Politecnico di Torino

Industrial Production and Technological Innovation Engineering

AY 2025/2026

Graduation Session March 2026

Design and Implementation of a Transparent and Verifiable Structural Health Monitoring Pipeline Based on Decentralized Technology

Supervisors:

Marco Domaneschi
Valentina Gatteschi
Valentina Villa
Leonardo Zunino

Candidate:

Giorgio Marengo

Acknowledgements

I would like to express my gratitude to my family and all those who have been a source of support and encouragement throughout this journey. It would not have been the same without you.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Civil infrastructure comprises a set of assets essential to modern society. Viaducts, bridges and transport networks ensure the daily movement of people and goods, playing a strategic role in a territory's economic and social fabric. Their proper functioning is therefore imperative. A service interruption, or worse, a structural collapse would have severe safety consequences and trigger cascading disastrous effects. Beyond the potential loss of human lives, the damage could rapidly impact other sectors, causing substantial economic losses and paralysing entire regions. A large number of European infrastructures were designed and built during the economic boom of the 1960s and 1980s, following usage criteria suited to the needs of that time. Today, more than half a century later, these structures are required to bear traffic volumes and commercial loads far exceeding the estimates made at the design stage, inexorably accelerating the processes of wear and deterioration. Historically, structural safety assessments have relied on periodic visual inspections carried out by specialized personnel, both during routine checks and following extreme weather events such as earthquakes, floods, or strong wind. This traditional approach has several limitations: inspection procedures are logistically complex and economically costly. Their effectiveness is severely constrained by the individual inspector's subjectivity and the practical impossibility of detecting invisible or internal material damage before it visibly compromises the structure. The scheduled nature of inspections severely limits the efficacy of this approach, as damage is not identified when it occurs, but only afterwards. To overcome these limitations, Structural Health Monitoring (SHM) systems have been developed. These involve installing a sensor network on the structure to continuously measure key parameters and monitor its condition. By analyzing the collected data, damage can be detected before it becomes visible to the human eye. In conventional SHM pipelines, the infrastructure operator collects vibration data and processes it within private servers or proprietary databases. This model creates a serious information asymmetry, generating a trust problem across the entire architecture. Institutions, citizens and

auditors have no means of independently and objectively verifying that the data provided by the operator is authentic and that the relevant protocols have been followed. This leaves open the possibility of manipulation by the operator to conceal structural damage, thereby avoiding the associated maintenance costs. This work aims to address the limitations of traditional SHM systems by improving overall transparency and enabling a verification mechanism that is public, independent and accessible to anyone. The three key technologies used are: IPFS for distributed storage, TEE for verifiable algorithm execution, and blockchain for Damage Index computation. The synergy between these tools enables the construction of a near tamper-proof system, with all phases tightly bound to one another. The structure of the work is as follows: Chapter 2 examines the central role of infrastructures, highlighting the limitations of traditional inspections and the state of the art of SHM systems. Chapter 3 outlines the key technologies used in the project, explaining why they were selected and their distinctive characteristics. Chapter 4 presents the implementation of the system applied to the practical case study of the Yonghe Bridge, illustrating the configuration of the entire pipeline. Finally, Chapter 5 analyses the results obtained, providing an overview and presenting potential improvements to the system.

Chapter 2

Overview and critical analysis of Structural Health Monitoring systems

One of the main limitations of current structural health monitoring solutions is a lack of transparency and information sharing. Most systems are completely private at every stage, from sensor measurement to decision-making. This organisational structure means that only the head of the facility has a direct view of the processes, thus creating information asymmetry. This reduces the possibility of independent data verification, limits stakeholder confidence, and hinders the reproducibility of scientific results.

The following chapter will examine this topic in greater detail, providing an overview of how SHM systems function, the issues related to data centralisation and the risks associated with this approach.

2.1 Infrastructural context and classification of civil works

Civil infrastructure comprises a wide range of projects with diverse functions, objectives and sizes. These constructions aim to organise a territory according to human needs. They can be divided into two broad categories: linear (or networked), and nodal [1].

The first category is much more extensive and aims to provide a service over a wide territorial area. These infrastructures have a branched and continuous presence, exerting a significant large-scale impact. Examples include highways, railways, oil pipelines, high-voltage power lines, and water supply systems.

Point (or nodal) infrastructures, on the other hand, are built for a specific, limited and clearly defined purpose. These constructions are located in specific areas and perform concentrated functions with localised impacts. Examples include airports, railway stations, power plants and wastewater treatment facilities.

It is important to emphasize that these are not two parallel situations but rather complementary, interdependent systems that integrate with one another. A linear infrastructure comprises multiple nodal infrastructures that ensure its operational continuity and enable its proper functioning. Taking the motorway network as an example, it can be regarded as an extensive system whose purpose is to ensure territorial connectivity. This, however, is made possible only by a set of nodal structures that guarantee its operability at a localized territorial scale. Bridges, viaducts, tunnels, and interchanges can therefore be regarded as smaller-scale works—when compared to a network system—that support the main structure and contribute to its overall organization [2].

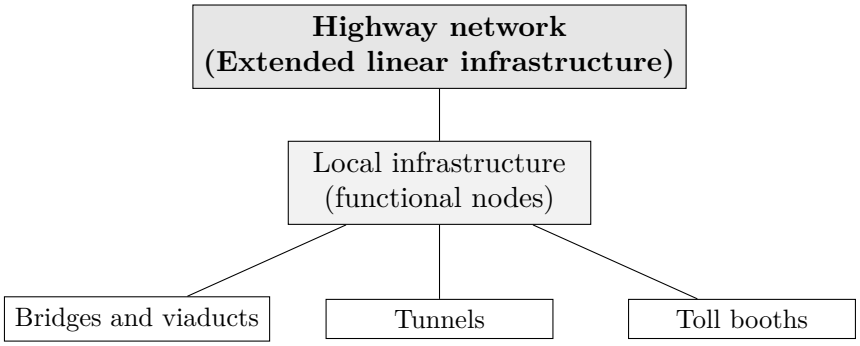

Figure 2.1: Vertical hierarchical representation of a highway network

2.1.1 Critical infrastructures and risk factors

All of the nodal structures contribute to the proper functioning of the primary linear infrastructure, ensuring its operability at the local level. Owing to this strong interdependence, it is essential to closely monitor the condition of these structures to maintain continuity of service. There is no single definition of critical infrastructure (CI); they can be described as systems whose operational disruption could jeopardise, or have direct consequences for, the safety, health, and well-being of a region or a nation.

