{i,j}$ where $j \in [1 \dots p]$ in D_i is a tuple $\langle d^1_{i,j}, \dots d^n_{i,j} \rangle$ where each element $d^k_{i,j}$ is the aggregation of the measurements provided by sensor s_k during the reference period j. Inside each district, a dedicated *application* evaluates its compliance against predefined regulatory thresholds. Compliance validation is performed using a ZK proof, which allows one to compare each aggregated measurement in $d_{i,j} = \langle d^1_{i,j}, \dots, d^n_{i,j} \}$ against the compliance threshold $\{\tau_1, \dots, \tau_n\}$, without revealing the actual values of $\{d^1_{i,1}, \dots, d^n_{i,j}\}$. In this way, using a ZK proof, the verifier can be convinced of the quality and quantity of water with-

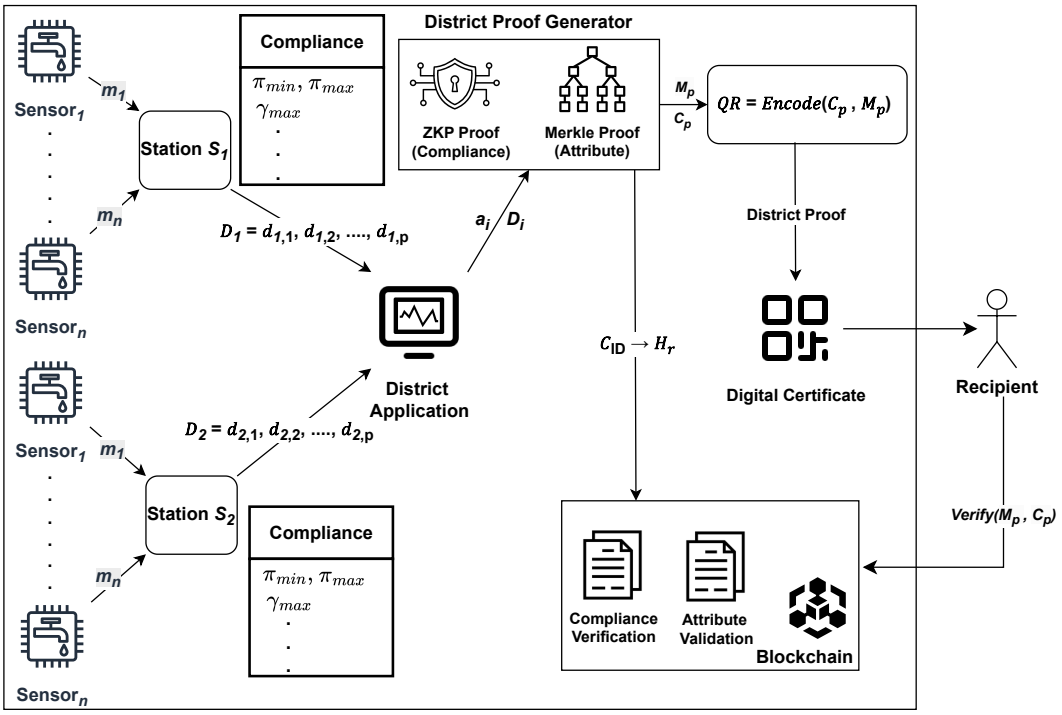


Figure 6.1: The architecture of the proposed solution.

out needing to know the exact measured values. Indeed, although the compliance thresholds are typically public and regulated by law, revealing precise measurement data from individual stations can raise privacy concerns, particularly for nearby residents. Moreover, exposing detailed sensor configurations or operational outputs could create security vulnerabilities.

If the data satisfies the compliance rules, the *District Proof Generator* will generate a valid proof using a ZKP to ensure compliance. In addition to this proof, a digital certificate issued by the water management system is typically emitted, which includes a set of publicly accessible attributes, such as issuer, recipient, expiry date, revocation status, and creation date. To enable selective, efficient verification of this information, the proposed solution integrates a ZK proof with a Merkle tree that contains the public certificate attributes and the certificate identifier. The Merkle tree root is then stored and verified on the blockchain using a smart contract to ensure tamper-proof and verifiability. The proposed architecture also operates under two distinct epistemic assumptions. The blockchain side follows a closed-world assumption: the on-chain state is treated as the complete and authoritative record of certificate validity, and anything not committed to the chain is considered unverifiable by the smart contract. The off-chain components, by contrast, operate under an open-world assumption: data readings and aggregated district data exist in a distributed environment where the absence of a record does not imply that the corresponding data does not exist, only that it has not been publicly committed. The ZKP circuit mediates between these two regimes. It takes the open-world witness data as private input, evaluates it against the closed-world compliance rules encoded in the circuit, and produces a proof that can be verified on-chain. This design allows the system to reason rigorously over real-world data without requiring that all of it be

publicly disclosed or permanently recorded.

The Merkle tree efficiently aggregates the certificate attributes into a single cryptographic structure, enabling selective disclosure of these attributes. In particular, each original data element serves as a leaf node, and recursive hashing is performed to derive a unique root hash. Starting from this data structure, it is possible to efficiently manage and validate individual certificate attributes. A blockchain-based smart contract maintains a tamper-proof, decentralized ledger for storing and selectively verifying certificates represented as Merkle trees. The proposed solution enables verifying certificate attributes and ensuring compliance with water management data without exposing sensitive information, thereby preserving privacy.

The following subsections provide a more detailed description of each component of the proposed solution.

6.2.1 ZKP-based Compliance Proof

As discussed earlier, the proposed system employs ZKPs to validate water management data against established compliance rules. From a formal perspective, a ZKP proves that a witness satisfies a constraint system without revealing the witness itself. In the case of zk-SNARKs, this system is typically encoded as a Rank-1 Constraint System (R1CS), in which each constraint is expressed as an arithmetic relation over a finite field. In this sense, the proof can be viewed as a certificate of satisfiability for a set of predicates over hidden data. In the proposed solution, the compliance rules are predicates over the district measurements and public thresholds, and the circuit proves that all of them hold without revealing the underlying sensor values. Although the circuit is not a general-purpose first-order reasoner, many policy checks relevant to certification can be reduced to arithmetic constraints and verified in zero-knowledge. This also suggests a natural integration path for policy engines or logical reasoners: a higher-level policy specification could first be translated into circuit constraints, after which the ZKP would certify that the hidden witness satisfies the compiled policy.

In the specific context of the motivating example considered in this chapter, each water monitoring station typically monitors water quality metrics, such as pH, Total Dissolved Solids (TDS), chlorine levels, turbidity, and quantities, such as loss and annual volume. Let us consider a district Δ composed of a set S of m stations:

$$S = \{S_1, S_2, ..., S_m\} \tag{6.1}$$

where each station S_i collects a set of water quality and quantity data measurements from n different sensors, in particular, it is assumed that each year the station S_i provides to the district Δ a set of aggregated measurement data tuples, one for each month in the year:

$$D_i = \{d_{i,1}, d_{i,2}, ..., d_{i,12}\} \tag{6.2}$$

where each tuple $d_{i,j} = \langle d^k_{i,j} \rangle^n_{k=1}$ is the aggregate value of the parameter k measured by S_i during month $j \in [1 \dots 12]$. The aggregation can be performed using simple operations, such as summation or averaging, or more complex ones, depending on the nature of k.

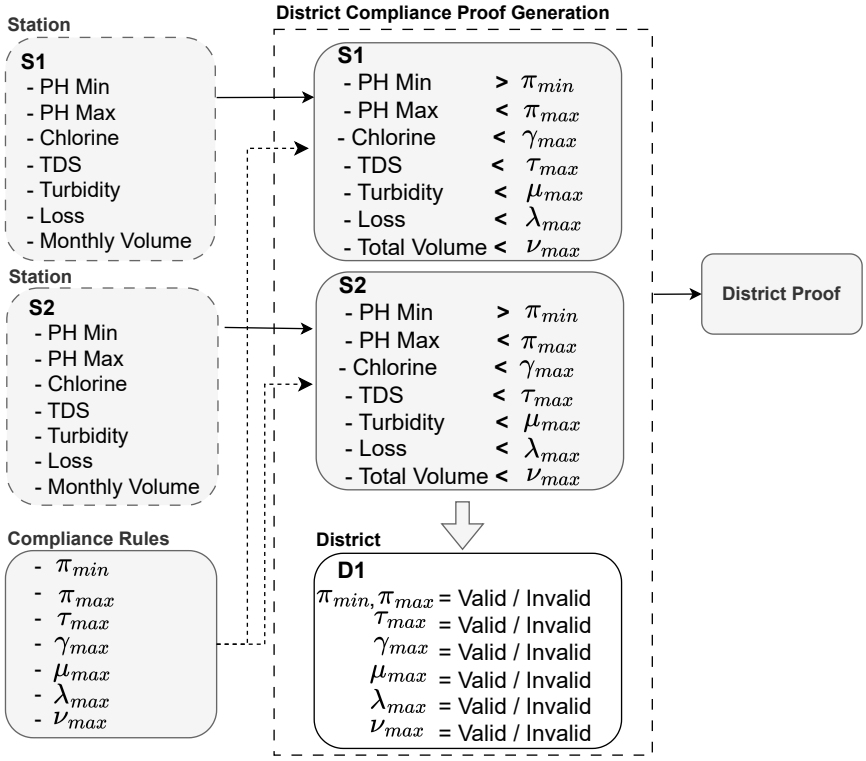


Figure 6.2: ZKP for Compliance Rules Verification.

Fig. 6.2 shows the compliance verification process for a set of parameters taken from the considered use-case example, where each tuple $d_{i,j} \in D_i$ contains measurements about seven features collected from the same number of sensors by S_i during month j:

- $d_{i,j}^{pH_{min}}$ and $d_{i,j}^{pH_{max}}$: the minimum and the maximum of registered pH values,
- $d_{i,j}^{tds}$: the total dissolved solids,
- $d_{i,j}^{chlorine}$: the monthly average value of chlorine,
- $d_{i,j}^{turbidity}$: the monthly average value of turbidity,
- $d_{i,j}^{loss}$: the total loss quantity by month,
- $d_{i,j}^{volume}$: the overall amount of water provided by month.

The proposed technique verifies water quality and quantity data from multiple monitoring stations against well-established compliance rules using a ZK circuit.

The compliance rules considered in our use case scenario typically involve verifying whether measured values fall within legally defined minimum and maximum thresholds for various quality and quantity parameters. Tab. 6.1 summarizes the compliance rules applied for the provided example. For instance, each station must ensure that the measured minimum and maximum pH values fall within the legally prescribed minimum and maximum pH range (i.e., π_{min} and π_{max}). In particular, the ZK circuit for compliance verification consists of the following:

Table 6.1: Example of compliance rules for a water management system.

Data $d_{i,j}$	**Thresholds**	**Compliance rule**
$d_{i,j}^{pH_{min}}$	π_{min}, π_{max}	$\forall j \in [1,12]\,.\,\pi_{min} \leq d_{i,j}^{pH_{min}} \leq \pi_{max}$
$d_{i,j}^{pH_{max}}$	π_{min}, π_{max}	$\forall j \in [1,12]\,.\,\pi_{min} \leq d_{i,j}^{pH_{max}} \leq \pi_{max}$
$d_{i,j}^{tds}$	τ_{max}	$\forall j \in [1,12]\,.\,d_{i,j}^{tds} < \tau_{max}$
$d_{i,j}^{chlorine}$	γ_{max}	$\forall j \in [1,12]\,.\,d_{i,j}^{chlorine} < \gamma_{max}$
$d_{i,j}^{turbidity}$	μ_{max}	$\forall j \in [1,12]\,.\,d_{i,j}^{turbidity} < \mu_{max}$
$d_{i,j}^{loss}$	λ_{max}	$\forall j \in [1,12]\,.\,d_{i,j}^{loss} \leq \lambda_{max}$
$d_{i,j}^{volume}$	ν_{max}	$\sum_{j=1}^{12} d_{i,j}^{volume} \leq \nu_{max}$

- A set of public inputs $\bar{x}$ represented by the values in column **Thresholds** in Tab. 6.1.
- A set of private inputs $\bar{y}$ represented by the data collected by each station $D_i = \{d_{i,j}\}_{j=1}^{12}$.

Based on the defined inputs, the ZK circuit Cir produces a compliance proof $C_p(\Delta)$ for the district Δ, which aggregates the data from all stations in the set S. This proof is generated by evaluating the circuit over the private inputs $\bar{x}$ (station measurements) and public inputs $\bar{y}$ (compliance thresholds), as expressed by:

$$C_p(\Delta) = Cir(\bar{x}, \bar{y}) \tag{6.3}$$

The construction of the proof also involves the calculation of all the intermediate signals of the circuit that satisfy all the constraints of the circuit, namely the computation of the witness, which can be exploited for the subsequent verification. In particular, using this and the computed witness, any verifier (whether authorities or final users) can verify the certificate's compliance without needing to access sensitive information. Given a compliance proof $C_p(\Delta)$, the verification process can be formalized through the following function *verify*():

$$verify(C_\Delta) = \begin{cases} valid, & \text{if } C_p(\Delta) = 1 \\ invalid, & \text{otherwise} \end{cases} \tag{6.4}$$

Successful verification results in a comprehensive district proof that ensures that all stations comply with the rules.

6.2.2 Merkle Tree for Certificate Validation

As explained at the beginning of Sect. 6.2, a digital certificate comprises two components: a ZK-based proof of compliance and a set of public metadata accessible to relevant stakeholders. In the context of water management systems, this metadata typically includes the certificate ID, creation date, expiration date, issuer, recipient, and revocation status. In particular, the revocation status is critical for managing complex revocation scenarios that cannot be handled with only a timestamp. For

instance, a certificate may be revoked for several reasons, such as expiration, issuance of a new certificate with updated measurements, changes to the compliance rules, or sensor malfunction. Certificate management and verification involve not only ensuring compliance with measurement rules but also providing secure, efficient access to associated metadata. To address this aspect, a Merkle tree to store a digital certificate is proposed, called *validation tree*, as illustrated in Fig. 6.3, where: attribute C_{ID} is unique certificate identifier, which could be randomly generated or assigned by the domain application, *Creation Date* is the date when the certificate was first issued, *Expiry Date* is the end date of validity of the certificate, *Issuer* is the authority or organization that issued the certificate, *Recipient* is the entity to whom the certificate is issued, *Revoked* is a status flag showing whether the certificate has been revoked, *Valid/Invalid* is the outcome of the compliance verification, showing whether the district data passed the rules, and *District ID* is the identifier of the district.

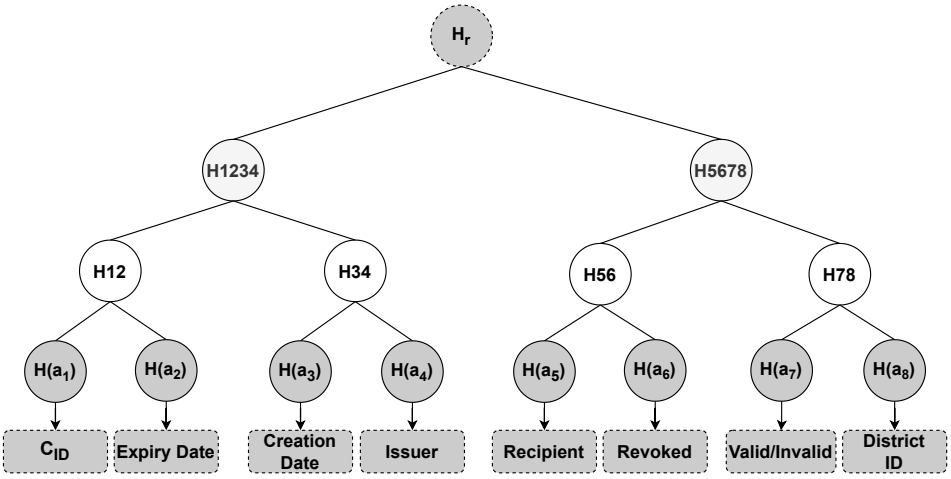


Figure 6.3: Merkle Tree for Certificate Validation.

Starting from the individual attributes a_i of the certificate, the leaf nodes of the Merkle tree are built by computing their corresponding hash value (i.e., $h_i = H(a_i)$). Given that, the tree is constructed from the leaves recursively. In particular, at each level ℓ, the nodes are computed by combining two neighbor children in the following way:

$$h_{nm}^{\ell} = H(h_n, h_m) \tag{6.5}$$

as illustrated in Fig. 6.3.

Through its recursive structure, the Merkle tree aggregates all certificate attributes into a single root hash while preserving the ability to verify each attribute independently. The root of the tree is computed as follows:

$$H_r = H(h_{left}, h_{right}) \tag{6.6}$$

The value H_r is stored within a smart contract through a map value whose corresponding key is the certificate identifier C_{ID}. To prove a particular attribute of the

certificate, such as the creation date or certificate status, it is sufficient to provide the issuer with the corresponding leaf node along with the path to the root.

A Merkle proof M_p (also known as a Merkle path) is the minimum set of nodes required to calculate the Merkle root again. For instance, consider the Merkle tree in Fig. 6.3 and a verifier who wants to confirm that a specific certificate with identifier C_{ID} a particular authority issued. In this case, the verifier would:

- Compute the hash $H(issuer)$.
- Retrieve the tree root H_r from the smart contract by using the key C_{ID}.
- Request the Merkle proof, that will consist of the nodes: $H(creationDate)$, $H(12)$, and $H(5678)$.
- Calculate H'_r from the retrieved Merkle proof and the computed leaf hash $H(issuer)$.
- Compare H_r with H'_r.
- If H_r and H'_r coincide, then the attribute is valid.

As can be observed, verification on a Merkle tree requires only a selective disclosure of information, which is crucial for privacy while still allowing transparency and verifiability.

In Fig. 6.3 among the metadata, the revocation attribute has been added to the leaf node to indicate the certificate revocation status. When the issuer wants to revoke the certificate, the status of the leaf node associated with the certificate status changes accordingly, and the recomputed root hash is posted on the blockchain against certificate C_{ID}.

6.2.3 Blockchain-based Certificate Verification

As described above and reported in Fig. 6.1, the *District Proof Generator* generates two kinds of information: the compliance proof C_p and the Merkle tree M_p for the certificate attributes, where the root H_r of M_p is stored in the blockchain to ensure immutability. In particular, the two informations C_p and M_p are embedded in a QR code as:

$$QR = Encode(C_p, M_p) \tag{6.7}$$

and made publicly available to anyone interested.

When a verifier scans the QR code, a set of smart contracts extracts the embedded data C_p and M_p to validate the certificate, check the compliance of the collected information, and retrieve the public metadata. Fig. 6.4 illustrates the verification and validation of certificates performed on the blockchain using smart contracts. The validation and verification require invoking two main smart contracts: the first one, called MerkleValidator, performs the attribute validation, while the second one, called Verifier and autogenerated by the `snarkjs` tools, provides a function *verifyProof*() for performing zk-SNARK proof verification. Alg. 11 illustrates the entire validation and verification workflow: it maintains a reference to an instance

of the MerkleValidator contract, which is responsible for performing the Merkle tree validation, and an instance of the Verifier contract, which takes care of the ZKP verification.

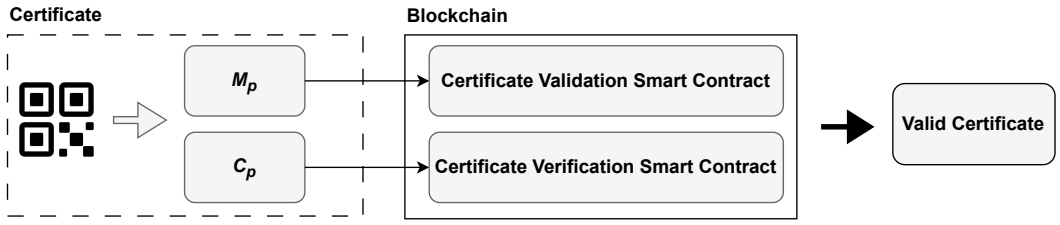


Figure 6.4: Certificate verification and validation on blockchain.