Country/Institution	Definition
European Union	Critical infrastructure means an asset, a facility, equipment, a network or a system, or a part of an asset, a facility, equipment, a network or a system, which is necessary for the provision of an essential service [3] (Directive 2022/2557, Art. 2).
UNDRR[1]	The physical structures, facilities, networks and other assets which provide services that are essential to the social and economic functioning of a community or society [4].
US	Critical infrastructure comprises the physical and virtual assets and systems so vital to the Nation that their incapacity or destruction would have a debilitating impact on national security, national economic security, or national public health or safety [5] (USA National Security Memorandum on Critical Infrastructure Security and Resilience, 2024).

Table 2.1: Definitions of Critical Infrastructure by major international institutions.

With reference to the example above, bridges, viaducts, and tunnels may be considered CI [6], since their proper functioning is essential to preserve the continuity and efficiency of the whole road network. These components constitute true strategic nodes within the mobility system, as their proper operational condition directly affects safety, traffic flow efficiency, and the territorial economy. In this context, the notion of structural dependence and interdependence [7] assumes particular significance. The first case indicates that one infrastructure is entirely dependent on another, whereas the latter denotes a further condition in which both architectures require each other and mutually influence one another. This phenomenon has the potential to generate cascading effects, since a failure that is localised in a single plant or section has the capacity to spread to the entire network, compromising not only direct functionality but also that of contiguous or complementary elements. For these reasons, nodal infrastructures require a higher level of attention in terms of structural monitoring and preventive maintenance, with the aim of reducing or even eliminating systemic interruptions and improving network resilience.

The collapse of the Polcevera Viaduct, commonly known as the Morandi Bridge, is one of the most emblematic and significant examples of how the loss of functionality of a single nodal infrastructure can generate large-scale systemic effects. The

event, which occurred on 14 August 2018, disrupted a key node in the Italian motorway network, specifically the A10 axis connecting Genoa, the capital of Liguria, to northwestern Italy. In addition to the extremely serious human consequences, the collapse caused road connections to grind to a halt, reducing accessibility to the port of Genoa. This significantly reduced the capacity for freight handling and urban traffic. The national railway system experienced increasing delays and disruptions [8]. The entire economic system was severely affected; the most impacted businesses were freight transport companies, which suffered record losses [9]. This event unequivocally highlighted the vulnerability of critical infrastructures and demonstrated how their proper functioning is essential for other sectors as well, due to the high degree of interdependence that characterizes them [10].

The analysis will then focus on key infrastructure, with a specific focus on bridges and viaducts. These structures are strategic elements of the transport network, whose reliability has a direct impact on safety, continuity of traffic flow, and the overall efficiency of the infrastructure system. This type of facility is of fundamental importance for the motorway network as it connects two areas separated by a natural obstacle (rivers, streams, valleys) or an artificial one (another piece of infrastructure). The durability of bridges is a crucial requirement for the continuity of operation and reliability of the entire network. For these reasons, it is necessary to identify the mechanisms of degradation, monitor the situation, and plan maintenance interventions to ensure safety.

Among the main risk factors, three primary phenomena can be identified as responsible for structural deterioration: material aging, atmospheric effects, and vehicular traffic. The term aging refers to a slow, continuous, and inevitable process that leads to a gradual reduction in the mechanical performance and durability of structural components [11]. The majority of the infrastructure in the EU15 countries[2] was built between the 1960s and 1980s [12], during a period of strong economic expansion, but with construction standards and durability regulations that are now outdated. Aging is a systemic factor as it affects the entire life cycle of structures, progressively reducing their resilience and their ability to maintain performance and withstand the loads estimated in the initial design. Structures nearing the end of their design life are also more susceptible to degradation due to factors such as traffic and external agents.

[2]EU15 includes Austria, Belgium, Denmark, Finland, France, Germany, Greece, Ireland, Italy, Luxembourg, the Netherlands, Portugal, Spain, Sweden, and for this pre 2020 study also the United Kingdom.

2.2 Purposes and functions of SHM systems

As discussed in the previous section, critical infrastructures require a higher level of attention compared to ordinary structures. Their service interruption can cause not only localized damage but also large-scale disruptions, affecting the entire infrastructure network. Moreover, due to the high degree of interconnectivity, other networks may also experience a loss of functionality. In addition to safety concerns, the inefficiency of a nodal infrastructure can lead to significant economic consequences for the region, also impacting the local social fabric [13]. These considerations highlight the importance of a management system based on prevention and continuous monitoring, capable of ensuring the resilience and operability of infrastructure over time. In this context, Structural Health Monitoring systems represent the most effective solution for issues related to the supervision and maintenance of civil works, introducing a proactive approach that integrates sensors, data processing, and human expertise. This section will illustrate the functional principles of this solution, its evolution over time and the main benefits of adopting an SHM system.

2.2.1 From inspection-based control to intelligent monitoring

Visual inspection is the oldest and most commonly used method for assessing the structural condition of a bridge. These inspections are carried out at variable intervals, with higher frequency for critical infrastructures or following specific events such as extreme weather conditions or vehicle impacts, in order to evaluate their consequences [14]. The main purpose is to identify visible signs of deterioration, corrosion, and deformation, and to document such evidence in order to plan future safety measures and targeted interventions aimed at restoring the mechanical and structural capacity of the structure. The inspection is conducted with the aid of instruments such as binoculars, video cameras, and drones, which allow the current condition to be documented and archived. The final report is prepared using specific checklists designed to formalize and standardize the safety assessment process and to provide an immediate overview of the structure's condition [15]. On-site evaluation is still one of the most widely used and important practices for monitoring structures, but it has several limitations and disadvantages that greatly reduce its effectiveness and the possibility of relying solely on it. The main obstacle is accessibility. Certain parts of bridges, such as underwater foundations or the lower sections, are extremely difficult to inspect without the use of specialized equipment and machinery. The most common vehicle used for this type of examination is the Under Bridge Inspection Truck (UBIT), a truck equipped with a movable mechanical arm and an inspection basket. UBIT inspections allow access to and

observation of even the most concealed structural areas, but they involve higher operational costs and can cause service disruptions along the network. Moreover, due to the vehicle's considerable size and weight, its presence adds a significant load to the structure, often requiring partial or complete traffic closure. Finally, UBIT inspections demand the involvement of specialized personnel, further increasing the overall costs [16]. Another key factor to consider is the subjectivity of the inspector conducting the examination. The inspector's skills and experience are crucial for accurately assessing a bridge's condition, and their personal judgement significantly influences the outcome of the evaluation [17]. In conclusion, it can be said that visual inspections are the most widely used and popular method of checking and verifying the condition of infrastructure. They allow for immediate, simple, and relatively low initial cost. However, their limitations are equally evident. A detailed and comprehensive examination would significantly extend inspection times and substantially increase costs due to the need for specialized equipment, qualified personnel, and partial traffic closures. The main drawback of this monitoring approach lies in the periodic nature of the inspections, which are typically scheduled at regular intervals or following specific events. This means that action is taken only after damage has occurred or the first warning signs have appeared. In essence, it represents a reactive form of control, one that responds to specific events rather than preventing them, thus overlooking hidden or non-visible risks and compromising preventive maintenance.