Algorithm 11: Certificate validation and verification process after a QR code scan.

```
1  Contract DistrictApplication :
2      merkleValidator ← MerkleValidator();
3      zkpVerifier ← Verifier();
4      … ;
5      function verifyCertificate(C_id, a_i, M_p, C_p, x̄) :
6          // Step 1: validate certificate attributes;
7          valid ← merkleValidator.verifyAttribute(M_p, c_id, a_i);
8          if !valid then
9              return false;
10         // Step 2: verify compliance proof;
11         valid ← verifier.verifyProof(C_p, x̄);
12         return valid;
```

The validation and verification are performed through the function *verifyCertificate*(), which takes as input the certificate id C_{ID}, the attribute a_i to check, the Merkle proof M_p, the compliance ZKP proof C_p, and the corresponding public information $\bar{x}$, representing the public thresholds provided by the authorities. First of all, the certificate attribute a_i is checked by invoking the function *verifyAttribute*() of the smart contract MerkleValidator in Alg. 12. This contract maintains the mapping *roots* that associates each certificate identifier C_{ID} with the corresponding Merkle root H_r. After retrieving the root related to the passed certificate identifier C_{ID}, the function uses the OpenZeppelin MerkleProof library[1] to check membership proofs and identify any changes in certificate data. Regarding access control, only the contract owner can update these roots using the *setRoot*() function, which is protected by OpenZeppelin Ownable2Step[2]. This contract is safer than the traditional Ownable modifier because the owner cannot accidentally transfer smart contract ownership to a mistyped address. Finally, any changes made to the *roots* mapping are recorded by the *RootSet* event for logging purposes.

[1]https://docs.openzeppelin.com/contracts/5.x/api/utils/cryptography#MerkleProof
[2]https://docs.openzeppelin.com/contracts/5.x/api/access#Ownable2Step

Algorithm 12: Certificate Merkle Validator Smart Contract

```
1  Contract MerkleValidator is Ownable2Step :
2      private mapping(string → bytes32) roots;
3      private _owner;
4      event RootSet(C_ID, H_r);
5      modifier onlyOwner :
6          require(msg.sender == _owner);
7      constructor (address initialOwner) :
8          _owner ← initialOwner;
9      function setRoot(C_ID, H_r) onlyOwner() :
10         roots[C_ID] ← H_r;
11         emit RootSet(C_ID, H_r);
12     function verifyAttribute(M_p, C_ID, a_i) → bool :
13         root ← roots[C_ID];
14         if root == 0x0 then
15             return false;
16         return MerkleProof.verify(M_p, root, a_i);
```

If the attribute validation succeeds, the second step involves verifying the ZKP compliance. In this case, the verification is performed by invoking the function *verifyProof*() of the Verifier smart contract in Alg. 13. This contract is automatically generated through the Circom tool to allow verifying proofs on the Ethereum blockchain[3]. The Verifier smart contract has only a view function called *verifyProof*() that returns TRUE if and only if the proof and the inputs are valid. In this way, the blockchain can validate C_p without revealing private measurement data, while ensuring that the compliance rules have been satisfied.

Algorithm 13: Verifier Smart Contract (auto-generated by snarkjs)

```
1  Contract Verifier :
2      function verifyProof(C_p, x̄) → bool :
3          // Verify the zk-SNARK proof C_p against the provided public inputs;
4          if proof is valid then
5              return true;
6          else
7              return false;
```

This blockchain-based design ensures that both the certificate attributes and compliance status can be independently and securely verified, without exposing any sensitive data. It provides strong guarantees of transparency, integrity, and privacy,

[3]https://docs.circom.io/getting-started/proving-circuits/#verifying-a-proof

which are the crucial requirements for digital certificate management in sensitive domains such as water resource monitoring.

6.3 Experimental Results

This section summarizes some experimental results obtained by implementing the proposed solution in the Solidity language, taking the Ethereum blockchain as a reference platform. The implementation will be evaluated in terms of resource utilization, cost, and security concerns. A single-page React application using the 18.3.1 version, along with Solidity version 0.8.0 smart contracts, was developed and deployed to the Ethereum Sepolia testnet. All experiments were conducted on a Windows 10 22H2 PC equipped with an Intel Core i7-3.60 GHz processor and 16 GB of RAM. The compliance circuit was written in Circom 2.1.5 and proved with snarkjs 0.7.5 using the Groth16 protocol. The trusted-setup phase relied on the public Ethereum Foundation's Perpetual Powers of Tau ceremony file. This ceremony involved numerous independent participants and generated a universal set of zk-SNARK parameters that can be safely reused for various circuits. Witness generation and proof creation were scripted in Node, while Hyperfine 1.17 was used for benchmarking. Smart contract deployment and debugging were carried out in the Remix IDE. The complete toolchain and version numbers are summarised in Tab. 6.2, and all the source code is available in the public GitHub repository[4].

Table 6.2: Experimental Setup

Specification	**Details**
Front-end framework	React 18.3.1
Programming language	JavaScript (Node 18 LTS)
Smart-contract language	Solidity 0.8.0
Blockchain network	Ethereum Sepolia (test-net)
Processor	Intel Core i7, 3.60GHz
RAM	16 GB
Operating system	Windows 10 22H2
Circuit compiler	Circom 2.1.5
Proof toolkit	snarkjs 0.7.5
Benchmark tool	Hyperfine 1.17
Hashing	Keccak-256

6.3.1 Resource Utilization for Proof

This section evaluates the proposed solution in terms of resource utilization, considering both the resources required to compute the zero-knowledge proof and the Merkle proof.

[4]https://github.com/MuhammadBinSaif/BC-and-ZKP-Certificate-Validation-and-Verification

Zero Knowledge Proof

Tab. 6.3 summarizes the structural complexity of the compliance circuit. The compliance circuit requires 17,803 wires, 19,987 signals (labels), and 17,795 arithmetic checks (rank-1 constraints), which implies that each wire is checked by a simple equation on average. The prover supplies the circuit with 840 private (monthly sensor readings) and eight public values (compliance thresholds). The circuit has only one output, a true or false flag that indicates whether all compliance conditions are valid or invalid. To evaluate the performance of the proposed solution, the processes of witness generation, proof construction, and verification are assessed. Each stage has been executed 30 times using the Hyperfine benchmarking tool (v1.17) to evaluate statistical performance in terms of runtime (mean, standard deviation, minimum, and maximum), as well as CPU and memory usage. Hyperfine measures the computation time (elapsed time) in terms of the actual wall-clock duration experienced by the user and CPU time, which represents the sum of total processor time across all threads and can exceed the computation time (elapsed time) on a multicore processor such as Intel i7 (3.60 GHz, 4 cores), where threads may run in parallel. In practice, these results demonstrate that the circuit is sufficiently small to run on standard hardware, while still verifying a full year of data for all monitored water parameters.

Table 6.3: Circuit metrics produced by `snarkjs r1cs info`.

Metric	**Value**
Elliptic-curve	BN-128
Number of wires	17,803
Rank-1 constraints	17,795
Private inputs	840
Public inputs	8
Outputs	1
Labels (signals)	19,987

Tab. 6.4 presents time and memory statistics for 30 runs. The workflow for ZKP is lightweight even on a standard system (Intel i7, 16 GB RAM). The witness computation completes in 0.099 seconds on average (SD 0.006 seconds), and proof construction in 1.486 seconds (SD 0.091 seconds). For verification, it takes 0.004 seconds, effectively making the entire process fast for end-users. The peak memory consumption is approximately 32 MB for witness creation, 5 MB for proving, and less than 1 MB for verification. The proof size is 1.19 KB, including public signals.

The results confirm that full compliance data for one year can be proven and verified in under two seconds with negligible memory usage.

Merkle Proof

Tab. 6.5 presents the time and memory statistics for Merkle-tree operations for 30 runs. Building the tree took an average of 0.184 seconds (with a standard deviation (SD) of 0.009 seconds), ranging from 0.171 to 0.202 seconds, with CPU time of

Table 6.4: Time & memory statistics for Groth16 (30 runs).

Stage	Computation (Elapsed) Time (sec)			CPU	RAM
	Mean	SD	Min-Max	(sec)	(KB)
Witness	0.099	0.006	0.089-0.113	0.162	32,580
Prove	1.486	0.091	1.359-1.696	4.520	5,168
Verify	0.004	0.0002	0.003-0.005	-	<1000
Proof size: 1.19 KB (including public signals)					

0.228 seconds and RAM usage of 32,700 KB. Generating a single proof required an average of 0.192 seconds (SD 0.015 seconds), with a time range of 0.170 to 0.239 seconds, utilizing 0.226 seconds of CPU and 9,800 KB of RAM. Computing all proofs averaged 0.193 seconds (SD 0.011 seconds), within a 0.179–0.227 seconds range, consuming 0.232 seconds CPU and 33,900 KB of RAM. Verification in JavaScript takes an average of 0.187 seconds (SD 0.009 seconds), spanning from 0.174 to 0.205 seconds, with a CPU time of 0.233 seconds and under 1,000 KB of RAM. Proof size is 1.68 KB for the full proof, confirming the feasibility of lightweight, browser-based verification.

Table 6.5: Time & memory statistics for Merkle-tree (30 runs).

Stage	Computation (Elapsed) Time (sec)			CPU	RAM
	Mean	SD	Min-Max	(sec)	(KB)
Build tree	0.184	0.009	0.171-0.202	0.228	32,700
Single proof	0.192	0.015	0.170-0.239	0.226	9,800
All proofs	0.193	0.011	0.179-0.227	0.232	33,900
Verify (JS)	0.187	0.009	0.174-0.205	0.233	<1000
Proof size: 1.68 KB (full proof)					

A mid-size QR-code version 15 stores 1,723 bytes in binary mode at error-correction level M and 2,203 bytes at EC-L (ISO/IEC 18004 capacity table). Therefore, the 1.68 KB Merkle-proof package fits comfortably in a single QR code version 15, even at the default EC-M level. The 1.19 KB Groth16 proof with public signal can also fit in a single QR-code version 15 with default EC-M level. The certificate proof remains well within the practical payload range of the QR code for both the ZKP and the Merkle proof.

6.3.2 On-chain Gas Cost

Tab. 6.6 presents the on-chain Gas and associated USD costs for storing Merkle roots and deploying the smart contracts required for Merkle and ZKP verification. Storing a Merkle root consumes 49,311 Gas units, corresponding to 0.16 USD, while the deployment of the Merkle smart contract requires 558,293 Gas units (1.76 USD), and the ZKP smart contract consumes 549,127 Gas units (1.71 USD). These cost estimates are based on Ethereum's market value as of 14 July 2025, when 1 ETH

was priced at 3,047 USD. The low computational and financial overhead of these operations demonstrates that the proposed solution is both efficient and scalable. It enables privacy-preserving verification and selective disclosure at minimal cost, thereby supporting practical and sustainable deployment in real-world applications.

Table 6.6: On-chain Gas and USD cost for saving a Merkle tree root and deploying the Merkle and ZKP verifiers smart contracts (SC).

Operation	**Gas Units**	**ETH Cost**	**USD Cost**
Save Root	49,311	0.00005161	0.16
Deploy Merkle SC	558,293	0.00057658	1.76
Deploy ZKP SC	549,127	0.00056070	1.71

ETH price = 3,047 USD at time of measurement.

6.3.3 Security

To evaluate the security robustness of the proposed system, a structured threat analysis was conducted based on the STRIDE framework [130]. Tab. 6.7 maps each relevant data flow in the system to its associated threat category, along with the corresponding mitigation strategies implemented within smart contracts, ZK circuits, or the decentralized application (dApp). It also includes an assessment of residual risk and a summary of the verification procedures used to validate each mitigation. The subsequent four subsections reference specific row identifiers from the table and provide experimental evidence demonstrating how threats in each STRIDE category are effectively addressed in practice.

Integrity

The proposed solution mitigates the risk of data tampering. Threat ID T2 indicates that a forged QR code with an incorrect Merkle root or path is immediately rejected, as the dApp recomputes the root and compares it to the root stored on-chain; therefore, any mismatch indicates an invalid proof. T4 confirms that any change in sensor data can be detected and invalidates the proof. T1 prevents the attacker from spoofing a false root on the contract because every transaction is signed by the legitimate owner, and the contract checks against the *msg.sender*. Threat T3 blocks an adversary who steals or misassigns the owner key, because *setRoot*() is protected by the two-step *transferOwnership* → *acceptOwnership* workflow and the *onlyOwner* modifier. Collectively, these restrictions maintain the immutability and trustworthiness of the information, ensuring that the verifier only accepts legitimate data.

Privacy

The proposed solution ensures transparency with strict privacy guarantees. As detailed in T6, raw sensor readings and station identities remain confined within the ZK proof and are never stored on-chain or embedded in the QR payload. The blockchain only keeps the Merkle root and a limited number of public attributes. The solution

Table 6.7: STRIDE threat model for the proposed solution.

ID	STRIDE	Operation	Threat scenario	Mitigation in design	Risk	Test
T1	**S**poofing	ProofGen→Smart Contract	Fake root hash	Tx signed by owner and contract checks msg.sender	Very low	Unit test for replay-attack and slither auth check
T2	**S**poofing/ **T**ampering	QR code→ Verifier dApp	Forged QR embeds wrong root/path	dApp recomputes root and queries smart contract, any mismatch rejected	Low	Random bogus QRs and 0 accepted
T3	**S**poofing/ **E**levation	Owner→updateRoot()	Adversary uses stolen key to overwrite root	Two-step handover transferOwnership then acceptOwnership, also onlyOwner with events log	Very low	Requires two calls and non-owner tx reverts. Slither also confirms onlyOwner
T4	**T**ampering	Owner→ ProofGen	Dataset altered before proof generation	SHA-256 hash of data and any bit flip invalidates the proof.	Low	Test flips one byte ⇒ verify() fails
T5	**R**epudiation	ProofGen→ Smart-contract	Owner denies submitted root	Tx permanently lives on chain	Very low	Tx hash confirms msg.sender is owner
T6	**I**nformation Disclosure	On-chain storage	Private sensor data	Only Merkle root with public data stored on-chain and raw sensor data in ZKP	Low	snarkjs r1cs info/debug confirms no private signals appear in the public output
T7	**D**enial of Service	QR Code→Verifier dApp	Oversized proof exhaust browser resources	dApp checks length before decoding	Moderate	Stress test: tested with oversized proofs and all reverted without exhausting CPU
T8	**E**levation	*verifyProof*() *validateCertificate*()	Re-entrancy swaps root mid-check	*verify*() function is *view*-only and makes no external calls. *updateRoot*() is *onlyOwner* and protected by *ReentrancyGuard*	Very low	Slither: no re-entrancy paths

also complies with GDPR regulations, while allowing public verification, since no operational or personal data is disclosed: the verifier learns nothing beyond the truth of the compliance statement. Therefore, the system achieves selective disclosure and public verifiability without compromising privacy.

Availability

Another key objective is maintaining the availability of the verification. T7 shows that the dApp mitigates denial-of-service attacks by implementing proof-length checks that discard oversized or malformed proofs before resources are depleted. Additionally, the decentralized nature of blockchain replication eliminates single points of failure, thereby enhancing fault tolerance. Furthermore, all verification functions are declared as `view` and make no external calls, so they cannot be blocked by re-entrancy or gas-heavy operations. Therefore, users can access and check the stored

information, even under network stress.

Correctness

Smart contracts and ZK circuits both enforce correctness. T8 ensures that ReentrancyGuard protects the verifier contracts and accepts only well-formed, side-effect-free calls; therefore, any attempt to change state during a proof check will fail. Each compliance rule is encoded as a rank-1 circuit-level constraint, so a proof can be constructed only when all conditions are met. If a malicious prover provides inconsistent data, the on-chain verification immediately returns an invalid result. These layered security measures guarantee that all malformed or adversarial inputs are reliably excluded, thereby upholding the correctness and integrity of the verification process.

6.4 Chapter Summary

This chapter presents a complete solution that combines blockchain technology and ZKP to achieve a robust, efficient system for digital certificate management and verification. In particular, while a ZKP-based compliance verification mechanism enables validating certificate compliance while safeguarding sensitive information, the use of smart contracts and a Merkle tree allows validating certificate metadata through selective attribute disclosure. Along with the system's architecture, several experiments were conducted to provide evidence on the solution's feasibility in terms of costs and resource utilization. Finally, the security and correctness of the solution are evaluated using the well-known STRIDE threat model, thereby confirming its robustness. Unlike general SSI-oriented credential systems, the proposed framework is tailored to regulated certification scenarios in which selective disclosure, privacy-preserving compliance proof, and publicly auditable verification must coexist.

Chapter 7

Blockchain Privacy and Compliance to Data Regulations

The blockchain technology can ensure data availability to all stakeholders. Indeed, when data is published on the blockchain, its availability does not strictly depend on a single company or organization storing it in an internal or private cloud system. Indeed, a single organization typically owns traditional information systems, and the availability of the collected data depends on the existence of such an organization and its intention to make it accessible to all stakeholders. Conversely, data stored in a public blockchain is freely available to everyone without spatial or temporal limitations. Even if these three properties are essential in many real-world applications, when the stored and collected data relate to critical systems or personal data, these desirable properties conflict with certain constraints related to data protection regulations, such as the GDPR in Europe [131]. For instance, the GDPR introduces eight data subject rights, including the "Right to rectification" (GDPR Art.16) and the "Right to be forgotten" (GDPR Art.17), which induce the need for a careful investigation of questions like blockchain deletions, or more generally, blockchain updates [132], with a consequent enrichment of the traditional notion of immutability. At the same time, the "Right to be informed" (GDPR Art. 15) ensures individuals the right to be informed about the collection and usage of their personal data, and it can be obtained only with a proper integration of blockchain and smart contract technology [96].

This chapter examines how the effective combination of blockchain technology, smart contract data structures and functionalities, and encryption techniques can support the eight data subject rights outlined in the GDPR. Several alternatives are evaluated and explored, starting with the simplest ones, which can only partially fulfill some of the prescribed rights, and progressing to the most sophisticated ones, which support all rights. The last proposed solution establishes best practices for any application dealing with sensitive information.

7.1 Problem Statement and Formalization

This section formalizes the considered problem by starting from the three properties of blockchain: immutability, traceability, and availability, and the eight data sub-

ject rights prescribed by the GDPR. The three blockchain properties have already been introduced and discussed in Sect. 5.2. In particular, data immutability refers to the fact that once information is stored within a blockchain, it cannot be further deleted or modified; in this sense, blockchain can be considered an essentially write-once ledger. However, as already discussed, this definition limits the applicability of blockchain in real-world contexts, not only due to GDPR prescriptions, but also because data frequently needs to be rectified or updated due to incorrect or incomplete data entry operations. For this reason, an extended definition of immutability has been introduced, in which updates are possible in the sense that they can be registered, but with a clear connection to the prior version and without losing the original information. This characteristic is closely related to the one that follows, which concerns the traceability and integrity of the data. Finally, data availability is strictly dependent on the characteristics of public blockchains, which do not rely on a limited and predefined set of nodes. Indeed, it can typically exploit many nodes that, without a central authority, can provide data access to everyone interested.

Nowadays, any application that handles data must also comply with data protection regulations, such as the GDPR in Europe. In particular, this regulation identifies the following eight data subject rights that have to be supported in some way by data-oriented blockchain applications.

Right 1 (Right to be informed). The right to be informed allows individuals (data subjects) to access information about their personal data processing. In particular, it enables individuals to know what personal data is collected, why it is collected, who is collecting it, for how long, and if any data sharing is involved. This right is a consequence of Articles 12, 13, and 14.

This right connects to the next one, which gives users the ability to request details about how their data is processed.

Right 2 (Right of access). Individuals have the right to request and receive information from an organization about how their personal data is processed and used. This right is derived from Articles 12, 15, and 46 of the GDPR.

Once individuals learn how their data is collected and processed, they can require organizations to correct any inaccurate personal data, delete it entirely, restrict the type and amount of processing, or prohibit processing altogether.

Right 3 (Right to rectification). The right to rectification enables individuals to request that an organization update any inaccurate or incomplete personal data it maintains about them (GDPR Art. 16).

Right 4 (Right to be forgotten). The right to be forgotten is also known as the right to erasure. It allows individuals to request that an organization delete their personal data from storage (GDPR Art. 17).

Right 5 (Right to restrict processing). Individuals can request that an organization limit how it uses their personal data, although the organization is not automatically required to delete it (GDPR Art. 18).

Right 6 (Right to object to processing). This right allows individuals to object to the processing of their personal data at any time, in certain situations, depending on the purpose of processing and the lawful basis for processing (Article 21 of the GDPR).

This right extends the previous one by allowing individuals (data subjects) to request that the organization (data controller) cease processing their personal data. Their decision-making and profiling activities represent the way organizations process personal data. This activity is very important and common, so it deserves a specific right prescribed by the GDPR.

Right 7 (Rights in relation to automated decision-making and profiling). The data subject shall have the right not to be subject to a decision based solely on automated processing, including profiling, which produces legal effects concerning him or her, or similarly significantly affects him or her (GDPR Art. 22).

The following right is one of the novelties among data subject rights, since it deals with the ability of individuals to not only request the set of their personal data that an organization is storing and processing, but also to obtain it in a portable format.

Right 8 (Right to data portability). It allows individuals to obtain personal data they have previously provided to the organization in a structured, commonly used, and machine-readable format (GDPR Art. 20).

Given the blockchain properties and the GDPR data subject rights presented here, the following section discusses how they can be achieved and satisfied through the combined usage of blockchain technology and smart contracts.

7.2 Proposed GDPR Compliant Framework

This section discusses how the various data subject rights previously introduced can be implemented using blockchain technology. In particular, an incremental approach is adopted, starting with the most natural solution, which involves storing data directly on the blockchain using smart contracts. This approach provides immutability and traceability, but it may contradict certain rights. Then, it moved on to the most sophisticated solution, which combines blockchain, smart contracts, and encryption technology. The discussion will focus on each right individually, enhancing the robustness and complexity of the solution to accommodate all GDPR rights.

7.2.1 Direct Storage of User Data On-Chain

The *Right to be informed* (Right 1) is automatically ensured by the data availability property characterizing blockchain technology. In particular, any information directly stored in a blockchain transaction or as an input for a smart contract functionality is permanently and immutably registered in the ledger. Anyone interested in knowing which data has been collected can scan the blockchain content for details. However, to implement this right effectively, one can design custom smart contracts that allow users to directly access their collected data and log every action along with the addresses that perform them.

This solution is summarized by the data structures in Alg. 14 and the smart contract in Alg. 15. In particular, to properly store both the collected user data and information about the collection activity on the blockchain, two data structures are

Algorithm 14: Data structure describing the user data, as well as the collection or sharing activities performed.

```
1  struct Data :
2      string name;
3      string surname;
4      string birthday;
5      string email;
6      ...
7  struct Log :
8      address organization;
9      uint date;
10     uint duration;
11     address[] sharing;
```

proposed. The user data must be properly customized based on the specific application's needs; the *Data* in Alg. 14 represents only an example of sensitive information. At the same time, the information about data collection (i.e. *Log*) should be extended as well, but has to contain at least the address of who has performed the collection (i.e., *organization*), when it has been performed (i.e., *date*), and for how long (i.e., *duration*), as well as the information about eventual data sharing with other organizations (e.g., represented by the list *sharing*). Given these two data structures, the permanent storage of such information can be performed through the smart contract presented in Alg. 15, where data subjects are identified through their address, and for each of them it is possible to retrieve the collected data through the mapping *dataset* and the list of collection logs *access*.

Algorithm 15: Example of a smart contract storing on-chain data subject and access information.

```
1  Contract OnChainStorage :
2      mapping(address ⇒ Data) private dataset;
3      mapping(address ⇒ Log[]) private access;
4      function getUserData(address a) public view return Data :
5          access[a].put(access to user data);
6          return dataset[a];
7      function operation(address a, ...) public :
8          // perform operation;
9          access[a].put(access log);
10     function getDataAccess(address a) public view return Log[] :
11         return access[a];
```

This solution can likewise be extended to satisfy the Right of Access (Right 2). Indeed, by properly enriching the *Log* data structure to support a range of operations beyond data collection, users can obtain comprehensive information about how their

data has been processed. Although this solution is simple and easy to implement, it suffers from a limitation inherent to blockchain technology. In particular, since read operations on blockchain content or smart contract storage in a virtual machine environment are always possible without incurring any fees, any processing performed outside the blockchain by directly retrieving the data is not properly registered and cannot be communicated to the user. Moreover, this solution is insufficient to guarantee the rights of other data subjects prescribed by the GDPR. For this reason, more sophisticated solutions are necessary, as described in the following subsections.

7.2.2 Versioned Storage of User Data On-Chain or Off-Chain

Regarding the Right to Rectification (Right 3), data stored on the blockchain can be rectified in several ways, each offering a different balance between immutability and preservation of prior versions. One approach employs a versioned data structure within a smart contract, assigning an array of versions to each data item. When an update occurs, a new element is appended to that array. This method effectively turns the blockchain into a version-control system, retaining all previous versions and recording every update immutably. Alg. 16 illustrates a possible implementation of this solution: whenever an updated version of the same piece of information needs to be stored on-chain, it is appended to a list associated with the same key, represented by the subject address s. A limit on the maximum number of possible versions for each piece of information has been introduced to prevent Denial-of-Service (DoS) attacks and to maintain the version array's iterability [133, 134].

This solution is a good compromise between the immutability property of block -chain and the right to rectify previously stored information. It is strictly related to the immutability definition given in Sect. 7.1, but it does not accommodate the following right, which requires the possibility to delete the information stored on the blockchain completely. With the *Right to be forgotten* (Right 4), the data subject has the right to request the deletion of the data stored on the blockchain. This right is challenging to achieve, as information stored directly on the blockchain cannot be deleted or altered. However, in this case, the possibility of storing real data within IPFS and maintaining only the content identifier (CID), a hash of the data, and a timestamp indicating when the CID was published on the chain (see Alg.17) can be exploited. In other words, with reference to the solution proposed in Alg. 16, in this case, the data structure *Data* will not contain the actual data, but only the CID, the hash of the information stored in IPFS, and the timestamp. The immutable storage of the hash ensures that the data cannot be changed on the IPFS without being incontrovertibly recognized. Moreover, if users require the deletion of their data, it is sufficient to remove the content stored on IPFS, while the CID can remain on the blockchain without compromising Right 4.

One of the challenges associated with off-chain storage of data on IPFS, as opposed to directly on-chain, as in the previous solution, is ensuring that data on IPFS cannot be removed but is maintained by the nodes indefinitely. In this regard, IPFS provides a pinning service to keep the data alive and maintained by the nodes. However, when the owner decides to unpublish the data to be forgotten, they can directly request the data to be unpinned, and the IPFS garbage collector will remove the data.

Algorithm 16: Example of smart contract storing versioned on-chain subject data.

```
1  Contract VersionedOnChainStorage :
2      mapping( address ⇒ Data[] ) private dataset;
3      uint private MAX_VERSIONS;
4      ...;
5      function insertData(address s, Data d) public :
6          n ← dataset[s].length;
7          if n = 0 then
8              dataset[s].push(d);
9          else
10             raise "UserDataAlreadyPresent";
11     function updateData(address s, Data d) public return uint :
12         n ← dataset[s].length;
13         if 0 ≤ n < MAX_VERSIONS then
14             dataset[s].push(d);
15             return n;
16         else if n < 0 then
17             raise "UserDataNotPresent";
18         else
19             raise "MaxVersionsAlreadyReached";
20     function getData(address s, uint v) public view return Data :
21         n ← dataset[s].length;
22         if n > 0 ∧ v < n then
23             return dataset[s][n];
24         else if n < 0 then
25             raise "UserDataNotPresent";
26         else
27             raise "UnexistingVersion";
28     function getLastData(address s) public view return Data :
29         n ← dataset[s].length;
30         return getData(s, n);
```

Algorithm 17: Data structure for storing user data off-chain on IPFS and securing it on the blockchain.

```
1  struct Data :
2      string CID;
3      string hash;
4      uint32 timestamp;
```

It is essential to note that publishing an updated version of specific content generates a new CID, as each CID is derived directly from its content. Therefore, when users require the deletion of their data, all versions stored on IPFS must be removed. An alternative solution is to employ IPNS (InterPlanetary Name System), which returns a static CID that points to the latest version of the data. In this case, at any time, only the most recent version of the data is accessible within the blockchain, while previous versions are permanently deleted after being unpinned. This solution simplifies the management of updates and deletions, as it does not require storing all versions of the entire user data on-chain, thereby reducing operational costs. However, this approach comes at the expense of sacrificing immutability and traceability properties. To maintain a record of the data history, it is possible to store on-chain only a hash of each data version and its publication timestamp, as illustrated in Alg. 18. In this case, in the smart contract of Alg. 16, the mapping *dataset* will associate to each data subject's address only a *Data* value, and in this structure it can retrieve both the unique CID always pointing to the last version of the data, as well as the history of hashes and timestamps.

Algorithm 18: Data structure for storing user data off-chain on IPNS and securing it on the blockchain.

```
1 struct Data :
2     string CID;
3     string[] hashes;
4     uint32[] timestamps;
```

These solutions clearly ensure Right 3 and Right 4, but only assuming that once the data has been accessed, they are not stored somewhere by the nodes. Although off-chain storage solves rectification and deletion, fine-grained control over who can access data remains a challenge, since data stored on blockchain (i.e., the CIDs) and on IPFS (i.e., the actual data) can be easily accessed by everyone. To provide a higher security level and support Rights 5, 6, and 7, it is necessary to implement encryption techniques as well. To fill this gap, encryption using the Ciphertext-Policy Attribute-Based Encryption(CP-ABE) [135] and Proxy Re-Encryption (PRE) [136] is implemented as discussed in the following subsection.

7.2.3 Encrypted Storage of Data Off-Chain through Proxies

The rights related to the restriction, object or prevention of some processing activities, namely the *Right to restrict processing* (Right 5), the *Right to object to processing* (Right 6), and the *Right about automated decision-making and profiling* (Right 7), need the implementation of more sophisticated access control techniques which give the data subject ownership and responsibility of their data. To achieve these three rights, the proposed solution can leverage a combination of CP-ABE and PRE techniques, as illustrated in Fig. 7.1. The main idea is that each piece of data that needs to be collected and eventually processed is preliminarily encrypted under a CP-ABE policy and then stored in IPFS with its ABE header. The ABE is a cryp-

tographic scheme that allows users to encrypt and decrypt data based on attributes rather than just their identity.

In a CP-ABE scheme, users are associated with a set of attributes that describe their roles and permissions, while each data element is associated with a set of attributes that describe the access policies, which are stored in the ciphertext published on IPFS. In this case, this access policy specifies the conditions under which the encrypted data would be decrypted. At the same time, PRE is a cryptographic technique that allows a proxy entity to convert a ciphertext encrypted under one public key into an encryption under another key, without learning the original plaintext. In this schema, a data subject could use a PRE technique to delegate or revoke decryption without re-uploading the payload, while a smart contract will record all grants and revocations.

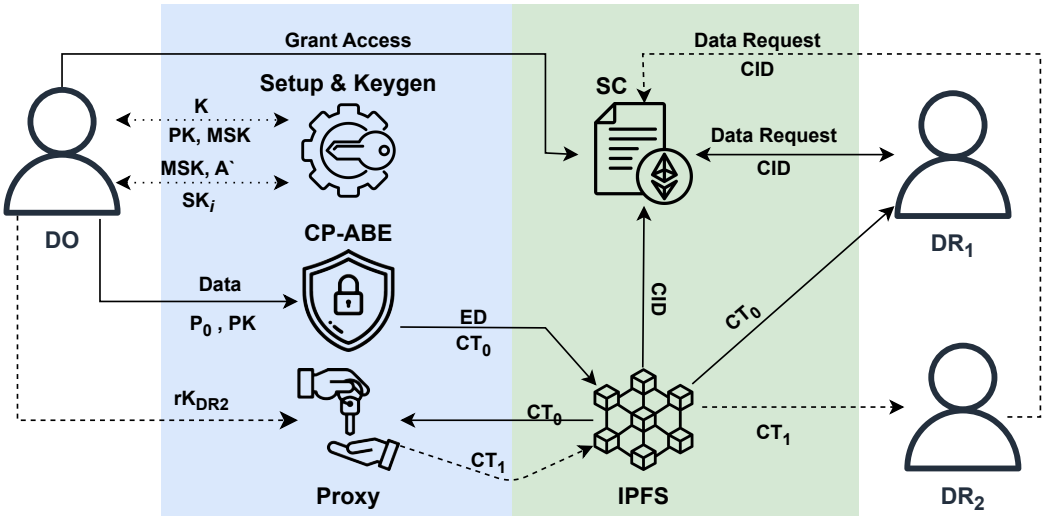


Figure 7.1: Proposed Solution

With reference to the situation in Fig. 7.1, it can recognize two kinds of users: the *data owner* (*DO*), which represents the data subject responsible for encrypting their data and for granting and revoking access permission to one or more *data requesters* (DR_i). The publication of data encrypted with the CP-ABE scheme and the interaction with the proxy for granting and revoking access to the data are performed in the following phases.

Setup and KeyGen – At the beginning, the *DO* runs a one-time setup algorithm with a security parameter (k), which produces a Public Key *PK* and a Master Secret Key *MSK*. *PK* encodes the global parameters required by everyone to encrypt data. When granting access rights to a new Data Requester DR_i, the *DO* selects a specific set of Attributes *A'* that define the decryption permissions for the user. Using the Master Secret Key *MSK* and these attributes, *DO* will generate a unique Session Key SK_i for that user.

Encryption – Before proceeding with the encryption of data and its storage on IPFS, the *DO* defines an access policy P_0 that specifies which attribute sets are required to perform the decryption (e.g., Doctor $\wedge$ EU-citizen). *DO* passes P_0 along with *PK* to a local CP-ABE module to encrypt the data. The execution of such a module produces, as a result, an encrypted data *ED* and a ciphertext CT_0 containing

the specified access policies. These two pieces of information are stored on IPFS in the next step.

Distributed Storage – The *DO* uploads the encrypted payload *ED* and the ABE header CT_0 to IPFS. IPFS returns two unique CIDs, one for *ED* and one for CT_0. These two pieces of information are stored in the smart contract illustrated in Alg. 19, which in this case is responsible for regulating the access to a specific, sensitive piece of information and is in the ownership of the data subject (see the modifier *onlyDO*).

In line with the solution provided in Alg. 18, IPNS for storing information off-chain is employed, allowing the same information CID to be maintained while updating the actual stored content. Therefore, the CID for *ED* is unique and is stored inside a data structure *edData* which contains the CID always resolving to the latest information version, an array of hashes, one for each stored version, and an array of corresponding publication timestamps (see Alg. 18 lines 2–5). Conversely, the CID for CT_0 is stored in a mapping *ctCID* in correspondence with the *DO* address.

Access Grant – To authorize a new requester DR_1, *DO* invokes the function *grantAccess*() passing as parameters the public address of DR_1 and the CID for CT_0. It is important to notice that the function is protected by the modifier *onlyDO*, so only the data owner can grant access to a new requester. This function essentially adds the address dr of DR_1 to the mapping *accessList* (Alg. 19, line 22) and updates the mapping *ctCID* by assigning to DR_1 the CID for CT_0. When the requester DR_1 wishes to access the plain data to decrypt *ED*, it calls the *getCID* function, which checks the presence of the corresponding address dr inside *accessList*, logs the access in the *logs* mapping, and returns the CID of *ED* and CT_0 (Alg. 19, lines 31-34). Given these two pieces of information, DR_i can run the CP-ABE decryption to recover the plaintext data. By employing this approach, *DO* can manage access control over the data, which satisfies the *Right to restrict processing* (Right 5) and the *Right to object to processing* (Right 6). The combination of the CP-ABE technique with the PRE method also enables revocation and association of grants with other data requesters, which differs from the original approach.

Proxy Re-encryption – Whenever the *DO* wants to grant decryption rights to a new requester DR_2 on the same data already encrypted under policy P_0, *DO* does not need to re-encrypt or re-store the entire payload. Indeed, *DO* can exploit PRE, so a semi-trusted proxy can transform the existing ciphertext CT_0 into a new ciphertext CT_1, which DR_2 can decrypt, without learning about the plaintext. The *DO* computes the re-encryption key rK_{DR_2} and sends it to the proxy service. The proxy runs the PRE algorithm to compute CT_1, which only rK_{DR_2} can use to decrypt the *ED*. The *DO* invokes the smart-contract function *grantAccess*(), adding DR_2 to the mapping *accessList* and updates the new address in the *updateCTCID* function (Alg. 19, line 28).

Revoke Access – The proxy allows *DO* to revoke the decryption right of an already authorized requester, such as DR_2, without re-encrypting the large payload *ED*. This operation is performed by *DO* through the *revokeAccess*() function, which will remove the DR_2 address from the *accessList* mapping. In this way, any request from DR_2 to *getCID*() will automatically fail (see *require* check in Alg. 19, line 25). In order to make the decryption information eventually already retrieved by DR_2 with a previous call to *getCID*(), the *DO* generates a re-encryption key using the *MSK* and

Algorithm 19: Access Control Smart Contract

```
1  Contract AccessControl :
2      struct EDData :
3          string CID;
4          string[] hashes;
5          uint32[] timestamps;
6      public EDData edData;
7      public mapping(address → string) ctCID;
8      public mapping(address → uint32[]) logs;
9      public mapping(address → bool) accessList;
10     private _owner;
11     modifier onlyDO :
12         require(msg.sender == _owner);
13     constructor (address owner, string cid_ed, string hash, string ts, string
         cid_ct_0) :
14         _owner ← owner;
15         edData.CID ← cid_ed;
16         edData.hashes[0] ← hash;
17         edData.timestamps[0] ← ts;
18         ctCID[owner].put(cid_ct_0);
19     function updateED(string hash, uint32 ts) onlyDO() :
20         edData.hashes.push(hash);
21         edData.timestamps.push(ts);
22     function grantAccess(address dr, string ctCID) onlyDO() :
23         accessList[dr] ← true;
24         ctCID[dr] ← ctCID;
25     function revokeAccess(address dr) onlyDO() :
26         accessList[dr] ← false;
27         delete ctCID[dr];
28     function updateCTCID(address dr, string newCTCID, uint32 ts)
         onlyDO() :
29         ctCID[dr] ← newCTCID;
30         logs[dr].push(ts);
31     function getCID(string edCID, string ctCID) :
32         require(accessList[msg.sender], "Access denied");
33         logs[msg.sender].push(ts);
34         return (edData.CID, ctCID[msg.sender]);
```

a revised access policy that omits DR_2 attributes, and sends that re-encryption key to the proxy. The proxy uses the updated re-encryption key to convert the old ciphertext header into a new header that only a valid SK_i can open. It uploads this new header to IPFS and updates the smart contract with the new CID. The combination of the

functions *grantAccess*() and *revokeAccess*() allows the system to completely support the *Right to restrict processing* (Right 5), since data subjects can define a granular access to their data decryption functionalities, and can decide to revoke them in a following step. The address whitelist in *accessList* (Alg. 19, lines 22 and 25) gives the *DO* the ability to switch any requester on or off with a single transaction. At the same time, the timestamped *logs* mapping records when accesses are granted or revoked. Therefore, in this way, data subjects can manage access to their data more effectively, even if it is publicly available. After a certain time, they can decide to revoke access or remove them from the IPFS if they want to be forgotten (Right 4).