2.2.2 Structure and operating logic of SHM systems

The limitations of traditional inspection systems have highlighted the need for more reliable, continuous solutions to address the gaps of periodic checks. It was therefore essential to introduce an approach capable of directly and continuously measuring structural parameters, thereby overcoming reliance on sporadic assessments and subjective operator interpretation.

To meet these requirements, Structural Health Monitoring (SHM) systems were developed as architectures that periodically acquire data from sensors installed on the infrastructure and use it to perform measurements at regular intervals. The objective of these systems is to supplant traditional inspection methods and provide continuous, transparent control grounded in quantitative evidence, capable of detecting damage or potential anomalies promptly, before they evolve into critical conditions [18].

The advent of SHM has transformed the management and maintenance of civil structures. Conventional approaches involve interventions carried out retrospectively, only after visible damage has occurred; this reactive stance inherently limits preventive capacity. By contrast, SHM systems follow a proactive philosophy based on prevention and continuous monitoring. Sensors and automated data collection

enable detection and real-time analysis of even the smallest variations in structural behaviour, allowing prompt responses and predictive maintenance planning. This approach reduces dependence on human judgement and renders the inspection process more objective, standardised, and traceable over time. The evolution of structural monitoring systems is indicative of the technological developments that have taken place in recent decades. The earliest applications, dating back to the 1970s, utilised wired sensors for the measurement of vibrations in bridges and aeronautical structures, with constrained data acquisition and manual data storage. The advent of modal identification techniques and enhanced computing capabilities in the 1990s engendered the establishment of inaugural permanent monitoring programmes on intricate infrastructures. Since the 2000s, the proliferation of wireless sensors, IoT networks, and machine learning-based analysis techniques has facilitated distributed and intelligent data management, thereby establishing the foundation for predictive systems capable of correlating structural signals with degradation phenomena over time. Common approaches to damage detection describe SHM systems as processes divided into three main phases: data acquisition and transmission, data processing, and data interpretation. Together, these processes form the foundation for developing maintenance and safety strategies for a structure [19]. The acquisition phase involves the measurement, collection, and recording of physical signals obtained from the network of sensors installed on the infrastructure. It represents the starting point of the entire architecture, as well as one of its most critical stages, since any disturbances or data losses could compromise the whole process. The most commonly used sensors for measuring mechanical quantities are accelerometers and strain gauges. The data is then transmitted via wireless solutions and used as input for algorithms developed for its processing. This phase includes noise filtering, normalization, time synchronization, signal decomposition, and the extraction of damage-sensitive features. Among the most widely used signal processing strategies are decomposition techniques, time-frequency analysis, modal analysis, and statistical and clustering methods [20]. The final phase of an SHM system is the interpretation of the results obtained from the previous step. Its purpose is to detect, locate, and quantify potential damage, as well as to develop appropriate monitoring strategies. This phase translates the acquired signals into actionable decisions, determining whether intervention is required, further measurements should be carried out, or the structure can be deemed safe.

2.2.3 Benefits of data-driven solutions

The adoption of an SHM system offers multiple advantages from different perspectives. It enables inspections and assessments to be performed only when there is a genuine need, rather than through cyclical and scheduled interventions, thereby

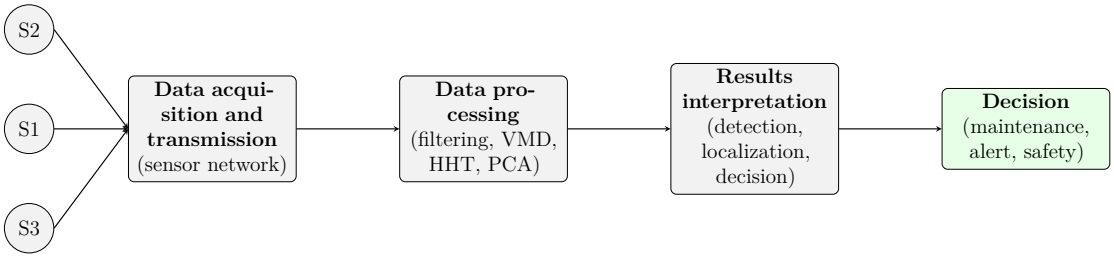

Figure 2.2: Schematic representation of the SHM pipeline

prioritizing urgent and critical maintenance activities. A clear example can be found in the guidelines for monitoring existing bridges issued by the Italian Ministry of Infrastructure and Transport. The report explicitly states that the maximum time interval between two extraordinary inspections should not exceed five years; however, when an instrumental monitoring system is employed, this frequency can be reduced [21]. Fewer interventions also mean reduced disruption and service interruptions for maintenance purposes and, consequently, lower costs. Another significant advantage is the extension of the infrastructure's average service life. By acting preventively, before damage occurs, it is possible to preserve the structure in optimal condition and prolong its operational period. The principal benefit, however, is safety. A real-time monitoring system makes the structure's health status continuously available, enabling prompt identification of repair needs. This allows the situation to be kept under constant control and supports the planning of targeted, rapid interventions. Damage is therefore detected immediately, rather than only during the next scheduled visual inspection.

2.3 Limitations of current solutions

As highlighted in the previous paragraphs, the adoption of an SHM system can play a crucial role in infrastructure management. The advantages are numerous, among the most significant are the reduction in costs resulting from fewer visual inspections and the consequent overall improvement in structural safety. However, it is important to emphasize that these architectures are highly complex and require proper implementation and interpretation in order to fully exploit their potential.

As reported in this study [22] one of the main challenges is to underestimate or not fully understand the difficulty of implementing an SHM system. In fact, a multidisciplinary team is needed to work together in a coordinated approach to design, install, and effectively use the monitoring network. Several professional figures must interact with one another throughout the different phases of the project in order to achieve the desired outcome. Infrastructure owners and managers, engineering consultants, national political leaders and construction engineers are

just a few examples of the professionals involved in such projects. It is also essential to perform periodic checks on the condition of the sensors, as they are highly sensitive to seasonal variations, temperature changes, and environmental factors [23]. If these influences are not properly accounted for, they may affect the analysis and lead to inaccurate measurements or, in the worst cases, false damage detections.

One of the most critical issues with current solutions is the lack of transparency and verifiability throughout the entire monitoring pipeline. It is, in fact, extremely difficult to precisely verify the correct execution of the overall process due to the opacity of existing systems. Each stage of an SHM system is fully managed and controlled by the contracting company, which owns both the hardware and software components. This situation makes it challenging to ensure the proper functioning of the entire system, as no audit trail processes exist to guarantee the correct execution of the architecture. Moreover, datasets and algorithms are often protected by intellectual property rights, making them inaccessible and non-reproducible by third parties. As a result, it becomes impossible to understand the methods and techniques used during the data processing and interpretation phases. Additionally, there is no certainty regarding the authenticity of the measurements or their correct use, leaving room for potential manipulation.