Similarly, to support the *Right to object to processing* (Right 6), the *DO* can issue purpose-specific CP-ABE *SK* (via KeyGen), use PRE to generate per-purpose ciphertext headers, and then call functions *grantAccess*() (Alg. 19, line 22) or *updateCTCID*() (Alg. 19, line 28) to grant or revoke decryption rights for only those purposes. In this way, the technique also gives support to the *Right to object to processing* (Right 6) and *Right in relation to automated decision-making and profiling* (Right 7), considering the latter a special case of the former. The final right that remains to consider is the *Right to data portability* (Right 8). As already discussed in Sect. 7.1, this is the most specific right under the GDPR, as it concerns the format of data rather than their access and processing activities. However, since the proposed solution reverses the roles and makes the data subject the actual owner of the shared and collected data, this right could be considered automatically respected. Moreover, storing data on IPFS enables the application of any format and the storage of any information (i.e., not only text, but also images and videos).

7.3 Chapter Summary

This chapter explores the potential of blockchain to facilitate the data subject rights outlined in regulations such as the GDPR. In particular, it investigates how the blockchain's properties of immutability, traceability, and availability can either facilitate or hinder the support of the eight data subject rights outlined in the GDPR. Starting with the analysis of a very simple solution based solely on on-chain storage of the collected data, progressing to more sophisticated solutions that involve versioned storage of data, either on-chain or off-chain, and finally to the use of encryption techniques to achieve complete control over access and processing of data. Ultimately, it identifies a solution combining CP-ABE and PRE that addresses all eight investigated data subject rights.

The discussion in this chapter and the envisioned solution can be considered best practices and implementation patterns for any blockchain-based application that handles data that must comply with data protection regulations.

Chapter 8

Layer 2 Blockchain Scalability and Data Availability

Blockchain technology has transformed digital systems since the introduction of Bitcoin in 2009 [32], offering a decentralized and tamper-proof transaction platform maintained through peer-to-peer consensus. Despite their extensive adoption, major public blockchains like Bitcoin and Ethereum still face significant scalability limitations. Their low throughput, approximately 7 and 15–30 transactions per second, respectively, results in high costs and delays, especially when compared with traditional systems like VISA, which process 1,500-2,000 transactions per second [137]. Scalability remains one of the foremost challenges in implementing blockchain-based solutions. Each smart contract interaction incurs gas costs, and repeated deployments of similar contracts multiply both storage and operational costs. Moreover, the throughput of the underlying blockchain restricts the number of transactions that can be executed within a given time frame. These limitations are particularly challenging in applications that demand a large number of interdependent contracts, where efficiency and cost-effectiveness are critical [138]. These constraints arise from the decentralized consensus process, where every node must verify each transaction, ensuring security but creating a fundamental performance bottleneck. To address this, both industry and academia have proposed scalability solutions [139, 140, 141], generally categorized as Layer 1 (L1) and Layer 2 (L2) approaches. L1 methods modify core blockchain parameters, such as consensus algorithms or block size, but these changes are disruptive and may introduce new risks, such as network forking. Consequently, L2 solutions have gained prominence. They operate on top of the base chain, inherit its security guarantees, and increase throughput by executing transactions off-chain while relying on L1 for settlement and data availability. ZK-rollups exemplify this idea by posting only compact state updates and cryptographic proofs to L1, significantly improving scalability without altering the underlying protocol [142].

Overall, the inherent performance limits of L1 blockchains motivate the adoption of L2 protocols that offload computation and reduce on-chain overhead, enabling blockchain systems to scale to meet real-world demands. At the same time, *data availability* remains an essential challenge for L2 rollup solutions and requires additional attention and the development of innovative solutions. However, the adoption

of L2 solutions has introduced additional challenges, most notably the data availability problem: transaction data processed off-chain must be made available on the main chain to prevent data-withholding attacks and ensure that all participants can independently verify the blockchain state. In this context, data availability refers to the ability of all L2 participants to access data processed outside L1 to verify state transitions independently, an issue that poses a fundamental challenge for L2 solutions. Existing approaches address this problem either by publishing data on-chain or by relying on off-chain mechanisms; however, while on-chain data availability solutions incur high costs and long wait times, off-chain approaches can introduce significant data security risks. Moreover, in optimistic rollups, reliance on challenge mechanisms introduces an inevitable dispute period that may last up to a week, resulting in high transaction finality latency and limiting the practical usability of such solutions. As a consequence, data availability emerges as a core limitation that can undermine the scalability, security, and trust assumptions of L2 protocols if not adequately addressed. This chapter builds upon a comprehensive survey of rollup-based L2 solutions with a specific focus on the data availability problem and their respective advantages and limitations. Furthermore, it introduces a novel on-chain data availability solution that mitigates existing cost and latency issues by enabling the sharing of multiple rollup data within EIP-4844 blobs and by employing an Adaptive Multi-Factor Scoring (AMFS) prioritization mechanism, which constitutes the primary contribution of this chapter.

8.1 Rollups and Data Availability Problem

L2 solutions have emerged in recent years as a valuable alternative to increase the throughput and scalability of blockchain-based architectures. As already discussed in Sect. 2, the primary types of L2 solutions are state channels [51], sidechains [53], plasma [54], and rollups [56]. Among the various L2 solutions, rollups are considered particularly promising due to their ability to process transactions off-chain, thereby improving scalability while inheriting the security and decentralization of the underlying L1 by posting transaction data and proof of state changes [55]. Although rollups are a promising solution for scalability, they introduce new challenges, particularly data availability.

The term *data availability* refers to the ability of any participant to access all the information needed to verify transactions independently in a trustless environment. With reference to rollup-based L2 solutions, this means that the data used for state transitions in L1 must be accessible to everyone to ensure that any participant can independently validate the rollup state [143]. Data availability is crucial in rollups, as it directly impacts security, functionality, and interoperability. It is essential to ensure that participants interact with a valid blockchain state and trust the rollup protocol. For instance, if data is unavailable and a malicious rollup operator is involved, users may not be able to withdraw their funds from the main L1 chain because nodes cannot correctly compute the new blockchain state and the user's current funds [144]. The main reasons for which data availability is important can be summarized as follows:

- **Security** – Users can independently verify all the transactions and state transitions, maintaining the trustless nature of the blockchain.
- **Integrity** – Transactions and states are reliable, correct, and resilient to modifications.
- **Functionality** – It ensures that all required data is available for processing and validation, and ensures the proper functionality of the rollup.
- **Interoperability** – It ensures consistent data access to allow the interaction between multiple blockchain layers and applications.
- **Censorship Resistance** – It ensures that transaction data is processed and stored without involving a centralized authority or a malicious rollup operator. Users can submit transactions directly to the L1 zk-rollup contract to bypass rollup operator censorship.

Despite the importance of data availability, this property can lead to cost and performance issues due to the large volume of data that must be posted back to the L1 chain. Currently, there are two main approaches to data availability in rollups-based L2 solutions: on-chain and off-chain data posting. The first option typically offers greater security but also incurs higher costs for the end user. Conversely, the second one typically reduces costs and improves performance at the expense of smaller security or greater trust requirements. Tab. 8.1-8.2 presents a detailed comparison of data availability in the most prominent rollup-based L2 solutions. Specifically, Tab. 8.1 summarizes the main technical characteristics of these solutions, while Tab. 8.2 highlights their scalability and security properties, as well as the strengths and weaknesses with respect to data availability.

This thesis focuses on on-chain solutions, due to their security characteristics, which include *calldata* and *blob*. The *calldata* refers to a location where function arguments are stored. It is a read-only data area that does not alter the Ethereum state directly. This makes it a cost-effective way to store essential transaction data on-chain. On the other hand, *blob* is a new technique introduced with Ethereum's EIP-4844 (Proto-Danksharding), which enables the efficient posting of large amounts of data on-chain. *Blob* is not directly accessible by the EVM and is stored temporarily for verification. The following section details on-chain data availability solutions.

8.2 On-chain Data Availability

The on-chain data posting, or rollup mode, is a basic functioning mode for data availability in rollup. To ensure the security, transparency, and verifiability of blockchain transactions, all related data must be publicly available on-chain for all network participants to verify. This approach leverages the security and consensus mechanisms of the L1 blockchain, ensuring data integrity and information accessibility. Compared to optimistic rollups, ZK-rollups do not require storing much data on-chain, as validity proofs can validate the authenticity of state transitions without revealing the details. However, on-chain data availability still accounts for 80% of the

Table 8.1: L2 Rollup-based solutions to data availability: technical characteristics.

Rollup Name	Data Availability Solution	Provider	Verification Mechanism
ZKSync [145]	On-chain (Calldata)	Ethereum	zk-SNARKs
Starknet [146]	On-chain (Calldata or Blob)	Ethereum	zk-STARKs
Aztec (Private Blockchain) [147]	Off-chain (Initial Plan for Validium)	Off-chain	zk-SNARKs
Scroll [148]	On-chain (Calldata or blob)	Ethereum	zk-SNARKs
Linea [149]	On-chain (Blob)	Ethereum	zk-SNARKs
DeGate V1 [150]	On-chain (calldata)	Ethereum	zk-SNARKs
Taiko [151]	On-chain (calldata)	Ethereum	zk-SNARKs
Arbitum [152]	On-chain (blob) and off-chain (anytrust)	Ethereum/DAC	Fraud Proofs (Optimistic Rollup)
Optimism [153]	On-chain (calldata or blob)	Ethereum	Fraud Proofs (Optimistic Rollup)
Metis [154]	Off-chain	Memolabs	Hybrid (Fraud Proofs and Zk Proofs)
Polygon (zkEVM) [155]	On-chain (Calldata)	Ethereum	zk-SNARKs
Loopring (L2+Dex) [156]	On-chain (calldata)	Ethereum	zk-SNARKs
Immutable X (NFT Solution) [157]	Onchain (calldata) or Off-chain (Validium)	Ethereum/DAC	zk-STARKs
Manta Pacific [158]	Off-chain (Celestia)	Celestia	zk-SNARKs
ZKFair [159]	Off-chain (Celestia)	Celestia	zk-SNARKs

total transaction cost [160]. To overcome this challenge, various data compression techniques are used, ultimately reducing Gas fees for end users. These techniques include posting only state differences and employing data compression algorithms such as Huffman coding, Run-Length Encoding (RLE), and delta encoding. Moreover, state compression using Merkle trees and redundancy reduction via erasure coding can significantly reduce on-chain data while ensuring data availability and integrity. Posting only the state difference instead of complete transaction data helps to achieve scalability without overburdening the underlying L1. For instance, ZKSync [145] compresses the transaction data and computes the state differences, along with a zk-proof, to post on Ethereum L1. StarkEx [161], which utilizes the zk-STARK for its zk-rollup solution, is another prominent example of posting only the state difference on-chain for data availability.

Despite the usage of specific compression or optimization techniques, *calldata* and *blob* storage are the two main techniques to ensure on-chain data availability in rollups. Both solutions have their benefits and limits, as discussed below.

Calldata

In a smart contract, *calldata* is a read-only data area that works similarly to memory and is used to pass arguments to a function. Therefore, *calldata* remains on-chain as a part of the Ethereum chain history logs and is not stored as a part of the Ethereum

Table 8.2: L2 Rollup-based solutions to data availability: strengths and weaknesses.

Rollup Name	**Scalability**	**Security Features**	**Strengths**	**Weaknesses**
ZKSync [145]	High	Strong	Secure, Cost-effective	State Difference Only
Starknet [146]	High	Strong	Scalable, Secure	State Difference Only
Aztec (Private Blockchain) [147]	High	Strong	Privacy-preserving	Pending EIP-4844 Decision
Scroll [148]	High	Strong	Scalable, Secure	Centralized Operator
Linea [149]	High	Strong	Efficient, Scalable	Data Expiry After 18 Days
DeGate V1 [150]	Medium	Strong	Trustless, Decentralized	Unclear Merkle Tree Storage
Taiko [151]	High	Strong	Decentralized, Secure	Relies on Ethereum Validators
Arbitum [152]	High	Strong	Low cost, high throughput	Dispute Delays
Optimism [153]	High	Strong	EVM Equivalence	Dispute Delays
Metis [154]	High	Strong	Cheapest Transactions	Less Community Adaptation and Data Withholding Attack
Polygon (zkEVM) [155]	Medium to High	Strong	EVM Equivalence	Pending EIP-4844 Implementation
Loopring (L2+Dex) [156]	Medium to High	Strong	Flexible, Secure	Centralized Operator
Immutable X (NFT Solution) [157]	Medium to High	Strong	Decentralized, Flexible	Trust Issues in Validium Mode
Manta Pacific [158]	High	Moderate	Scalable, Cost-effective	Relies on Celestia
ZKFair [159]	High	Moderate	Scalable, Cost-effective	Relies on Celestia

state. Since *calldata* does not modify the Ethereum state, it provides a cost-effective way to store information. ZK-rollups employ *calldata* to publish compressed transaction data on-chain; the rollup operator creates a new batch by calling the relevant function in the contract and passing the encoded bytes array as payload to the function arguments. Let $T_1, T_2, ..., T_n$ be the individual transactions in the rollup batch. The rollup operator computes proof π and the state transition S of the batch B. The *calldata* for B is:

$$B_S = CE(T_1, T_2, ...T_n) \tag{8.1}$$

where *CE* is the function that compresses the transaction data and encodes the compressed data in a byte array. B_S along with proof π are then passed as parameters in the function call F_{verify} of the rollup smart contract R_S. An L1 smart contract verifies the proof, updates the state on-chain, and finalizes the batch.

$$F_{verify}(\pi, B_S) \rightarrow \text{Updated state in contract } R_S \tag{8.2}$$

Loopring [156], a decentralized exchange, employs Ethereum *calldata* for data

availability and zk-SNARK for verification. This ensures that all essential verification data is available without preserving the entire transaction history on-chain. In contrast, the Immutable X [157], a zk-rollup-based NFT solution, employs zk-STARK for proof generation and posts the state differences as *calldata* in L1 or off-chain. Conversely, Optimism [153] is an optimistic rollup employing the *calldata* or *blob* to ensure data availability. The batch includes both the transaction data and the result of the state transition, ensuring verification during the dispute period. Indeed, optimistic rollups assume that the transaction batch is valid unless someone challenges its validity.

Blob-carrying Transaction

Initially, the *calldata* was the only available option for rollups to ensure on-chain data availability. However, using *calldata* for transaction data storage led to higher Gas fees and limited scalability. For instance, storing a single non-zero byte costs 16 units of Gas, and a single zero byte is 4 units of Gas [99]. Furthermore, Ethereum currently has a limit of 30M Gas per block, allowing a maximum block size of 1.8 MB. To overcome these challenges, on March 13, 2024, Ethereum implemented EIP-4844 (Ethereum Improvement Proposal), also known as Proto-Danksharding [102]. This improvement is part of the Ethereum scaling roadmap, addressing both the cost and performance limitation of the previous *calldata* method. The EIP-4844 proposal introduced a new transaction type, called a *carrying transaction*, or *blob* [162]. The *blob* persists in the beacon node for a short time (almost two weeks), significantly enhancing the scalability of the Ethereum network and data availability in L2 [163]. Beacon nodes are special nodes in the network, primarily responsible for coordinating validators and maintaining consensus on Ethereum's Proof-of-Stake (PoS) beacon chain. *Blobs* do not allow direct access to Ethereum Virtual Machine (EVM) or smart contracts because their design enables efficient data storage and retrieval without requiring direct interaction with the EVM. Ethereum nodes store *blobs* separately from the main transaction data to efficiently process and validate transactions. Each *blob* holds 4096 field elements, and each element size is 32 bytes; therefore, a single *blob* can hold $4096 \times 32 = 128 KB$ of data. Furthermore, each block can hold up to 16 *blobs* or 2 MB of data, significantly improving storage and cost compared to *calldata*. A zk-rollup called Linea [149] is the first to implement *blob* for data availability fully, while other ZK-rollups, such as Starknet [146], Optimism [153], and Scroll [148], provide both *blob* and *calldata* options.

However, the introduction of EIP-4844 blobs significantly reduces the cost of publishing rollup data on L1; it does not fully eliminate inefficiencies related to data availability provisioning. In particular, each rollup must independently publish its data to a blob, which can lead to suboptimal blob utilization, increased competition for limited blob space, and unnecessary cost duplication when multiple rollups coexist. Moreover, the limited blob capacity per block and the growing number of active rollups introduce contention, potentially delaying the inclusion of rollup data and degrading data availability guarantees. As a consequence, while blobs improve scalability compared to calldata-based approaches, they still exhibit structural limitations that prevent efficient and fair usage across multiple rollup instances. To

address this issue, this chapter introduces the concept of shared blobs, in which data from multiple rollups is aggregated and published together in a single blob. By enabling controlled sharing of blob space and prioritizing rollup data based on multiple factors, the proposed approach aims to improve blob utilization, reduce costs, and mitigate contention without compromising data availability or security guarantees.

8.3 Proposed Solution

The proposed solution is based on the concept of shared blob, theoretically introduced in [30]. In particular, a comprehensive and operational architecture is proposed that can be integrated into real-world L2 blockchains to optimize the fixed storage space allocated to each blob, while minimizing posting costs and delays as much as possible. Fig. 8.1 shows the overall architecture of the proposed solution. It

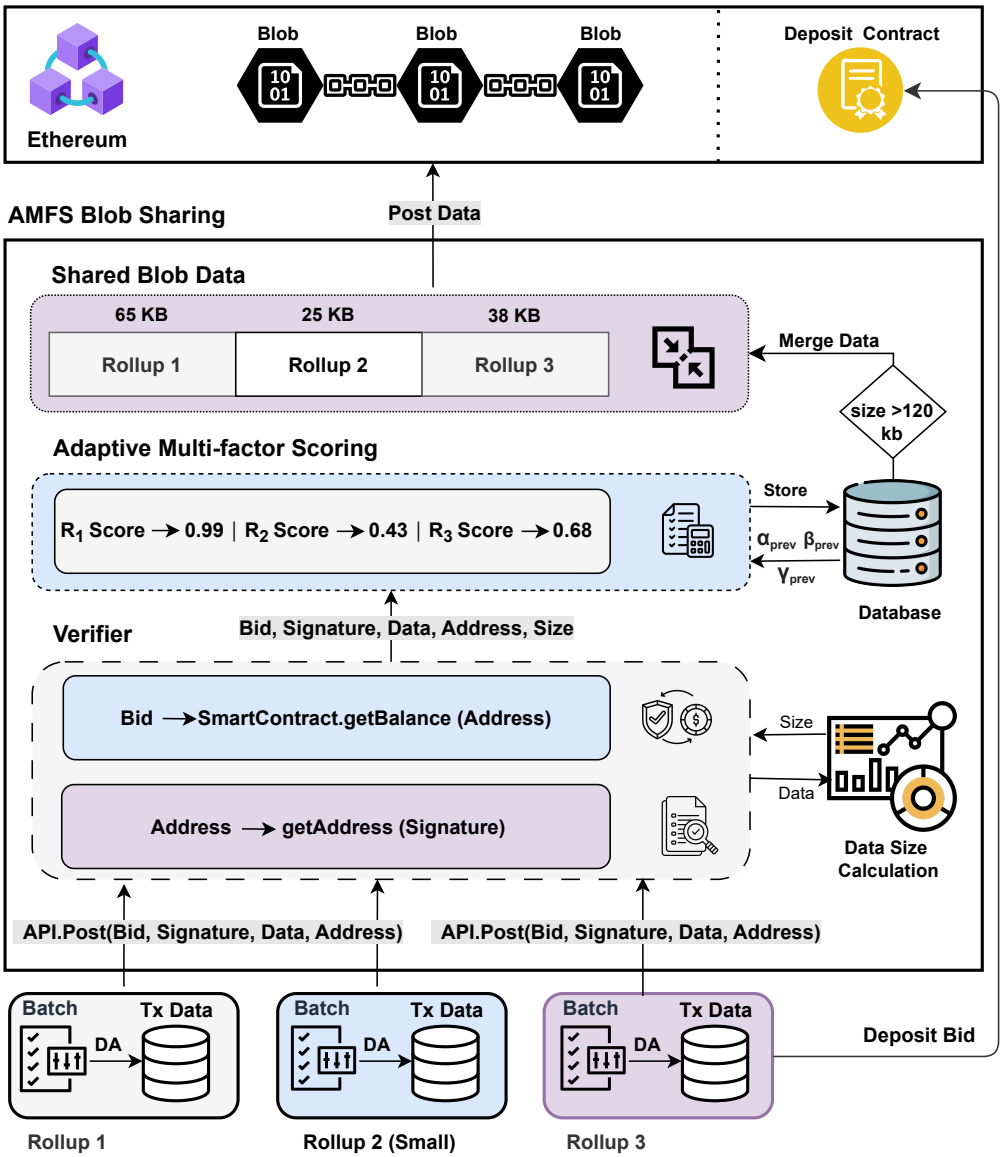


Figure 8.1: Adaptive Multi-Factor Scoring in shared blob.

implements an AMFS blob sharing system. This system consists of two main components: the Verifier, which verifies the validity of the submitted rollups, and the AMFS layer, which calculates the score, or priority, of each rollup and determines which ones to include in the next shared blob. The score of a rollup r is based on three parameters:

- B_r (bid): the amount of ETH that r is willing to pay to see its data included in a shared blob;
- W_r (waiting time): the number of rounds that r has already been waiting for inclusion in a shared blob;
- S_r (size): the size of the data submitted by r.

The joint consideration of these three parameters allows the definition of a fair and balanced score, particularly for small or lower bid rollups. More generally, one can evaluate the level of balancing provided by a scoring system with respect to the following three metrics:

1. Storage Utilization: a measure of the efficient use of the 128 KByte available in each blob. It depends on S_r.
2. Fairness: a measure of the waiting time for small or lower bid rollups, before their data gets included in the blob. It is strictly related to W_r and might depend on both S_r and B_r.
3. Efficiency: a measure of the overall time for filling and processing the blob. This parameter is cumulative and depends on the characteristics of the available rollups and the rate at which they arrive.

As illustrated in the lower part of Fig. 8.1, multiple rollups with variable data sizes and bids initially post their data using the AMFS API, together with their digital signature and public address. A digital signature is a cryptographic proof generated by using the private key of r to ensure that data has not been tampered with and to guarantee its origin. The address is a unique identifier for each rollup. Simultaneously, rollups also deposit the desired bid into a smart contract. The Verifier then checks this information through a two-step mechanism for each rollup r:

1. It verifies r's address by comparing the posted address with that retrieved from the signature. This verification is crucial to ensure the data's origin and integrity, and to mitigate the risk of data tampering.
2. It confirms the declared bid B_r associated with the address of r by accessing the smart contract.

After these two checks, the Verifier calculates S_r, including signature and address information. Then it passes B_r, W_r, and S_r to the AMFS layer to dynamically calculate the priority score of r. Initially, W_r will be zero. This score, together with r's data, is temporarily stored in a database to be retrieved later and used to build an optimized shared blob. The computed priority score helps optimize the utilization

of the storage provided by each blob and ensures that higher-priority transactions are posted first to L1. In particular, once the total size of the queued data reaches 128 Kbytes, data and metadata related to the pending rollups are merged and posted to a single blob during the final step.

The computation of r's priority score is based on a dynamic adjustment of B_r, W_r, S_r, and their relative importance, represented by three weights α, β, and γ, whose values are computed dynamically in a way detailed in Sect. 8.3.2. Fig. 8.1 shows this as an access to the database that retrieves the previous values of the three weights, that is, α_{prev}, β_{prev} and γ_{prev}, from which their new values are computed.

The following sections provide a more detailed illustration of the three components of the AMFS solution: the Verifier, the AMFS layer for scoring rollups, and the structure of the shared blob data.

8.3.1 Rollup Signature and Verification

In the proposed solution, each rollup data is signed before being posted through the API, and the Verifier checks this signature. This plays an important role in maintaining the integrity and security of rollup data. In particular, each rollup data is signed by using the rollup's private key before being posted to the AMFS system through the API. The Verifier employs a robust verification system where ECDSA (Elliptic Curve Digital Signature Algorithm) signatures [164] are validated by using the corresponding public keys associated with the data. This verification prevents unauthorized data tampering during transmission and guarantees data integrity. Moreover, it ensures data authenticity, proving that it originates from a rollup. Furthermore, this signature allows one to recover the sender's address, which is particularly important in scenarios where rollup users want to verify this sender independently and accept the data.

8.3.2 Adaptive Multi-Factor Scoring (AMFS)

The core of the AMFS system is the layer that computes the priority score for each rollup r. To ensure fairness and efficiency in shared blob storage, this computation is based on a weighted sum of the score parameters B_r, W_r, and S_r. In particular, these factors are weighted based on current and historical network conditions, resulting in dynamic adjustments to system behavior, as formalized below.

Definition 8.1 (Rollup priority)**.** Given a rollup r with bid B_r, waiting time W_r and size S_r, its priority score P_r is computed in the following way:

$$P_r = \alpha \times B_{norm,r} + \beta \times W_{norm,r} + \gamma \times S_{norm,r} \tag{8.3}$$

where $B_{norm,r}$ is the normalized bid for r, $W_{norm,r}$ is the normalized waiting time of r and $S_{norm,r}$ is the normalized size of r. These three normalized factors are computed from B_r, W_r, and S_r, respectively, using a min-max normalization.

This normalization is necessary because the three parameters typically have very different ranges, which would make some more relevant than others solely because of scale differences. Therefore, in order to avoid the domination of a single factor and

let the parameters contribute similarly to the overall score, in proportion to weights α, β, and γ, the min-max normalization scales each parameter to a 0-1 range, based on their minimum and maximum values, as below:

$$B_{norm,r} = \frac{B_r - B_{min}}{B_{max} - B_{min}} \tag{8.4}$$

where B_{min}, and B_{max} are the minimum and maximum bid, respectively. Similarly,

$$W_{norm,r} = \frac{W_r - W_{min}}{W_{max} - W_{min}} \tag{8.5}$$

where W_{min} and W_{max} are the minimum and maximum waiting rounds, respectively. Finally,

$$S_{norm,r} = \frac{S_r}{S_{max}} \tag{8.6}$$

where S_{max} is the maximum size allowed for a rollup, that is, the 128 Kbytes of a blob.

A dynamic adjustment of α, β, and γ is proposed, rather than static weights, to update them based on the current system condition and historical performance. This dynamic adjustment ensures that the scoring system remains effective and fair under changing system conditions and demand patterns, thus providing a good compromise between efficiency and fairness. Namely, the weights are adjusted from α_{prev}, β_{prev}, and γ_{prev} to α_{new}, β_{new}, and γ_{new}, respectively, at each round. A final normalization is performed to ensure that the sum of these weights is always 1, thereby stabilizing the resulting values.

Definition 8.2 (Dynamic weight adjustment)**.** Given a rollup r with bid B_r, waiting time W_r and size S_r, the weights α, β, and γ in its priority score P_r (Eq. 8.3) are initially set equal to balance their impact. At each subsequent round of rollup construction, they are adjusted to their new values, starting from their previous values, by embedding the current status of the system inside an exponential smoothing with factor $\lambda \in [0, 1]$. This ensures smooth transitions and avoids sudden weight changes (higher λ's entail more rapid changes and vice versa):

$$\alpha_{new} = \lambda \times \alpha_{current} + (1 - \lambda) \times \alpha_{prev} \tag{8.7}$$
$$\beta_{new} = \lambda \times \beta_{current} + (1 - \lambda) \times \beta_{prev} \tag{8.8}$$
$$\gamma_{new} = \lambda \times \gamma_{current} + (1 - \lambda) \times \gamma_{prev}. \tag{8.9}$$

The current state of the system is expressed by $\alpha_{current}$, $\beta_{current}$, and $\gamma_{current}$, where

$$\alpha_{current} = \frac{average\ bid}{max\ bid} \tag{8.10}$$

is the ratio between the average and maximum bids of the pending rollups and reflects the overall willingness of the users to pay for blob storage.

$$\beta_{current} = \frac{median\ time}{target\ wait} \tag{8.11}$$

is the ratio between the median waiting time of the pending rollups and the target, the desired waiting time for the system, and reflects its fairness and latency.

$$\gamma_{current} = \frac{average\ size}{S_{max}} \tag{8.12}$$

is the average size of pending rollups relative to the maximum rollup size. It indicates how efficiently the system utilizes storage.

This dynamic adjustment ensures that weights gradually adapt to new network conditions while still preserving scoring stability. In this way, the AMFS system balances incentives for higher bids (to maximize revenue), minimizes waiting times (for fairness), and optimizes storage use (to enhance efficiency).

Property 8.1 (Monotonicity)**.** *For fixed normalization bounds and fixed weights $\alpha, \beta, \gamma \geq 0$ with $\alpha + \beta + \gamma = 1$, the priority score P_r is non-decreasing in each of B_r, W_r, and S_r when the other two are held constant.*

Proof. Given the formula of P_r in Eq. 8.3, we can notice that each normalized factor $B_{norm,r}$, $W_{norm,r}$, and $S_{norm,r}$ is a non-decreasing function of B_r, W_r, and S_r, respectively, and P_r is a convex combination of these terms. Increasing the bid, waiting time, or data size of a rollup within a given round cannot, therefore, decrease its score. □

Property 8.2 (Fairness)**.** *The AMFS mechanism increases the relative priority of rollups that remain pending for multiple rounds, allowing them to be scheduled after a reasonable amount of time.*

Proof. The normalized waiting-time term $W_{norm,r}$ contributes directly to P_r, and when the median waiting time exceeds the target, $\beta_{current}$ in Eq. 8.11 increases before smoothing and normalization. Therefore, under congestion, the mechanism prioritizes waiting time more strongly and reduces the risk that low-bid or small rollups are persistently deprioritized. □

Property 8.3 (Resistance to manipulation)**.** *The AMFS score is bounded and cannot be inflated arbitrarily by changing a single input factor.*

Proof. Since the normalized variables in Eqs. 8.4–8.6 lie in $[0, 1]$ and the weights satisfy $\alpha + \beta + \gamma = 1$, the score always satisfies $0 \leq P_r \leq 1$. Because the weights α, β, and γ are computed from aggregate network statistics, individual rollups have only limited influence over the weight adjustment in Eqs. 8.10, 8.11, and 8.12. □

8.4 Shared Blob Structure and Smart Contracts

The shared blob data scheme is essential for efficient data management. The proposed scheme manages and aggregates data from multiple rollups to optimize blob storage and reduce the costs of posting data to blobs. The primary goal is to ensure the maximum utilization of blob space while maintaining fair access to shared blob data for all participants. Fig. 8.2 illustrates the structure of a shared blob containing

multiple, sequentially stored rollups. Several rollups can be included in a single blob until all 128 KB of its space is filled. As illustrated, every single rollup consists of five data segments. Moreover, rollups within the same blob can have different sizes, as they contain both fixed and variable parts.

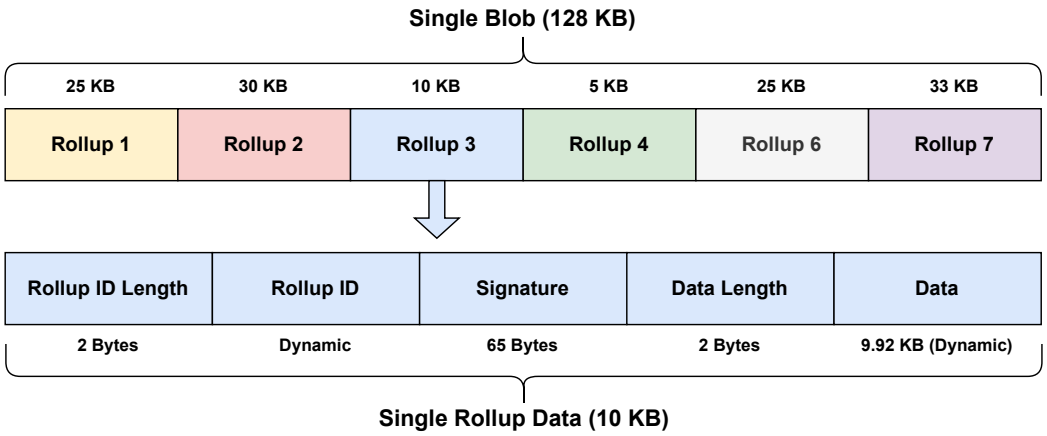


Figure 8.2: Shared blob data schema.

Each rollup begins with a fixed-size field that contains the length of the rollup identifier. This segment accommodates variable-sized identifiers with different configurations and versions without changing the underlying parsing logic, as it dynamically determines their offsets. The subsequent segment is the rollup identifier, which is crucial for retrieving the actual rollup data stored inside the shared blob. After that, there is the rollup data signature, whose size is fixed to 65 bytes, a common length for blockchain and rollup signatures. This cryptographic signature verifies the integrity of the data stored in the blob during retrieval. The fixed signature size simplifies parsing while ensuring consistent security across all data blobs. The next segment contains a length indicator for the data. Similar to the rollup identifier length, it allows dynamic parsing of variable-length data. The last segment is the actual rollup data. In blockchain systems, hexadecimal data representation is a widely used technique for data management. Therefore, the hexadecimal representation is used to merge data from multiple rollups. Moreover, the offset provides a uniform, concise way to precisely locate dynamic-length segments during parsing. Furthermore, this enables a scalable and adaptable approach to introduce new or extended data types without impacting existing systems.

As shown in Fig. 8.2, storing more rollups of different sizes within a single blob incurs overhead due to metadata computation and storage. In particular, part of the blob space stores information describing the structure of each rollup. However, the following section demonstrates that this overhead is acceptable and improves the system's performance compared to using fixed-sized blobs without sharing.

8.5 Experimental Setup and Evaluation

A series of experiments is conducted to assess the efficiency, fairness, and security of AMFS. These experiments evaluate the effectiveness of AMFS in optimizing

transaction processing time and resource utilization to achieve equitable resource distribution and access under various scenarios, while maintaining resilience against security vulnerabilities. The experiments were conducted on an Intel Core i7-3.60 GHz processor with 16 GB of RAM. Tab. 8.3 summarizes all tools and technologies used. The source code is available on a GitHub repository[1].

Table 8.3: Tools and technologies used in the experiments.

Technology	**Version/Network**
Ethereum Test Network	Sepolia
Solidity	0.8.20
Contract Deployment	Remix IDE
Node Provider	Infura
Blockchain Interaction	Viem.js 2.21.48
Backend Framework	Express.js 4.21.1
Frontend Framework	React.js 18.3.1
KZG Commitment	c-kzg 2.1.2
Signature	ECDSA
Database	MongoDB 6.10.0

8.5.1 Evaluation

The roles and effects of four factors are considered: the value of the parameter λ, the fairness of the system across different kinds of rollups, the efficient use of storage, and the cost of the solution.

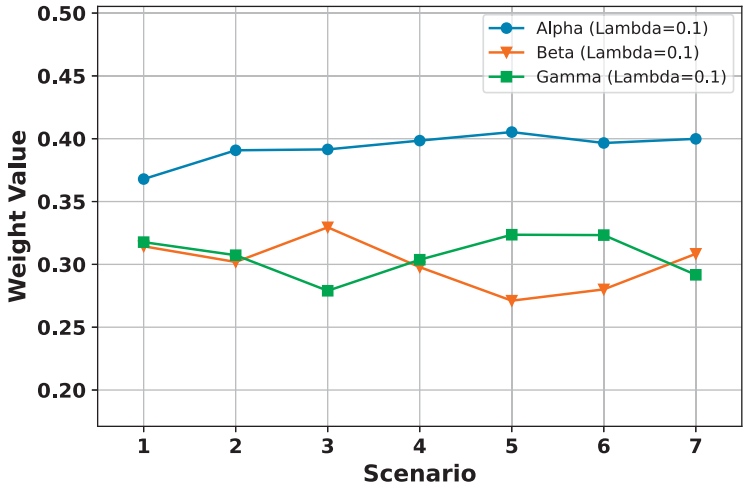
Impact of Lambda

The λ parameter is critical in the AMFS, as it affects the system's response to changes in bid, waiting time, and storage utilization. These changes adjust the weights of α, which prioritizes rollups with higher bids, β, which prioritizes rollups with higher waiting times, and γ, which prioritizes larger rollups to fill a blob quickly.

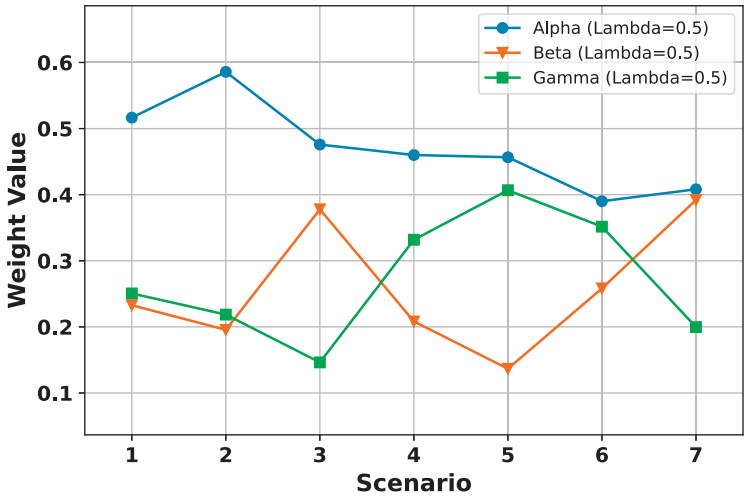
Fig. 8.3 illustrates the variation of these parameters across different scenarios and under varying λ settings, reflecting their sensitivity to current network dynamics and historical network demand. Tab. 8.4 summarizes the characteristics of the seven considered scenarios, which essentially represent different network conditions.