For this reason, several studies have been conducted with solutions integrating decentralized technologies and including verifiable proof of correct execution of the entire pipeline. Some of the most innovative approaches include the implementation of Blockchain, IPFS, and Digital Twin [24].

2.4 Research objectives

The objective of this research is the design and development of an alternative solution capable of reducing the reliance on trust within structural health monitoring systems. The proposed architecture aims to minimize the opaque stages present in current data processing pipelines, maximizing transparency, traceability, and verifiability at every step of the process. To this end, the research introduces and employs decentralized technologies whose operation is based on the immutability and integrity of results. The ultimate goal is to create an end-to-end workflow in which every operation can be reconstructed and, most importantly, verified. The study will also highlight the feasibility and current limitations of this approach, providing comparisons among different alternative solutions and emphasizing their respective strengths and weaknesses.

From a broader perspective, this research seeks to contribute to the transition toward more transparent and reliable SHM systems, reducing the risks of data manipulation and increasing stakeholder confidence in digital monitoring processes.

Chapter 3

Theoretical foundations of decentralized systems for verifiable data management

SHM systems, as discussed in the previous sections, represent complex architectures that continuously measure, analyze, and process vast amounts of data on a daily basis. Their accuracy and reliability are essential to ensure the effectiveness of damage detection and prevention pipelines.

The traditional approach relies on storing sensor measurements within proprietary, centralized servers, which increases the risk of data manipulation and, more critically, potential data loss. Relying on a single storage point implies complete dependence on the proper functioning of that system. In the event of temporary service interruptions or, in the worst case, permanent failure the data may be irreversibly lost, compromising the continuity and trustworthiness of the entire monitoring process.

Decentralized solutions have emerged as a response to the limitations described above, offering a distributed storage system across multiple trusted nodes within a network. In such architectures, information is replicated and validated by independent participants who collectively ensure its availability, authenticity, and security through cryptographic verification.

This chapter provides the theoretical foundations of the most mature decentralized technologies that will later be integrated into the development of a transparent SHM pipeline. Specifically, it presents the operating principles of the InterPlanetary File System (IPFS), verifiable computation frameworks based on Trusted Execution Environments (TEEs) and Zero-Knowledge Proofs (ZKPs), and blockchain architectures supported by smart contracts and oracles. Together, these technologies constitute the conceptual basis for achieving verifiable, tamper-resistant, and

transparent data management in structural health monitoring applications.

3.1 The concept of decentralization

Decentralized systems have been conceived and developed with the aim of overcoming the limitations imposed by centralized architectures. In traditional models, data control, validation, and storage are entrusted to a single entity that defines the management procedures, access policies, and update schedules. In such structures, the entire trust and ownership of the information are concentrated in a single server or, in larger systems, within a network of proprietary databases. This organization makes the availability and usability of data entirely dependent on the status and reliability of the node that hosts it.

A malfunction, service interruption, or cyberattack can compromise data accessibility, rendering information temporarily or even permanently unavailable. Moreover, the proprietary nature of the servers prevents external entities, such as regulatory bodies or independent verifiers, from accessing the data or inspecting their authenticity, since no public verification mechanisms exist. This condition creates a twofold vulnerability: on one hand, the end user is entirely dependent on the correct operation of the central system; on the other, supervisory or auditing entities lack the means to certify the accuracy or integrity of the stored information [25]. In this context, decentralization emerges as a concrete response, capable of distributing trust and responsibility among multiple independent nodes, thereby reducing vulnerability and enhancing the overall transparency of the system.

To mitigate the risks associated with data loss resulting from server failures or malfunctions, distributed systems have been introduced [26]. These systems consist of multiple physically separated components, often located in different geographical regions, that work together to achieve a common goal. Although these components operate with a degree of autonomy, they coordinate through consensus, replication, leader-election, and orchestration mechanisms, while being managed under unified operational policies.

This approach is common among major global providers of digital information services. Companies such as Amazon Web Services (AWS), Google Cloud, and Meta operate vast infrastructures based on distributed databases and servers deployed worldwide. Coordination relies on sophisticated replication and synchronization protocols, as well as on control-plane services that are typically redundant and globally managed, rather than dependent on a single physical control node.

This architectural design enhances the system's overall resilience, ensuring that a local failure does not compromise the entire network, but rather affects only the portion managed by the affected node. The pervasive deployment of servers has been demonstrated to enhance system performance by reducing latency and, most

significantly, by ensuring a balanced distribution of workload across active nodes, thereby facilitating more uniform response times. The implemented solutions have been shown to engender enhancements in terms of both performance and reliability. Nevertheless, while these architectures significantly reduce single points of technical failure, they remain under the governance of centralized corporate entities, which retain ultimate control over policies, access, and data management. In such instances, data remains inaccessible to the public, thus allowing for potential manipulation without the presence of any verifiable evidence. The absence of an autonomous control or verification mechanism leaves the issue of trust unresolved, rendering the system opaque and vulnerable to undetectable alterations. Decentralization emerged as a response to this situation, proposing a physical, logical, and decision-making approach to network management, eliminating the need for central control.

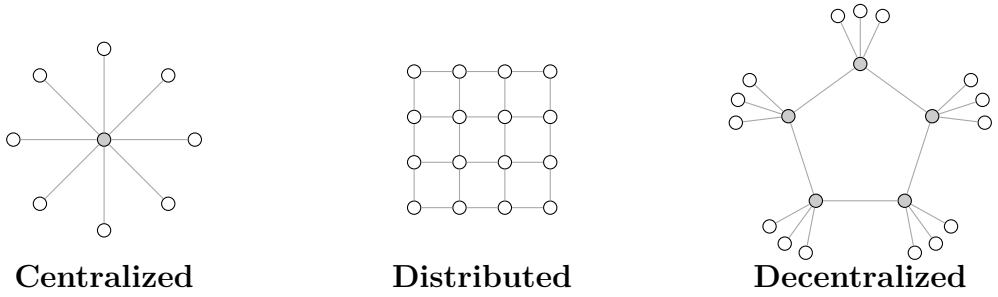

Figure 3.1: Conceptual comparison between centralized, distributed, and decentralized network architectures.

In a decentralized architecture, each node participates in the maintenance and validation of the system with equal rights and responsibilities, without fixed hierarchies or permanent central authorities. Every operation must be confirmed and approved through consensus protocols, which ensure agreement among nodes and prevent malicious actions or attempts to manipulate data. These mechanisms enable the network to achieve a consistent and shared system state even in the presence of untrusted or unreliable nodes, thus removing the need for any pre-existing trust among participants. Among the main consensus protocols are the Byzantine Fault Tolerance [27], Proof of Work (PoW) and Proof of Stake (PoS) [28] each offering different trade-offs between security, performance, and energy efficiency.

Network	Description
Centralized	Every aspect of control, data validation, and archiving is managed by a central node. Failures and attacks can compromise the entire operation and integrity of the system.
Distributed	The network consists of various geographically distributed nodes that share resources, and work together to achieve a common goal. This solution increases resilience and load balancing. Command and coordination protocols may be present. A local error does not compromise the operation of the architecture.
Decentralized	Control and validation are shared among multiple independent entities with equal authority. Consensus mechanisms ensure data integrity and trust without a central authority, making the system transparent, verifiable, and resistant to manipulation.

Table 3.1: Conceptual comparison between centralized, distributed, and decentralized networks.

The adoption of decentralised architectures has been shown to significantly mitigate the critical issues typically associated with both centralized and distributed solutions. The advantages can be attributed to the nature of the entire system, its nodal independence, and the methods used to validate each decision. Specifically, four primary dimensions can be analysed:

1. **Transparency and traceability**: every operation can be independently verified by any participant, ensuring the transparency of the system and the immutability of its data. The complete history of each process remains accessible, allowing the reconstruction of the sequence of transactions, validations, and operations.

2. **Data Integrity and Tamper Resistance**: The validation of data necessitates the attainment of approval through distributed consensus mechanisms. Subsequent to the confirmation of the operation, the mechanisms ensure its permanence. Any attempt to modify the results is promptly detected by the network, which does not validate the execution. This significantly increases the reliability and accuracy of the results.

3. **Reliability and operational continuity**: the management and functionality of the system are ensured by the multiple independent nodes that compose

the network. A local failure does not cause any serious issue to the overall architecture which, thanks to the distribution of its components, remains active. These features maximise reliability, resilience, and operational continuity, making widespread service disruptions or system-wide downtimes extremely unlikely.

4. **Trust independence**: No entity requires or needs implicit trust. Verification is based on mathematical proofs and cryptographic guarantees.

After having analyzed the importance and advantages of decentralized systems, it is necessary to understand how they can be implemented in a real pipeline. The following sections will introduce and explain how and which technologies can be used to create a transparent and verifiable architecture. A solution will be proposed for each main phase of an SHM system, namely data acquisition and transmission, data processing, and results interpretation.

3.2 Transparent and Distributed Data Storage

As previously mentioned, the acquisition, management and storage of data represent the first step in any monitoring system. Each infrastructure is equipped with a network of sensors, meticulously arranged to collect information on various structural and environmental parameters. The recorded values are subject to variation depending on the position of the sensors, their exposure to atmospheric agents, and the intensity of dynamic loads. These factors can result in significant deviations from standard conditions.

In Structural Health Monitoring (SHM) contexts, measurements constitute the raw data on which all subsequent analyses are based. It is therefore essential that these values are stored and preserved securely, in order to prevent data loss and to ensure their authenticity and integrity. Any alteration or absence of data would compromise the entire damage identification system at its core.

3.2.1 InterPlanetary File System

The InterPlanetary File System (IPFS) is a decentralized data storage and sharing system introduced in 2014 through this white paper [29]. It is an open-source set of protocols that combines several concepts already known in peer-to-peer (P2P) architectures. IPFS introduces a new content-based retrieval approach as opposed to the traditional location-based one. In conventional solutions, retrieval is location-dependent: each resource is identified and accessed based on where it is stored. On the web, searches are performed through the Uniform Resource Locator (URL), which defines the addressing scheme. The process is straightforward: a

client connects to a server via the HTTP/HTTPS protocol [30] and requests the desired content. The server processes the request and returns the corresponding data. The hierarchical nature of this model is evident and non-symmetrical, as the client can only issue requests, while the server, occupying a higher position, provides the service. The central authority holds all the information, logic, and decision-making authority and chooses whether, how, and when to provide what is requested.

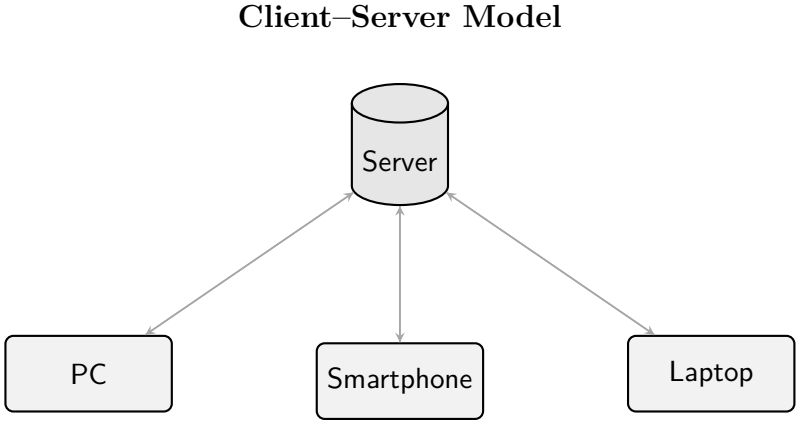

Figure 3.2: Representation of the Client–Server model.

This location-based structure allows information to be managed in a simple and efficient manner, since a single point holds control over the contents, facilitating their updating and the definition of new data access policies. However, the limitations of this model are equally evident. As in any centralized system, its functioning depends on one or a limited number of servers. In the event of a failure or malfunction, all hosted information would become unavailable, resulting in issues of resilience and service continuity. Moreover, a file can be modified, replaced, or removed while maintaining the same URL. This means that, when querying the same location, there is no guarantee of retrieving the same content. For the same reason, no reliable history of the information is ensured, as data can be altered by the administrator without leaving any clear trace.

Finally, under a high number of simultaneous client requests, a bottleneck may occur, resulting in high latency due to excessive incoming traffic. For these reasons, a content-based retrieval system has been introduced, implemented through Content Identifiers (CID). These are unique fingerprints designed to associate the file and its content with a well-defined string of characters. Two identical files will therefore have the same CID code, but even a single comma difference will result in the creation of a completely different identifier. Each piece of content uploaded to the IPFS network is divided into 256 kB blocks, and each block is assigned a

cryptographic hash (typically SHA-256). This process is known as chunking. The values are then organized into a data structure called a Merkle Directed Acyclic Graph (Merkle DAG). This tree contains all the identifiers of the file portions. The hashes of the individual blocks are combined recursively until a single root hash is obtained that represents the entire content: the CID.