In Fig. 8.3a, the value of λ is set to 0.1, which means that it assigns more weight to historical values rather than the current one. Therefore, α, β, and γ are more stable against sudden variations in bid, waiting time, or rollup size, resulting in a more stable system. In Fig. 8.3b, the λ is increased to 0.5, which results in a more responsive system and allows it to adjust more to dynamic system changes. In this case, the importance of historical and contingent factors is balanced, which is crucial for system stability and adaptability, particularly in managing higher bid fluctuations. In Fig. 8.3c, the parameters are more responsive to recent network conditions, with

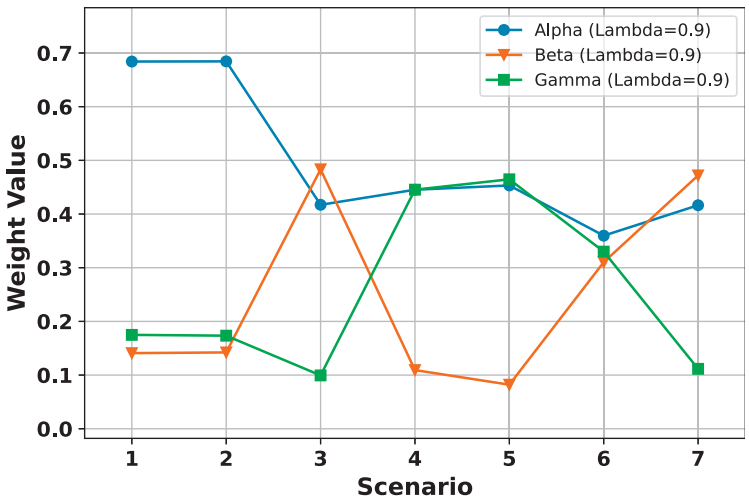
[1]https://github.com/MuhammadBinSaif/AMFS



(a) Lambda = 0.1



(b) Lambda = 0.5



(c) Lambda = 0.9

Figure 8.3: Variation of α, β, and γ with different λ values.

a higher λ value of 0.9. All three weights, α, β, and γ, are updated more frequently, providing a robust approach to handling sudden fluctuations, particularly when the

Table 8.4: Values for different rounds.

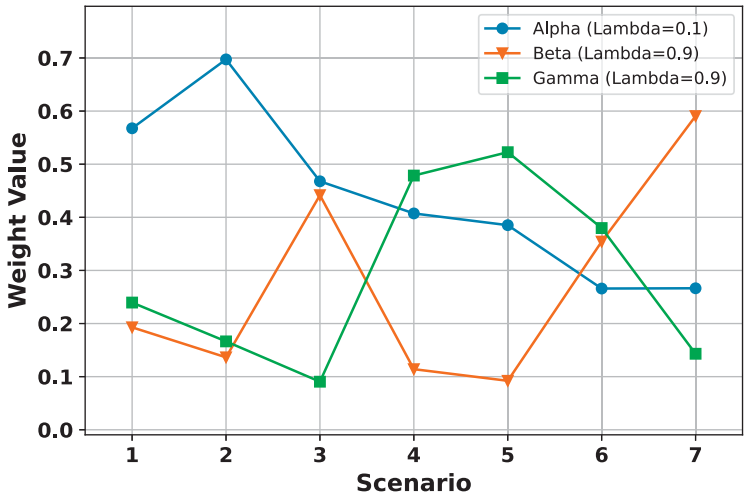
Scenario	Average Bid (Gwei)	Median Wait (Rounds)	Average Data (KB)
1	0.00006	1	17.03
2	0.00470	1	15.56
3	0.00005	7	16.38
4	0.00006	1	78.75
5	0.00552	1	74.93
6	0.00006	6	76.31
7	0.05470	7	17.52

waiting time and storage utilization are high.

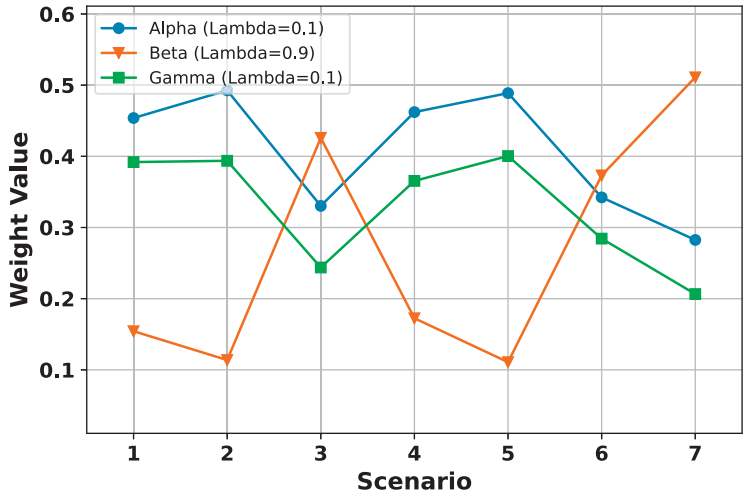
However, since the three weights α, β, and γ play different roles in system performance, addressing the distinct requirements for responsiveness and stability for each parameter can be achieved by using a different λ for each. This approach can help optimize fairness, efficiency, and storage utilization. Fig. 8.4 shows the variation of α, β, and γ under different λ settings for each parameter. As seen in Fig. 8.4a, a value of $\lambda = 0.1$ for α minimizes the impact of a sudden increase in a bid, resulting in a more stable α. However, it ignores the recent trend in dynamic market conditions. Therefore, a balanced $\lambda = 0.5$ for α can adapt to historical and recent bidding trends, as seen in Fig. 8.4c. Conversely, a value of $\lambda = 0.1$ for β could lead to a slow response to sudden network congestion. As shown in Fig. 8.4, a higher $\lambda = 0.9$ prioritizes the current network congestion, ensuring fairness for small rollups. Similarly, Fig. 8.4b shows that $\lambda = 0.1$ for γ ignores the recent changes in storage utilization, which could lead to inefficient blob usage. Fig. 8.4c illustrates that a slightly higher value of $\lambda = 0.7$ for γ allows it to adapt to current storage conditions and ensures an efficient blob utilization. Therefore, it is interesting to identify an optimal configuration in which the three weights change smoothly, without sudden jumps. This optimal configuration has been identified, after a set of iterative tests to balance profitability (bids), fairness (waiting time), and efficient use of blobs (size), as 0.5 for α (i.e., $\lambda(\alpha) = 0.5$), 0.9 for β (i.e., $\lambda(\beta) = 0.9$), and 0.7 for γ (i.e., $\lambda(\gamma) = 0.7$). In general, a moderate $\lambda(\alpha)$ prevents high-bid rollups from surpassing weaker bids, reducing excessive volatility. A greater $\lambda(\beta)$ prioritizes older transactions during excessive waiting times, preventing smaller bid rollups from accumulating indefinitely. A mid-range $\lambda(\gamma)$ adjusts to real-time size changes without preferring huge rollups. Although these selections are heuristic rather than mathematically established optima, they performed well across various scenarios in tests.

Fairness

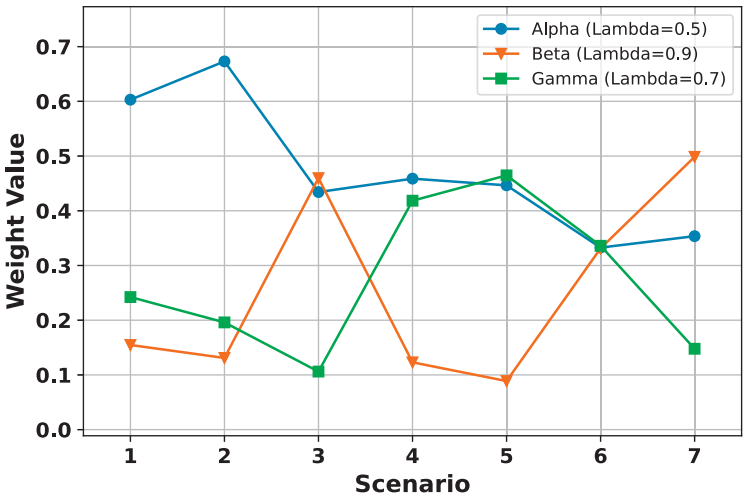
Fairness can be defined as the provision of a fair opportunity for all participating rollups to have their data included in a blob promptly, regardless of bids or data size. Therefore, ensuring fairness for small rollups with small bids and lower data sizes is critical. Fig. 8.5 compares the median waiting time for small and large rollups in three cases. To ensure that no rollup experiences a long inclusion delay, the target



(a) $\lambda(\alpha)$=0.1, $\lambda(\beta)$=0.9, $\lambda(\gamma)$=0.9



(b) $\lambda(\alpha)$=0.1, $\lambda(\beta)$=0.9, $\lambda(\gamma)$=0.1



(c) $\lambda(\alpha)$=0.5, $\lambda(\beta)$=0.9, $\lambda(\gamma)$=0.7

Figure 8.4: Variation of α, β, and γ with different λ values and weight configurations.

waiting time is set to ten rounds. In the first case, when the average size of small rollups is 16 KBytes, the average bid is 0.00005 Gwei. The average size of large

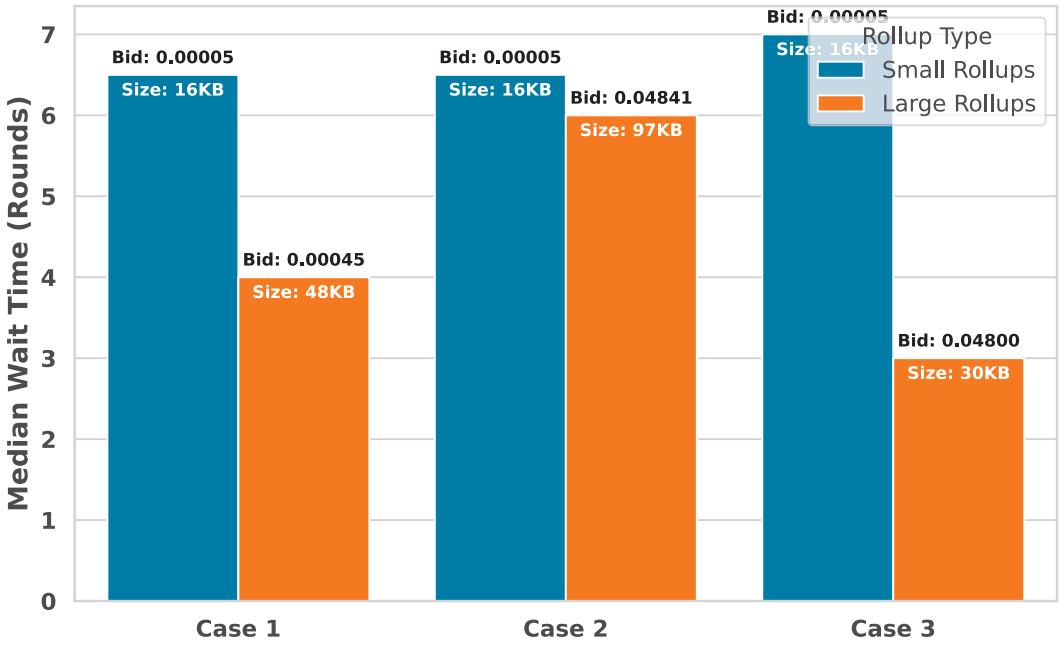


Figure 8.5: Waiting time for small rollups.

rollups is 48 KBytes, and the average bid is 0.000448 Gwei. Small rollups experience slightly longer waiting times than large rollups due to lower bids; however, the median waiting time for both rollups is less than ten rounds. This shows that small rollups do not experience critically longer delays despite lower bids. In the second case, AMFS prefers rollups with larger data for blob inclusion to fill the blob quickly, thereby increasing the median waiting time for large rollups. However, the median waiting time for large rollups does not exceed the threshold, indicating a balance between efficiency and fairness. Similarly, in the third case, the waiting time for small rollups remains under ten rounds, despite lower bids with respect to large rollups. Meanwhile, the adaptive response of the system to data sizes and bidding dynamics is evident in the lower delays for large rollups.

Storage

Compared to the actual rollup data size, the stored data size increases slightly because of the metadata overhead of the proposed technique. However, this overhead is important for efficient data storage and retrieval in the shared blob. Fig. 8.6 reports the percentage of increase in data size due to rollup-specific and blob-sharing information. The increase is even less than 1%, particularly for larger rollups, and slightly over 1% for smaller ones. The added data helps separate rollups merged into a shared blob more easily, improves data extraction time, and guarantees reactive data verification.

Furthermore, the proposed AMFS prioritizes larger data segments in order to fill the blob quickly, as case 2 of Fig. 8.5 shows. The blob can reach the 128 KBytes threshold more quickly by prioritizing larger data segments over smaller ones, thereby reducing the per-unit time required to create and post the blob. While this approach might increase waiting times for some rollups, it improves overall

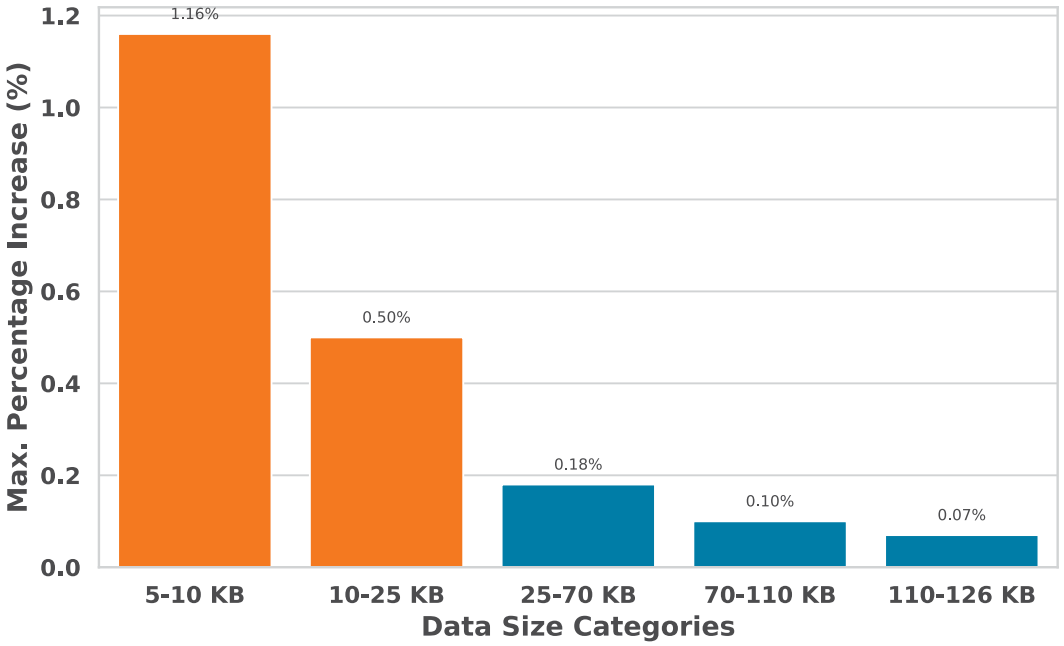


Figure 8.6: Percentage increase in rollup data by size.

throughput and efficiency.

Cost

In addition to improving storage utilization and fairness for small rollups, the shared blob significantly reduces the cost of posting data to the blob. The actual cost of blob posting in a shared blob only depends on rollup bids. However, to evaluate the cost of the proposed solution, it is assumed that the blob is shared equally across rollups of equal size and that the cost is divided equally among all rollups. The cost of posting data to a shared blob is evaluated in three different scenarios: 1) if the blob is shared among 12 rollups with 10 KBytes of data each, 2) if the blob is shared among 5 rollups with 25 KBytes of data each, and 3) if the blob is shared among 2 rollups with 64 KBytes of data each. Fig. 8.7 shows the cost of posting data to a shared blob, a single blob, and calldata. To ensure a realistic pricing baseline, blob-related costs are reported by separating the blob fee from the blob transaction (execution) fee. The blob fee captures the data-availability charge for including one blob and, in shared-blob scenarios, it is amortized across rollups; thus, a single blob posting costs approximately 0.000131072 Gwei, while sharing the blob among 12 rollups reduces the per-rollup blob fee to approximately 0.000010923 Gwei. In addition, submitting the blob requires a Type-3 (EIP-4844) commitment transaction, which incurs an L1 execution fee (reported separately as the blob transaction fee) that reflects prevailing EIP-1559 gas prices. For comparison, the calldata fee represents the cost of publishing the same payload directly as calldata on L1 and is substantially higher (e.g., approximately 0.0008970 ETH for 10 KBytes). Even for medium (25 KBytes) or large (64 KBytes) data, sharing blobs remains markedly cheaper than using calldata, and the blob fee is also lower than posting an individual blob per rollup.

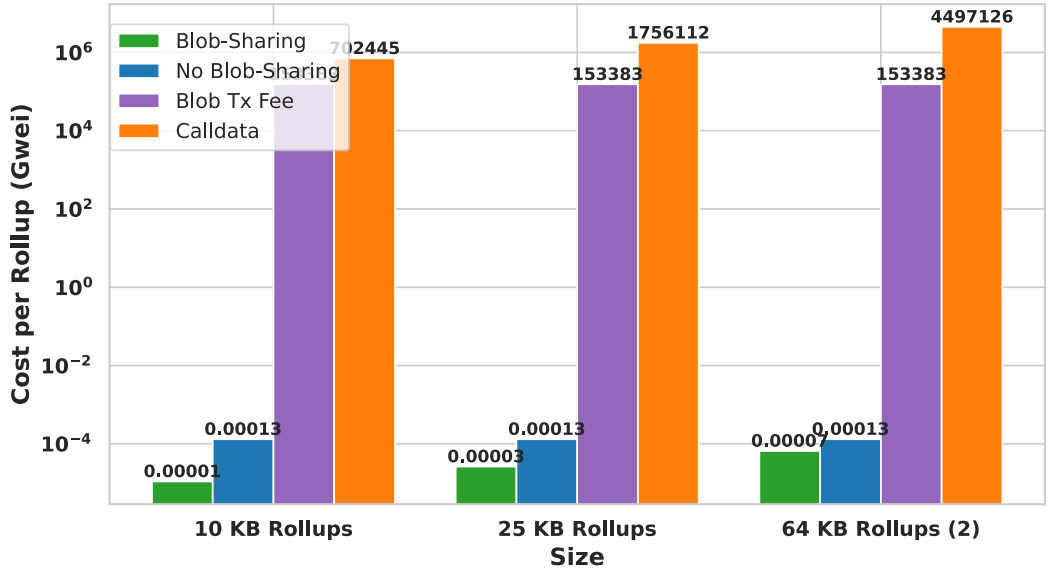


Figure 8.7: Cost comparison by data size.

Adversarial and Congestion Conditions

The current evaluation does not include a dedicated simulation evaluation for fully adversarial scheduling or high-congestion scenarios. However, the expected behavior of AMFS in such conditions can be interpreted from Properties 8.1–8.3. Since each normalized factor is bounded in $[0, 1]$ and the weights satisfy $\alpha + \beta + \gamma = 1$, no single factor can increase priority without limit. If a rollup raises its bid and improves its score, this effect remains bounded by the normalization structure. Under blob scarcity or sustained congestion, the median waiting time increases, raising $\beta_{current}$ in Eq. 8.11 and, after exponential averaging and normalization, increasing the contribution of waiting time to P_r. This shifts priority toward older pending rollups and mitigates the risk that small or low-bid rollups are persistently deprioritized, in line with the fairness-oriented behavior described in 8.2. When rollup sizes are highly uneven, the waiting-time component progressively compensates for repeated exclusion, consistent with the results in Fig. 8.5. A full experimental study of adversarial scheduling and persistent congestion is left for future work.

8.6 Chapter Summary

This chapter addressed the data availability problem in L2 rollup-based solutions, which represents one of the main challenges in ensuring scalability while preserving the security guarantees of L1 blockchains. After surveying and comparing existing approaches, the chapter highlighted that while on-chain solutions ensure strong security properties, they incur high costs and long wait times, whereas off-chain approaches may introduce data security risks. The analysis further showed that introducing EIP-4844 blobs significantly reduces the cost of posting rollup data compared to calldata-based approaches; however, blobs still have limitations due to their fixed capacity and independent use per rollup. In this context, the chapter proposed

a solution based on the concept of shared blobs, enabling multiple rollups to publish their data within a single blob. By sharing blob space among rollups and prioritizing rollup data using an AMFS mechanism, the proposed solution improves blob utilization and reduces the cost of data availability, particularly for small rollups. The evaluation demonstrates that this approach provides a cost-efficient, practical solution for ensuring data availability in rollup-based L2 systems while preserving the trust assumptions of the underlying L1 blockchain.