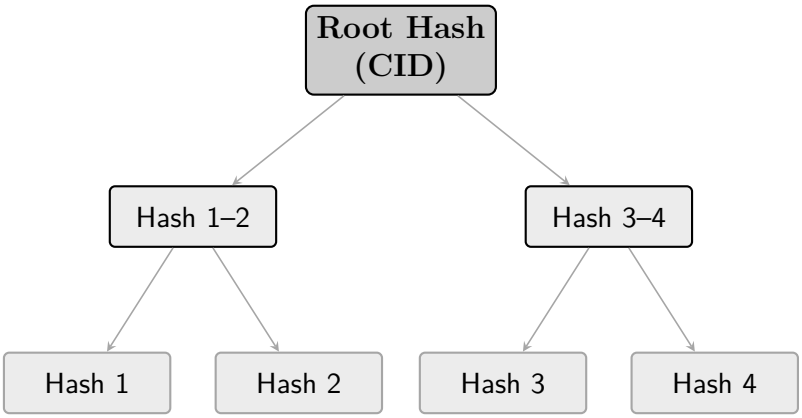

Figure 3.3: Merkle Tree structure showing the cryptographic hash hierarchy.

This file organization has numerous advantages and ensures better efficiency of the entire network. Dividing documents into smaller portions allows certain parts that are common to multiple files to be reused. If, at the local level, a first file consisting of ten blocks is uploaded, followed by a second file consisting of eight blocks (five of which are identical to those already stored), the system will only save thirteen in total. IPFS identifies the content and is able to reuse the sequences already present in the node, avoiding re-archiving them. This allows for better optimization of storage space and avoids redundancy at the local level. Chunking also allows for parallel downloading of information from multiple different nodes that share the same content or portions of it, thus improving the speed, reliability, and resilience of the system. Searches on the IPFS network are performed via CID, without knowing where it is located or which node in the network owns it. To retrieve this information, Distributed Hash Tables (DHT) are used, which are registers that associate certain content with its owner. Each CID is therefore associated with at least one peerID, which is the hash of the public key of the node that holds the information. There is no single DHT registry, but each node stores a small portion of it. The search is based on Kademlia [31], a protocol for organizing and finding data in a peer-to-peer network quickly and without a central server. After obtaining the list of nodes that hold the file, it is necessary to communicate with some of them to obtain the necessary information. This is handled by Bitswap [32], the exchange protocol used within the network. Each node maintains two distinct lists: the Wantlist, containing the blocks it is requesting, and the Have list,

containing the blocks it possesses and can provide to others. The requesting node transmits its Wantlist to selected peers, which compare the requested blocks with those in their Have list and, if available, proceed to send them. Each peer keeps track of how much it has given and received from each other peer, creating a sort of balance sheet. Nodes that request without giving will have a worse reputation and other peers will be less inclined to share data with them. Otherwise, collaboration is incentivized with creditor nodes.

This system was designed to avoid parasitic peers in the network, ensuring proper contribution from all actors and encouraging cooperation. It is important to emphasize that a file, once uploaded to IPFS, does not remain permanently available by default. Locally cached data or previously retrieved blocks are periodically removed through a process known as garbage collection [33]. The purpose of this mechanism is to purge data that is no longer referenced or pinned, thereby freeing storage space for new content. To ensure the long-term preservation of a file, it must be pinned, meaning that the node explicitly specifies that the corresponding blocks should not be removed during data cleanup operations. In the event of a node being offline, even when the documents are pinned, access will not be possible. It is imperative to emphasise the necessity of ensuring that other network entities utilise pinning mechanisms to secure the file, or alternatively, employ permanent pinning systems. The primary benefits of an IPFS-based storage architecture are as follows:

1. **Decentralization**: there is no central server coordinating and managing operations; all nodes have the same hierarchical level and can perform both client and server functions. Blackouts or local service interruptions do not compromise data availability.

2. **Deduplication**: through chunking, documents are broken down into smaller portions. If two files are identical or share identical portions, IPFS will only store one copy of the shared chunks, reducing the storage space required.

3. **Data integrity**: The Content Identifier (CID) guarantees that there have been no changes or alterations and that the version is the original one.

4. **Resilience**: sharing and pinning files across multiple nodes ensures file redundancy across the network. Even if some nodes are disconnected, accessibility is guaranteed.

5. **Cost**: the service is completely free of charge, as each node retains part of the storage.

6. **Permanence**: once a file is uploaded, it cannot be removed as long as it remains stored on at least one node, since no single entity has greater authority

or control over the network. Copies are distributed across multiple peers, and the local deletion of the file on one node does not compromise its overall availability.

In conclusion, it can be posited that an architecture such as IPFS solves many of the problems associated with the centralisation of information storage, providing an open source, peer-to-peer, and free service. The protocols on which it is based guarantee maximum transparency of processes, allowing for secure, distributed, immutable, and above all transparent storage.

3.3 Verifiable Computation

There are numerous scenarios where proving the correct execution of a program is of fundamental importance. In today's increasingly digitized and interconnected world, we rely daily on thousands of software applications and programs. These systems not only manage complex operations at the corporate and industrial levels but also provide critical information essential for the work, communication, and daily lives of billions of people. From financial data management to transport control systems, and from medical devices to communication platforms, software reliability and correctness have become indispensable requirements. A single error or malfunction can lead to significant consequences, ranging from data loss and reduced productivity to serious risks for safety and personal security. For this reason, formal verification and rigorous software testing represent crucial phases in modern software development.

Ensuring computational correctness is therefore a priority, but it presents some challenges. It is not possible for a subject with limited hardware resources to replicate the execution of the program, thus preventing verification. Another key limitation is that of intellectual property. Companies and institutions invest significant financial resources in the development of sophisticated, proprietary algorithms that are strictly confidential and must not be disclosed to third parties or potential competitors.

To address these challenges, a set of methods and processes have been implemented to verify the execution of the program, ensuring its proper functioning and delivering accurate results. These measures effectively eliminate the limitations previously identified. The term verifiable computation (VC) refers to the possibility of providing a verifier with limited computing power compared to that required to perform the computation with proof that a function F applied to inputs i has correctly produced a result F(i) [34]. This solution enables a third party to verify the correct execution of a program without having to re-execute it completely. This allows a greater number of people to perform independent, simple checks without the need for specific or highly advanced hardware.

In this manner, the prover (the individual seeking to demonstrate the correctness of the execution) can share with the verifier (the person responsible for the verification process) a concise proof that contains only the information strictly necessary for the validation process. This approach ensures that the input and/or output values and source code remain private.

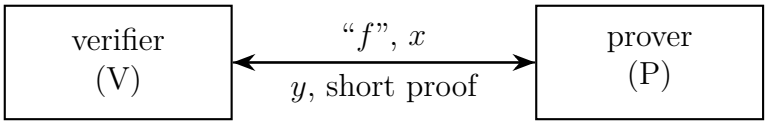

Figure 3.4: Interaction scheme between verifier and prover in verifiable computation

In the following sections of this thesis, two alternative methods commonly used to ensure the correct execution of a program will be introduced and analysed. It is important to note that these two approaches are entirely distinct, and as such, they offer divergent guarantees and trust models. The presentation will explore their unique characteristics, practical applications, and associated limitations.

3.3.1 Trusted Execution Environment

During program execution, data must be loaded into memory in a readable format to be processed by the hardware components, primarily the CPU. This state, known as "data in use", represents a critical vulnerability point. The operating system, or an administrator with high-level privileges, could potentially access this information, altering its veracity and compromising security. Such exposure creates a significant risk to the correct execution of the program. While traditional cryptographic techniques effectively protect data at rest (via encryption) and data in transit (via protocols like TLS/HTTPS), they leave data exposed during runtime processing.

To address this challenge, Trusted Execution Environments (TEEs) have been introduced. TEEs are isolated execution environments implemented directly within the processor hardware, designed to guarantee the confidentiality and integrity of code and data during computation. Unlike software-based solutions, where security relies on the trustworthiness of the operating system or hypervisor, a TEE leverages hardware-enforced mechanisms to create a secure enclave completely isolated from the rest of the system, inaccessible even to users with the highest administrative privileges. Several implementations and applications of this technology exist, each tailored to address different threat models. Based on the level of data isolation and the scope of the trusted boundary, TEEs can be primarily classified into two categories:

1. **Process-Based Isolation**: In this approach, isolation is enforced at the application or sub-application level. The partitioned memory regions designated to store confidential information are referred to as enclaves, which provide strong isolation from the rest of the system. Memory partitioning therefore creates two parallel execution domains: an exposed, "untrusted" environment susceptible to external attacks, and a protected, "trusted" environment where external modification is prevented.

   Importantly, the CPU treats all data outside the enclave as untrusted by default. As a result, access to the protected region is enforced at the hardware level: neither the kernel, nor the hypervisor, nor any other process can read from or write to the enclave's memory space.

   A comprehensive definition of this technology states that a "Trusted Execution Environment (TEE) is a tamper-resistant processing environment that runs on a separation kernel. It guarantees the authenticity of the executed code, the integrity of the runtime states (e.g., CPU registers, memory, and sensitive I/O), and the confidentiality of its code, data, and runtime states stored on persistent memory" [35].

   This solution is designed to ensure the secure execution of specific software and does not guarantee the isolation of the entire machine, but only of certain processes. The primary benefit of this approach is that it involves a very small Trusted Computing Base (TCB), meaning that trusted components are kept to a minimum and, as a result, the attack surface is also minimised. Even in the event of a server operating system compromise, program execution is considered secure thanks to the level of isolation from the rest of the machine. This solution provides the highest level of granularity as it is able to partition even a single piece of code.

   However, the primary challenge inherent to this approach is the complexity of development. Applications require explicit partitioning to delineate which components must be isolated within the enclave and which remain in the untrusted domain. Furthermore, transitions between trusted and untrusted contexts (via ECALLs and OCALLs) incur significant computational overhead. This context switching increases CPU utilization, thereby negatively impacting both energy efficiency and execution latency. Additionally, early implementations imposed strict constraints on the size of the protected memory, specifically the Enclave Page Cache (EPC), which severely limited the scalability of the solution. However, subsequent architectural enhancements, such as Intel SGX2 and its support for Dynamic Memory Management, have substantially mitigated these constraints. The most prominent implementations of TEE technology include Intel SGX and ARM TrustZone.

2. **VM-Based Isolation**: This represents the most recent and deployment-friendly approach, designed to lower the barrier to TEE adoption while improving scalability. Rather than isolating individual processes, this model encrypts the entire memory region allocated to a virtual machine using dedicated hardware components within the processor. Each time the VM boots, the CPU automatically generates a fresh AES (Advanced Encryption Standard) key, which is never written to main memory; instead, it is stored in a processor-internal region inaccessible to any external component. Consequently, all data is encrypted before reaching RAM, ensuring that even an attacker with direct physical access to the hardware cannot extract meaningful information from the memory bus or modules. The reverse operation occurs on read: memory returns the encrypted value to the CPU, which decrypts and processes it internally. The same key material is additionally applied to protect CPU registers, allowing the hypervisor to operate normally while keeping register contents opaque to any external observer.

   In this model, the hypervisor retains full control over memory allocation throughout the execution process, yet remains unable to access the stored values.

   The primary advantage of this approach is its ease of deployment and the near-complete removal of the technical barriers typically associated with TEE adoption. No configuration changes or software modifications are required to run existing applications correctly, making the technology accessible to a significantly broader range of users. Moreover, since the entire system operates within an isolated environment, there is no need to transition between trusted and untrusted execution contexts—as is the case with process-based isolation—which substantially reduces the associated computational overhead. All system calls (syscalls) are handled within the encrypted domain, yielding performance characteristics comparable to those of a conventional virtual machine. This model also imposes no restrictions on accessible memory size, effectively eliminating the EPC limitations inherent to process-based solutions.

   The key concern with VM-based solutions lies in the broad scope of trust they entail: by encrypting the entire virtual machine, the guest operating system is inherently subsumed into the trusted domain, substantially expanding the Trusted Computing Base (TCB). This may significantly widen the attack surface, since any vulnerability present in the kernel or system services could compromise data security from within the VM itself. Additionally, this approach offers considerably less granularity than process-based isolation, as it provides no mechanism to isolate individual applications from the broader software environment.

   Another significant limitation is the dependence on cloud providers and the

associated costs. Deploying Confidential VMs requires dedicated hardware and hypervisor-level configurations managed by the provider, which frequently results in vendor lock-in and makes migrating the infrastructure to alternative platforms considerably more difficult. Additionally, these instances tend to carry higher operational costs compared to standard virtual machines or process-based solutions, owing to the specialised hardware requirements and the reduced flexibility in resource selection. Some of the solutions available on the market include Intel TDX and AMD SEV (in the basic, ES, and SNP versions).

A less common variant of TEE technology enables isolation at the function or library level, offering the finest degree of granularity currently available. This allows specific subroutines or modules within an application to be independently protected. It is, however, a highly specialised approach suited to equally narrow use cases, and can broadly be considered an extension of the process-based isolation model.

The analysis demonstrates how TEE technologies, despite their different architectural approaches (process-based vs. VM-based), offer an effective solution for protecting data confidentiality and integrity during execution (data-in-use). Yet isolated execution alone is not sufficient to establish trust in distributed or decentralised scenarios. A remote party (the Verifier), with no physical access to the machine, faces a fundamental problem: how can they be certain that the server is running a genuine TEE rather than a software emulator? And above all, how can they verify that the code executing inside the enclave is exactly what was intended, and not a malicious or tampered version? In the absence of a cryptographic proof mechanism, hardware isolation remains a "black box" that must be trusted blindly. To resolve this issue, TEEs use a key process called Remote Attestation. This is a cryptographic mechanism that allows the hardware to generate a digital proof (called a Remote Quote) signed directly by the processor. This proof unambiguously certifies two properties:

1. The authenticity of the hardware platform, namely the CPU, the firmware update level, and the integrity of the TEE configuration

2. The hash of the software, or the relevant portion of it, executing within the isolated environment. This value is known as MRENCLAVE and is both unique and deterministic: even the smallest modification to the code will produce an entirely different hash.