Chapter 9

Conclusion

This chapter summarizes and consolidates the contributions presented in this thesis, outlining some future extensions.

Blockchain technology has demonstrated significant potential; however, its adoption in real-world systems faces critical challenges. In particular, smart contracts lack modularity and robust access control, storage and off-chain data management pose risks to availability, consistency, and privacy, certification systems must support verifiability without exposing sensitive information, blockchain immutability conflicts with regulatory requirements such as the GDPR, while scalability remains limited, and data availability becomes a central requirement in rollup-based designs. The contributions of this thesis address these challenges across different layers of the blockchain ecosystem, collectively improving the role of blockchain as a trustworthy, scalable, privacy-preserving, and legally compliant infrastructure for critical applications.

9.1 Summary of Contributions

This thesis presents five key contributions, each developed as a dedicated chapter and addressing a specific challenge to blockchain technology.

Chapter 4 introduced a hierarchical factory structure with integrated multirole authentication and authorization to improve scalability, modularity, and governance in smart contracts. The solution supports the recursive instantiation of factories and contracts, reflects real organizational hierarchies, and integrates access control into the design. Using a water management system as a motivating example, the implementation demonstrated that while initial deployment at the first level of the hierarchy introduces a small overhead, cumulative savings at deeper levels lead to significant cost reductions. Security testing supported the robustness of the proposed approach against common access and privilege-related threats.

Chapter 5 developed a decentralized and privacy-preserving framework for storing and retrieving data by integrating blockchain anchoring, IPFS storage, and cryptographic techniques. By storing only cryptographic references on-chain and moving encrypted data off-chain, the proposed system ensures immutability, traceability, availability, and privacy. The proposed design maintains confidentiality during retrieval and querying while enabling transparent verification via blockchain anchors,

providing a practical solution for data-intensive applications.

Chapter 6 enhanced the results of the previous chapter by proposing a blockchain-based certification system that utilizes Merkle trees and zero-knowledge proofs to enable the selective disclosure of certificate attributes. This solution enables selective disclosure of certificate attributes, allowing verifiers to confirm authenticity and compliance without accessing sensitive details. The framework ensures both transparency and privacy by anchoring certificates on a blockchain while relying on cryptographic proofs for verification, addressing the "transparency and privacy paradox" of public ledgers.

Chapter 7 addressed the conflict between blockchain immutability and GDPR compliance. The chapter analyzed how blockchain properties can support or hinder data subject rights and compared multiple approaches, from on-chain storage to versioned storage and encryption-based solutions. The solution utilizes Ciphertext-Policy Attribute-Based Encryption (CP-ABE) and Proxy Re-Encryption (PRE) to manage sensitive data in accordance with all eight user rights, such as rectification and erasure, while preserving the audit trail and accountability features that make blockchain technology valuable. This contribution presents an implementation pattern for regulated domains that require transparency and compliance to be addressed simultaneously.

Chapter 8 tackled performance issues by discussing how Layer 2 (L2) protocols mitigate Layer 1 scalability issues and why data availability becomes central in rollups. Scalability and data availability are addressed through a comprehensive survey of L2 solutions, which leads to the development of an AMFS system. This approach aims to strike a balance between scalability, efficiency, and security by leveraging blob storage and dynamic scoring to prioritize rollup data. The AMFS system ensures that transaction data remains accessible for independent verification, preserving the trustless guarantees of blockchain systems even in high-throughput scenarios.

Overall, the thesis demonstrates that blockchain systems can be engineered to support governance, privacy, scalability, and compliance requirements while maintaining verifiability and decentralization, providing practical guidance for future research and the deployment of trustworthy decentralized infrastructures.

9.2 Future Directions

The contributions of this thesis strengthen the role of blockchain as a trustworthy, scalable, and privacy-aware infrastructure for critical real-world applications. This work enhances the transition of blockchain from experimental prototypes to reliable infrastructures by effectively addressing modularity in smart contracts, secure decentralized storage, verifiable certification, regulatory compliance, and L2 performance. While the thesis provides concrete architectures, implementations, and experimental evaluation, several future extensions can be identified.

A more extensive validation in real deployments is still required. The proposed patterns and frameworks were thoroughly tested in controlled environments. However, real-world infrastructures often face operational limitations, including diverse stakeholders, long-term maintenance, network unpredictability, and changing gov-

ernance. Future research should focus on implementing and benchmarking these systems in realistic settings to investigate their long-term scalability, performance overhead, and key management complexity.

While the compliance framework successfully identified a combination of Ciphertext-Policy Attribute-Based Encryption (CP-ABE) and Proxy Re-Encryption (PRE) to support GDPR rights, practical compliance remains subject to varying organizational processes and jurisdictional interpretations.

Current protocol designs and the evolution of the blockchain ecosystem determine the scalability analysis in L2 contexts. The thesis demonstrated that blob-based techniques can significantly reduce cost when compared to calldata-based alternatives. Therefore, integrating these blob-based techniques with comprehensive rollup workflows, such as fraud or validity proof, and testing their performance under actual transaction loads can be the next step.

Finally, the increasing integration of AI-driven systems into critical infrastructure raises new questions about verifiability and trust. As AI models and autonomous agents begin generating data that feeds into decision-making pipelines, the need to verify their correctness without exposing them becomes more pressing. The ZKP-based framework proposed in Chap. 6 is, in principle, well suited for this context: a prover could generate a proof that an AI-generated output satisfies a required compliance condition, without revealing the model, its parameters, or the underlying input. Extending the current framework to handle AI-generated witnesses and investigating how ZKP-based verification can contribute to trustworthy AI pipelines are, therefore, natural and valuable directions for future research. This would position the contributions of this thesis not only within the blockchain ecosystem but also within the broader challenge of building accountable and privacy-preserving infrastructures for AI-assisted applications.

Bibliography

[1] Arun Teja Polcumpally et al. “Blockchain governance and trust: A multi-sector thematic systematic review and exploration of future research directions”. In: *Heliyon* 10.12 (2024).

[2] Javad Zarrin et al. “Blockchain for decentralization of internet: prospects, trends, and challenges”. In: *Cluster Computing* 24.4 (2021), pp. 2841–2866.

[3] Ahmed Afif Monrat, Olov Schelén, and Karl Andersson. “A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities”. In: *IEEE Access* 7 (2019), pp. 117134–117151. DOI: 10.1109/ACCESS.2019.2936094.

[4] Udit Gulati and Mahesh Narayanan. “Blockchain for Critical Infrastructure Security: Applications and Challenges”. In: *2025 5th Intelligent Cybersecurity Conference (ICSC)*. IEEE. 2025, pp. 62–67.

[5] Johannes Sedlmeir et al. “The transparency challenge of blockchain in organizations”. In: *Electronic Markets* 32.3 (2022), pp. 1779–1794.

[6] Juan Benet. “Ipfs-content addressed, versioned, p2p file system”. In: *arXiv preprint arXiv:1407.3561* (2014).

[7] Juha Partala, Tri Hong Nguyen, and Susanna Pirttikangas. “Non-interactive zero-knowledge for blockchain: A survey”. In: *IEEE Access* 8 (2020), pp. 227945–227961.

[8] Vijay Rajasekar et al. “Emerging Design Patterns for Blockchain Applications”. In: *ICSOFT*. 2020, pp. 242–249.

[9] Abdelatif Hafid, Abdelhakim Senhaji Hafid, and Mustapha Samih. “Scaling blockchains: A comprehensive survey”. In: *IEEE access* 8 (2020), pp. 125244–125262.

[10] Nazanin Zahed Benisi, Mehdi Aminian, and Bahman Javadi. “Blockchain-based decentralized storage networks: A survey”. In: *Journal of Network and Computer Applications* 162 (2020), p. 102656.

[11] Muhammad Irfan Khalid et al. “A comprehensive survey on blockchain-based decentralized storage networks”. In: *IEEE Access* 11 (2023), pp. 10995–11015.

[12] AKM Bahalul Haque et al. “GDPR compliant blockchains–a systematic literature review”. In: *Ieee Access* 9 (2021), pp. 50593–50606.

[13] Ahcène Bounceur et al. “The Transparency Challenge in Blockchain-Enabled Sustainable Development Goals Applications: Exploring Privacy-Preserving Techniques and Emerging Platforms”. In: *IEEE Access* (2025).

[14] Jhong-Ting Lou, Showkat Ahmad Bhat, and Nen-Fu Huang. “Blockchain-based privacy-preserving data-sharing framework using proxy re-encryption scheme and interplanetary file system”. In: *Peer-to-Peer Networking and Applications* 16.5 (2023), pp. 2415–2437.

[15] Rosa Pericàs-Gornals, Macià Mut-Puigserver, and M Magdalena Payeras-Capellà. “Highly private blockchain-based management system for digital COVID-19 certificates”. In: *International Journal of Information Security* 21.5 (2022), pp. 1069–1090.

[16] Po-Wen Chi, Yun-Hsiu Lu, and Albert Guan. “A privacy-preserving zero-knowledge proof for blockchain”. In: *Ieee Access* 11 (2023), pp. 85108–85117.

[17] Jorge Bernal Bernabe et al. “Privacy-preserving solutions for blockchain: Review and challenges”. In: *Ieee Access* 7 (2019), pp. 164908–164940.

[18] Rahime Belen-Saglam et al. “A systematic literature review of the tension between the GDPR and public blockchain systems”. In: *Blockchain: Research and Applications* 4.2 (2023), p. 100129.

[19] Youssef Said et al. “The impact of blockchain on the banking sector: A systematic review of applications, challenges, and future directions”. In: *Asia and the Global Economy* 6.1 (2026), p. 100133.

[20] Han Song, Zhongche Qu, and Yihao Wei. “Advancing blockchain scalability: An introduction to layer 1 and layer 2 solutions”. In: *2024 IEEE 2nd International Conference on Sensors, Electronics and Computer Engineering (ICSECE)*. IEEE. 2024, pp. 71–76.

[21] Peiyao Sheng et al. “Proof of diligence: Cryptoeconomic security for rollups”. In: *arXiv preprint arXiv:2402.07241* (2024).

[22] Kaidong Wu et al. “A first look at blockchain-based decentralized applications”. In: *Software: Practice and Experience* 51.10 (2021), pp. 2033–2050.

[23] Jason Paul Cruz, Yuichi Kaji, and Naoto Yanai. “RBAC-SC: Role-based access control using smart contract”. In: *Ieee Access* 6 (2018), pp. 12240–12251.

[24] Bowen Liu, Siwei Sun, and Pawel Szalachowski. “Smacs: smart contract access control service”. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2020, pp. 221–232.

[25] Pan Yang, Naixue Xiong, and Jingli Ren. “Data security and privacy protection for cloud storage: A survey”. In: *IEEE Access* 8 (2020), pp. 131723–131740.

[26] G. Capece, N. L. Ghiron, and F. Pasquale. “Blockchain Technology: Redefining Trust for Digital Certificates”. In: *Sustainability* (2020). DOI: 10.3390/su12218952.

[27] Cristòfol Daudén-Esmel et al. “Lightweight blockchain-based platform for GDPR-compliant personal data management”. In: *2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP)*. IEEE. 2021, pp. 68–73.

[28] Nguyen Binh Truong et al. “GDPR-compliant personal data management: A blockchain-based solution”. In: *IEEE Transactions on Information Forensics and Security* 15 (2019), pp. 1746–1761.

[29] Muhammad Bin Saif, Sara Migliorini, and Fausto Spoto. “A Survey on Data Availability in Layer 2 Blockchain Rollups: Open Challenges and Future Improvements”. In: *Future Internet* 16.9 (2024). ISSN: 1999-5903. DOI: 10.3390/fi16090315. URL: https://doi.org/10.3390/fi16090315.

[30] Suhyeon Lee. “180 Days After EIP-4844: Will Blob Sharing Solve Dilemma for Small Rollups?” In: *arXiv preprint arXiv:2410.04111* (2024). URL: https://arxiv.org/html/2410.04111v1.

[31] Davide Crapis, Edward W Felten, and Akaki Mamageishvili. “EIP-4844 economics and rollup strategies”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2024, pp. 135–149. URL: https://doi.org/10.1007/978-3-031-69231-4_10.

[32] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: (2008).

[33] Oluwafemi Akanfe, Diane Lawong, and H Raghav Rao. “Blockchain technology and privacy regulation: Reviewing frictions and synthesizing opportunities”. In: *International Journal of Information Management* 76 (2024), p. 102753.

[34] Anulipt Chandan, Vidyasagar Potdar, and Michele John. “Systematic literature review of blockchain technology’s technical challenges: A tertiary study”. In: *Information* 15.8 (2024), p. 475.

[35] Jie Xu, Cong Wang, and Xiaohua Jia. “A survey of blockchain consensus protocols”. In: *ACM Computing Surveys* 55.13s (2023), pp. 1–35.

[36] Chandranshu Gupta and Asmita Mahajan. “Evaluation of proof-of-work consensus algorithm for blockchain networks”. In: *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE. 2020, pp. 1–7.

[37] Sheikh Munir Skh Saad and Raja Zahilah Raja Mohd Radzi. “Comparative review of the blockchain consensus algorithm between proof of stake (pos) and delegated proof of stake (dpos)”. In: *International Journal of Innovative Computing* 10.2 (2020).

[38] Gary Shapiro, Christopher Natoli, and Vincent Gramoli. “The performance of Byzantine fault tolerant blockchains”. In: *2020 IEEE 19th international symposium on network computing and applications (NCA)*. IEEE. 2020, pp. 1–8.

[39] Karthik Sai RadhaKrishna Puranam et al. “Anatomy and lifecycle of a bitcoin transaction”. In: *Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM), Amity University Rajasthan, Jaipur-India*. 2019.

[40] Emmanuelle Anceaume et al. “On finality in blockchains”. In: *arXiv preprint arXiv:2012.10172* (2020).

[41] Eugenia Politou et al. “Blockchain mutability: Challenges and proposed solutions”. In: *IEEE Transactions on Emerging Topics in Computing* 9.4 (2019), pp. 1972–1986.

[42] Ju Myung Song, Jongwook Sung, and Taeho Park. “Applications of blockchain to improve supply chain traceability”. In: *Procedia Computer Science* 162 (2019), pp. 119–122.

[43] Rajeev Singh, Sudeep Tanwar, and Teek Parval Sharma. “Utilization of blockchain for mitigating the distributed denial of service attacks”. In: *Security and Privacy* 3.3 (2020), e96.

[44] Vitalik Buterin et al. “A next-generation smart contract and decentralized application platform”. In: *white paper* 3.37 (2014), pp. 2–1.

[45] Shafaq Naheed Khan et al. “Blockchain smart contracts: Applications, challenges, and future trends”. In: *Peer-to-peer Networking and Applications* 14.5 (2021), pp. 2901–2925.

[46] Kose John, Leonid Kogan, and Fahad Saleh. “Smart contracts and decentralized finance”. In: *Annual Review of Financial Economics* 15.1 (2023), pp. 523–542.

[47] Asma Khatoon. “A blockchain-based smart contract system for healthcare management”. In: *Electronics* 9.1 (2020), p. 94.

[48] Ilhaam A Omar et al. “Enhancing vendor managed inventory supply chain operations using blockchain smart contracts”. In: *IEEE access* 8 (2020), pp. 182704–182719.

[49] Sara Migliorini, Mauro Gambini, and Alberto Belussi. “A blockchain-based platform for ensuring provenance and traceability of donations for cultural heritage”. In: *Blockchain: Research and Applications* (2025), p. 100278.

[50] Tuan-Dung Tran et al. “CrossCert: A privacy-preserving cross-chain system for educational credential verification using zero-knowledge proof”. In: *International Conference on Industrial Networks and Intelligent Systems*. Springer. 2024, pp. 256–271.

[51] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. “General State Channel Networks”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’18. Toronto, Canada, 2018, pp. 949–966. DOI: 10.1145/3243734.3243856.

[52] Lydia D. Negka and Georgios P. Spathoulas. “Blockchain State Channels: A State of the Art”. In: *IEEE Access* 9 (2021), pp. 160277–160298. DOI: 10.1109/ACCESS.2021.3131419.

[53] Amritraj Singh et al. “Sidechain technologies in blockchain networks: An examination and state-of-the-art review”. In: *Journal of Network and Computer Applications* 149 (2020), p. 102471. DOI: 10.1016/j.jnca.2019.102471.

[54] Joseph Poon and Vitalik Buterin. *Plasma: Scalable autonomous smart contracts*. Available at https://www.plasma.io/plasma.pdf. 2017.

[55] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. “Blockchain Scaling Using Rollups: A Comprehensive Survey”. In: *IEEE Access* 10 (2022), pp. 93039–93054. DOI: 10.1109/ACCESS.2022.3200051.

[56] Barry Whitehat. *Roll Up: Scale Ethereum with SNARKs*. https://github.com/barryWhiteHat/roll_up. Accessed: 2024-07-15. 2018.

[57] Thomas Lavaur, Jérôme Lacan, and Caroline P. C. Chanel. “Enabling Blockchain Services for IoE with Zk-Rollups”. In: *Sensors* 22.17 (2022). DOI: 10.3390/s22176493.

[58] Imad Aad. “Zero-Knowledge Proof”. In: *Trends in Data Protection and Encryption Technologies*. Ed. by Valentin Mulder et al. Cham: Springer Nature Switzerland, 2023, pp. 25–30. ISBN: 978-3-031-33386-6. DOI: 10.1007/978-3-031-33386-6_6.

[59] Zhixin Ren et al. “Blockchain-based CP-ABE data sharing and privacy-preserving scheme using distributed KMS and zero-knowledge proof”. In: *Journal of King Saud University-Computer and Information Sciences* 36.3 (2024), p. 101969.

[60] Hongbo Wen et al. “Practical Security Analysis of {Zero-Knowledge} Proof Circuits”. In: *33rd USENIX Security Symposium (USENIX Security 24)*. 2024, pp. 1471–1487.

[61] Marta Bellés-Muñoz et al. “Circom: A Circuit Description Language for Building Zero-Knowledge Applications”. In: *IEEE Transactions on Dependable and Secure Computing* 20.6 (2023), pp. 4733–4751. DOI: 10.1109/TDSC.2022.3232813.

[62] Jens Groth. “On the size of pairing-based non-interactive arguments”. In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2016, pp. 305–326.

[63] Iden3. *snarkjs: zkSNARK implementation in JavaScript*. 2023. URL: https://github.com/iden3/snarkjs.

[64] Nir Bitansky et al. “From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again”. In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS ’12. Cambridge, Massachusetts: Association for Computing Machinery, 2012, pp. 326–349. DOI: 10.1145/2090236.2090263.

[65] Eli Ben-Sasson et al. “Scalable, transparent, and post-quantum secure computational integrity”. In: *IACR Cryptol. ePrint Arch.* (2018), p. 46. URL: https://starkware.co/wp-content/uploads/2022/05/STARK-paper.pdf.

[66] Peng Zhang et al. “Applying software patterns to address interoperability in blockchain-based healthcare apps”. In: *arXiv preprint arXiv:1706.03700* (2017).

[67] Bowen Liu, Siwei Sun, and Pawel Szalachowski. “SMACS: Smart Contract Access Control Service”. In: *50th Annual IEEE/IFIP Inter. Conf. on Dependable Systems and Networks*. 2020, pp. 221–232. DOI: 10.1109/DSN48063.2020.00039.

[68] Yiyun Zhou et al. “Improving IoT Services in Smart-Home Using Blockchain Smart Contract”. In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, pp. 81–87. DOI: 10.1109/Cybermatics_2018.2018.00047.

[69] Wenjun Xia, Xiaohong Chen, and Chao Song. “A framework of blockchain technology in intelligent water management”. In: *Frontiers in Environmental Science* 10 (2022), p. 909606.

[70] Roberto Leonardo Rana, Nino Adamashvili, and Caterina Tricase. “The impact of blockchain technology adoption on tourism industry: a systematic literature review”. In: *Sustainability* 14.12 (2022), p. 7383.

[71] Rui P Pinto, Bruno MC Silva, and Pedro RM Inácio. “A system for the promotion of traceability and ownership of health data using blockchain”. In: *IEEE Access* 10 (2022), pp. 92760–92773.

[72] Lukas Stockburger et al. “Blockchain-enabled decentralized identity management: The case of self-sovereign identity in public transportation”. In: *Blockchain: Research and Applications* 2.2 (2021), p. 100014.

[73] Anwar ul Haque, M. Sayeed Ghani, and Tariq Mahmood. “Decentralized Transfer Learning using Blockchain & IPFS for Deep Learning”. In: *2020 International Conference on Information Networking (ICOIN)*. 2020, pp. 170–177. DOI: 10.1109/ICOIN48656.2020.9016456.

[74] Meet Shah et al. “Decentralized Cloud Storage Using Blockchain”. In: *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*. 2020, pp. 384–389. DOI: 10.1109/ICOEI48184.2020.9143004.

[75] J Hao, Yan Sun, and Hong Luo. “A safe and efficient storage scheme based on blockchain and IPFS for agricultural products tracking”. In: *J. Comput* 29.6 (2018), pp. 158–167.

[76] Maruti M Arer, Praveen M Dhulavvagol, and SG Totad. “Efficient big data storage and retrieval in distributed architecture using blockchain and ipfs”. In: *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*. IEEE. 2022, pp. 1–6.

[77] Syed Saud Hasan, Nazatul Haque Sultan, and Ferdous Ahmed Barbhuiya. “Cloud data provenance using IPFS and blockchain technology”. In: *Proceedings of the Seventh International Workshop on Security in Cloud Computing*. 2019, pp. 5–12.

[78] Xiaochen Zheng et al. “Decentralized industrial IoT data management based on blockchain and IPFS”. In: *IFIP International Conference on Advances in Production Management Systems*. Springer. 2020, pp. 222–229.

[79] Huawei Huang et al. “When blockchain meets distributed file systems: An overview, challenges, and open issues”. In: *IEEE Access* 8 (2020), pp. 50574–50586.

[80] Muneeb Ul Hassan, Mubashir Husain Rehmani, and Jinjun Chen. “Privacy preservation in blockchain based IoT systems: Integration issues, prospects, challenges, and future research directions”. In: *Future Generation Computer Systems* 97 (2019), pp. 512–529.

[81] Abylay Satybaldy and Mariusz Nowostawski. “Review of techniques for privacy-preserving blockchain systems”. In: *Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure*. 2020, pp. 1–9.

[82] Shangping Wang, Xu Wang, and Yaling Zhang. “A Secure Cloud Storage Framework With Access Control Based on Blockchain”. In: *IEEE Access* 7 (2019), pp. 112713–112725. DOI: 10.1109/ACCESS.2019.2929205.

[83] Meng Luo et al. “On the Complexity of the Web’s PKI: Evaluating Certificate Validation of Mobile Browsers”. In: *IEEE Transactions on Dependable and Secure Computing* 21 (2024), pp. 419–433. DOI: 10.1109/TDSC.2023.3255869.

[84] R. Priyadarshini et al. “A Faster, Integrated, and Trusted Certificate Authentication and Issuer Validation System Based on Blockchain”. In: *IEEE Access* 13 (2025), pp. 27037–27049. DOI: 10.1109/ACCESS.2025.3539180.

[85] Avni Rustemi et al. “A systematic literature review on blockchain-based systems for academic certificate verification”. In: *Ieee Access* 11 (2023), pp. 64679–64696.

[86] Mamta Bhamare et al. “Blockchain-Driven Automation for Degree Certificate Authentication While Preserving Data Integrity”. In: *2024 International Conference on Progressive Innovations in Intelligent Systems and Data Science (ICPIDS)*. 2024, pp. 266–274. DOI: 10.1109/ICPIDS65698.2024.00050.

[87] Nero Chaniago, Parman Sukarno, and Aulia Arif Wardana. “Electronic document authenticity verification of diploma and transcript using smart contract on Ethereum blockchain”. In: 7 (2021), pp. 149–163. DOI: 10.26594/REGISTER.V7I2.1959.

[88] M Shakila and A Rama. “Design and analysis of digital certificate verification and validation using blockchain-based technology”. In: *2023 Eighth International Conference on Science Technology Engineering and Mathematics (ICONSTEM)*. IEEE. 2023, pp. 1–9.

[89] Yves Christian Elloh Adja et al. “A blockchain-based certificate revocation management and status verification system”. In: *Computers & Security* 104 (2021), p. 102209.

[90] Nitima Malsa et al. “Framework and smart contract for blockchain enabled certificate verification system using robotics”. In: *Machine Learning for Robotics Applications*. Springer, 2021, pp. 125–138.

[91] Harshita Khandelwal et al. “Certificate verification system using blockchain”. In: *Advances in cybernetics, cognition, and machine learning for communication technologies*. Springer, 2020, pp. 251–257.

[92] Dodo Khan, Low Tang Jung, and Manzoor Ahmed Hashmani. “Systematic literature review of challenges in blockchain scalability”. In: *Applied Sciences* 11.20 (2021), p. 9372.

[93] Hanlei Cheng et al. “A permissioned blockchain-based platform for education certificate verification”. In: *International Conference on Blockchain and Trustworthy Systems*. Springer. 2020, pp. 456–471.

[94] Juan Alamrio Berrios Moya, John Ayoade, and Md Ashraf Uddin. “A Zero-Knowledge Proof-Enabled Blockchain-Based Academic Record Verification System”. In: *Sensors* 25.11 (2025), p. 3450.

[95] Tianyu Bai et al. “Health-zkIDM: a healthcare identity system based on fabric blockchain and zero-knowledge proof”. In: *Sensors* 22.20 (2022), p. 7716.

[96] Sara Migliorini et al. “The Blockchain Role in Ethical Data Acquisition and Provisioning”. In: *Proceedings of the 1st International Workshop on Processing Information Ethically co-located with 31st International Conference on Advanced Information Systems Engineering*. Vol. 2417. PIE@CAISE. Rome, Italy: CEUR Workshop Proceedings, 2019. URL: https://ceur-ws.org/Vol-2417/paper5.pdf.

[97] Haris Ahmad and Gagangeet Singh Aujla. “GDPR compliance verification through a user-centric blockchain approach in multi-cloud environment”. In: *Computers and Electrical Engineering* 109 (2023), p. 108747.

[98] Masoud Barati et al. “GDPR compliance verification in internet of things”. In: *IEEE access* 8 (2020), pp. 119697–119709.

[99] Arad Kotzer, Daniel Gandelman, and Ori Rottenstreich. “SoK: Applications of Sketches and Rollups in Blockchain Networks”. In: *IEEE Transactions on Network and Service Management* 21.3 (2024), pp. 3194–3208. DOI: 10.1109/TNSM.2024.3372604.

[100] Akaki Mamageishvili and Edward W. Felten. “Efficient Rollup Batch Posting Strategy on Base Layer”. In: *Financial Cryptography and Data Security. FC 2023 International Workshops*. 2024, pp. 355–366. DOI: 10.1007/978-3-031-48806-1_23. URL: https://doi.org/10.1007/978-3-031-48806-1%5C_23.

[101] Yishay Mansour Yogev Bar-On1. “Optimal Publishing Strategies on a Base Layer”. In: *Financial Cryptography and Data Security. FC 2024*. 2024. URL: https://fc24.ifca.ai/preproceedings/255.pdf.

[102] *EIP-4844 Blobs Documentation*. July 2024. URL: https://ethereum.org/en/developers/docs/data-availability/blockchain-data-storage-strategies/#eip-4844-blobs.

[103] Seongwan Park et al. *Impact of EIP-4844 on Ethereum: Consensus Security, Ethereum Usage, Rollup Transaction Dynamics, and Blob Gas Fee Markets*. 2024. URL: https://arxiv.org/abs/2405.03183.

[104] Carl R Worley and Anthony Skjellum. "Opportunities, challenges, and future extensions for smart-contract design patterns". In: *Business Information Systems Workshops*. 2019, pp. 264–276.

[105] Yibin Xu et al. "Safe design and evolution of smart contracts using dynamic condition response graphs to model generic role-based behaviors". In: *Journal of Software: Evolution and Process* 37.1 (2025), e2730.

[106] Muhammad Bin Saif, Sara Migliorini, and Fausto Spoto. "Blockchain-Based Multirole Authentication and Authorization in Smart Contracts with a Hierarchical Factory Pattern". In: *6th International Conference on Blockchain Computing and Applications (BCCA)*. 2024, pp. 22–29. DOI: 10.1109/BCCA62388.2024.10844391.

[107] Mirko Staderini, Caterina Palli, and Andrea Bondavalli. "Classification of Ethereum Vulnerabilities and their Propagations". In: *2020 Second International Conference on Blockchain Computing and Applications (BCCA)*. 2020, pp. 44–51. DOI: 10.1109/BCCA50787.2020.9274458.

[108] Priyanka Kamboj, Shivang Khare, and Sujata Pal. "User authentication using Blockchain based smart contract in role-based access control". In: *Peer-to-Peer Networking and Applications* 14.5 (2021), pp. 2961–2976.

[109] Andrew Banks et al. *MQTT Version 5.0*. OASIS Standard. OASIS, 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[110] Tarun Kumar Agrawal et al. "Blockchain-based framework for supply chain traceability: A case example of textile and clothing industry". In: *Computers & Industrial Engineering* 154 (2021), p. 107130. ISSN: 0360-8352. DOI: 10.1016/j.cie.2021.107130.

[111] Helena Handschuh. "SHA Family (Secure Hash Algorithm)". In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg. Boston, MA: Springer US, 2005, pp. 565–567. ISBN: 978-0-387-23483-0. DOI: 10.1007/0-387-23483-7_388.

[112] Kenneth Raeburn. *Advanced Encryption Standard (AES) Encryption for Kerberos 5*. RFC 3962. Feb. 2005. DOI: 10.17487/RFC3962.

[113] Shuyi Pu and Jasmine Siu Lee Lam. "The benefits of blockchain for digital certificates: A multiple case study analysis". In: *Technology in Society* 72 (2023), p. 102176.

[114] Untung Rahardja et al. "Socio-economic impact of Blockchain utilization on Digital certificates". In: *Aptisi Transactions on Management* 5.2 (2021), pp. 106–111.

[115] Antonia Stefani and Bill Vassiliadis. "A certification framework for E-commerce digital competencies". In: *Open Journal of Applied Sciences* 13.5 (2023), pp. 758–774.

[116] Georgios Karopoulos et al. “A Survey on Digital Certificates Approaches for the COVID-19 Pandemic”. In: *IEEE Access* 9 (2021), pp. 138003–138025. DOI: 10.1109/ACCESS.2021.3117781.

[117] Shengnan Zhang et al. “Application prospect of blockchain in renewable energy certificates”. In: *Proceedings of the 4th International Conference on Computer Science and Application Engineering*. 2020, pp. 1–5.

[118] Giandari Maulani et al. “Digital certificate authority with blockchain cybersecurity in education”. In: *International Journal of Cyber and IT Service Management* 1.1 (2021), pp. 136–150.

[119] Francesca Fallucchi et al. “Blockchain framework in digital government for the certification of authenticity, timestamping and data property”. In: (2021).

[120] Alexander Rieger et al. “The privacy challenge in the race for digital vaccination certificates”. In: *Med* 2.6 (2021), pp. 633–634.

[121] Nitima Malsa et al. “CERTbchain: a step by step approach towards building a blockchain based distributed application for certificate verification system”. In: *2021 IEEE 6th International Conference on Computing, Communication and Automation (ICCCA)*. IEEE. 2021, pp. 800–806.

[122] Narendra K Dewangan et al. “Enhanced privacy-preserving in student certificate management in blockchain and interplanetary file system”. In: *Multimedia Tools and Applications* 82.8 (2023), pp. 12595–12614.

[123] Untung Rahardja et al. “Immutable ubiquitous digital certificate authentication using blockchain protocol”. In: *Journal of applied research and technology* 19.4 (2021), pp. 308–321.

[124] Li Peng et al. “Privacy preservation in permissionless blockchain: A survey”. In: *Digital Communications and Networks* 7.3 (2021), pp. 295–307.

[125] Rui Zhang, Rui Xue, and Ling Liu. “Security and privacy on blockchain”. In: *ACM Computing Surveys (CSUR)* 52.3 (2019), pp. 1–34.

[126] Arijit Saha et al. “Review on “Blockchain technology based medical healthcare system with privacy issues””. In: *Security and Privacy* 2.5 (2019), e83.

[127] Carlo Mazzocca et al. “A survey on decentralized identifiers and verifiable credentials”. In: *IEEE Communications Surveys & Tutorials* (2025).

[128] Andrea Flamini et al. “On cryptographic mechanisms for the selective disclosure of verifiable credentials”. In: *Journal of Information Security and Applications* 83 (2024), p. 103789.

[129] Clemens Brunner et al. “Did and vc: Untangling decentralized identifiers and verifiable credentials for the web of trust”. In: *Proceedings of the 2020 3rd international conference on blockchain technology and applications*. 2020, pp. 61–66.

[130] Jaqueline Hans, Sajjad Khan, and Davor Svetinovic. “Blockchain CBDC security threats using STRIDE”. In: *2023 Fifth international conference on blockchain computing and applications (BCCA)*. IEEE. 2023, pp. 522–529.

[131] European Parliament and Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council*. https://data.europa.eu/eli/reg/2016/679/oj. Apr. 2016.

[132] Roman Matzutt et al. "Thwarting Unwanted Blockchain Content Insertion". In: *2018 IEEE International Conference on Cloud Engineering (IC2E)*. 2018, pp. 364–370. DOI: 10.1109/IC2E.2018.00070.

[133] Niclas Kannengießer et al. "Challenges and Common Solutions in Smart Contract Development". In: *IEEE Transactions on Software Engineering* 48.11 (2022), pp. 4291–4318. DOI: 10.1109/TSE.2021.3116808.

[134] Neville Grech et al. "MadMax: Surviving out-of-Gas Conditions in Ethereum Smart Contracts". In: *Proc. ACM Program. Lang.* 2.OOPSLA (2018). DOI: 10.1145/3276486.

[135] Vipul Goyal et al. "Attribute-based encryption for fine-grained access control of encrypted data". In: *Proceedings of the 13th ACM conference on Computer and communications security*. 2006, pp. 89–98.

[136] Matt Blaze, Gerrit Bleumer, and Martin Strauss. "Divertible protocols and atomic proxy cryptography". In: *International conference on the theory and applications of cryptographic techniques*. Springer. 1998, pp. 127–144.

[137] Ingo Weber et al. "On Availability for Blockchain-Based Systems". In: *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. 2017, pp. 64–73. DOI: 10.1109/SRDS.2017.15.

[138] Iqra Sadia Rao et al. "Scalability of blockchain: a comprehensive review and future research direction". In: *Cluster Computing* 27.5 (2024), pp. 5547–5570.

[139] Abdurrashid Ibrahim Sanka and Ray C.C. Cheung. "A systematic review of blockchain scalability: Issues, solutions, analysis and future research". In: *Journal of Network and Computer Applications* 195 (2021), p. 103232. DOI: https://doi.org/10.1016/j.jnca.2021.103232.

[140] Di Yang et al. "A Review on Scalability of Blockchain". In: *Proceedings of the 2020 2nd International Conference on Blockchain Technology*. ICBCT '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–6. DOI: 10.1145/3390566.3391665.

[141] Qiheng Zhou et al. "Solutions to Scalability of Blockchain: A Survey". In: *IEEE Access* 8 (2020), pp. 16440–16455. DOI: 10.1109/ACCESS.2020.2967218.

[142] Cosimo Sguanci, Roberto Spatafora, and Andrea Mario Vergani. "Layer 2 blockchain scaling: A survey". In: *arXiv preprint arXiv:2107.10881* (2021).

[143] Chengpeng Huang et al. "Data Availability and Decentralization: New Techniques for zk-Rollups in Layer 2 Blockchain Networks". In: *arXiv preprint arXiv:2403.10828* (2024). URL: https://arxiv.org/abs/2403.10828.

[144] Ertem Nusret Tas et al. "Accountable safety for rollups". In: *arXiv preprint arXiv:2210.15017* (2022). URL: https://arxiv.org/abs/2210.15017.

[145] *ZKsync Era Documentation*. Available at https://docs.zksync.io/build. July 2024.

[146] *Starknet Documentation*. Available at https://docs.starknet.io/. July 2024.

[147] *Aztec Protocol Documentation*. Available at https://docs.aztec.network/. July 2024.

[148] *Scroll Protocol Documentation*. Available at https://scroll.io/blog/blobs-are-here-scrolls-bernoulli-upgrade. July 2024.

[149] *Linea Documentation*. Available at https://docs.linea.build/. July 2024.

[150] *Degate Documentation*. Available at https://docs.degate.com/. July 2024.

[151] TAIKO Labs. "TAIKO: A type-1 Ethereum ZK-rollup". In: Available at https://github.com/code-423n4/2024-03-taiko/blob/main/packages/protocol/docs/taiko-whitepaper.pdf. 2022.

[152] *Arbitrum White Paper*. Available at https://github.com/OffchainLabs/nitro/blob/master/docs/Nitro-whitepaper.pdf. July 2024.

[153] *Optimism Documentation*. Available at https://docs.optimism.io/stack/transactions/fees. July 2024.

[154] *Metis Whitepaper*. Available at https://drive.google.com/file/d/1-hGL4mj8hLtWV8jlt6zRz63yKY14cvyr/view. July 2024.

[155] *Polygon zkEVM Documentation*. Available at https://docs.polygon.technology/zkEVM/. July 2024.

[156] Dani Ni Wang et al. "Loopring: A Decentralized Token Exchange Protocol". In: Available at https://docs-protocol.loopring.io/. 2018.

[157] *Immutable X Documentation*. Available at https://docs.immutable.com/docs/. July 2024.

[158] *Manta Network Documentation*. Available at https://docs.manta.network/docs/Introduction. July 2024.

[159] *ZKFair Documentation*. Available at https://docs.zkfair.io/. July 2024.

[160] Polygon Technology. *Polygon CDK: All About Data Availability*. https://polygon.technology/blog/polygon-cdk-all-about-data-availability. Accessed: 2024-07-15. 2024.

[161] *StarkEx Documentation*. Available at https://docs.starkware.co/starkex/con_data_availability.html. July 2024.

[162] Seongwan Park et al. "Impact of EIP-4844 on Ethereum: Consensus Security, Ethereum Usage, Rollup Transaction Dynamics, and Blob Gas Fee Markets". In: *arXiv preprint arXiv:2405.03183* (2024).

[163] Bernhard K Meister and Henry CW Price. “Gas Fees on the Ethereum Blockchain: From Foundations to Derivatives Valuations”. In: *arXiv preprint arXiv:2406.06524* (2024).

[164] Jiasong Liu. “Digital signature and hash algorithms used in Bitcoin and Ethereum”. In: *Third International Conference on Machine Learning and Computer Application (ICMLCA 2022)*. Vol. 12636. 2023, pp. 1302–1321. URL: https://doi.org/10.1117/12.2675431.