It is important to emphasise a fundamental distinction from pure cryptographic proofs (such as Zero-Knowledge Proofs). A TEE does not provide mathematical proof of the calculation's correctness; rather, it provides proof of trust in the

Table 3.2: Technical comparison between Process-based and VM-based TEE architectures.

Feature	**Process-Based (e.g., Intel SGX)**	**VM-Based (e.g., Intel TDX, AMD SEV)**
Isolation Granularity	Application code / Single process	Full Virtual Machine (OS + Apps)
Trusted Computing Base (TCB)	**Minimal**: CPU + App Code only. (OS is untrusted)	**Large**: CPU + Guest OS Kernel + Firmware + Apps
Software Compatibility	**Low**. Requires recompilation or LibOS. No direct syscalls.	**High**. "Lift and shift" for legacy apps. OS runs unmodified.
Attestation Type	Precise measurement of code/data identity (MRENCLAVE).	Coarse-grained. Measures initial boot state (firmware/kernel).
Memory Limit	**Limited** by EPC size (expensive paging if exceeded).	**No practical limit** (uses main RAM encrypted).
Protection from Root	**Yes**. Host Root cannot access enclave memory.	**No** (Internal Root). Root inside VM sees data. Host Root is blocked.
Performance Overhead	**High for I/O** intensive tasks (Context switching).	**Low** (2–5%). Hardware memory encryption overhead only.

hardware. It guarantees that a specific piece of software (identified by the MRENCLAVE) has been executed in a secure environment and has not been tampered with. The reliability of the output is therefore contingent on the trustworthiness of the processor and the logic of the certified source code, rather than on a mathematical verification of the computation itself.

3.3.2 Zero-Knowledge Proofs

As introduced in the preceding sections, the two principal actors in Verifiable Computation are the Prover, the party seeking to demonstrate the correct execution of a process by presenting a proof and the Verifier, who is responsible for checking its validity. One of the core challenges in VC is the need to preserve the confidentiality of information during the verification phase. In general, proving a theorem tends to be more complex and to require more information than the mere fact of the theorem being true. The verification process would therefore demand considerably more knowledge than what is contained in the proof itself, introducing a significant limitation.

To overcome this, Zero-Knowledge Proofs were introduced in 1989, defined as "those proofs that convey no additional knowledge other than the correctness of the proposition in question" [36]. To address the trust problem and minimise the information exchanged, interactive proof systems were introduced. The demonstration no longer consists of a static reading of a proof but of a dynamic conversation in which the parties interact using randomisation. In this way, the Verifier can assess the Prover's honesty through random challenges. By repeating these requests a number of times they deem satisfactory, they can become convinced of the truth of a proposition with a vanishingly small margin of error. The key contribution of ZKPs lies in the fact that the proofs reveal no additional knowledge to the Verifier beyond the correctness of the statement itself. Formally, a protocol is zero-knowledge if everything the Verifier "sees" during the interaction is something they could have simulated or computed independently in polynomial time, without any assistance from the Prover. The three fundamental requirements for defining a zero-knowledge interactive proof system are:

1. **Completeness** : this property guarantees that, if a statement is true and both the Prover and the Verifier follow the protocol honestly, the Verifier will always accept the proof.

2. **Soundness**: It guarantees that, if the statement is false, no dishonest Prover (even by cheating) will be able to convince the Verifier of its truthfulness, except with a negligible probability.

3. **Zero-Knowledge**: It guarantees that the Verifier learns no additional information from the interaction beyond the fact that the statement is true.

> In other words the Knowledge Complexity, the amount of computational information transferred, is zero.

The "Coloured Balls Protocol" is a classic example illustrating the logical workings of a ZKP. In this scenario, Peggy (the Prover) aims to convince Victor (the Verifier), who is colour-blind, that she can distinguish between two apparently identical balls—one red, one green. The protocol proceeds as follows: Victor takes one ball in each hand and presents them to Peggy. He then hides them behind his back and decides, based on a purely random choice (for instance, a coin flip), whether to swap them or leave them in their original positions. Victor then reveals his hands again and issues the challenge: "Did I swap the balls?" If the balls were identical, Peggy would be unable to detect the swap and could only guess, with a 50% chance of success. If she is telling the truth, however, she will answer correctly in 100% of cases. By repeating the test multiple times, Victor can become convinced of the validity of the claim, as the probability of Peggy succeeding by pure chance decreases exponentially. The following table illustrates how the probability of deception (Soundness Error) drops sharply as the number of challenges increases.

Table 3.3: Probability of deception as a function of the number of multiple rounds in the Zero-Knowledge protocol.

Round (n)	**Formula** ($1/2^n$)	**Probability of Deception** (Soundness Error)	**Verifier certainty** (Completeness)
1	$1/2$	50%	50%
2	$1/4$	25%	75%
5	$1/32$	3.125%	96.875%
10	$1/1024$	$\approx 0.097\%$	$\approx 99.90\%$
20	$1/2^{20}$	$\approx 0.00009\%$	$\approx 99.9999\%$
40	$1/2^{40}$	$\approx 10^{-12}$	$\approx 100\%$

Despite being theoretically and mathematically sound, the interactive model presents operational limitations that hinder its adoption in distributed contexts such as blockchain. The core issue lies in the requirement for continuous synchronous communication between the parties: this ties the process to the simultaneous presence of both Prover and Verifier, significantly increasing verification latency. While this constraint may be manageable in controlled settings, it becomes impractical in a decentralised network composed of thousands of nodes. A further limitation concerns the lack of transferability: an interactive proof mathematically convinces only the specific Verifier who participated in the exchange. In a public ledger,

therefore, every individual validator node would be forced to query the Prover independently to verify the same piece of information, making the system inherently unscalable. Finally, the factors of cost and efficiency must be considered. In distributed cryptographic protocols, network interaction is frequently the most costly resource compared to local computation [37]. While the mathematical generation or verification of a proof may take only a few milliseconds, the latency introduced by exchanging dozens of messages would render the protocol slow and expensive. To address these critical issues, Non-Interactive Zero Knowledge Proofs (NIZK) have been introduced. These systems are capable of generating a unique proof that can be verified at any time by anyone. The level of interaction is unidirectional, from the prover to the verifier. The hypothetical scenario involves two mathematicians: A and B. A leaves on a long journey to conduct scientific research and, during this period, intuits the proof of a new theorem. The objective is to validate this discovery solely by means of a postcard sent to B. The process will therefore be non-interactive and unidirectional because, even if B wanted to respond, he does not know the location of his colleague.

NIZK systems introduce the concept of a static proof, grounded in a common element known to all parties involved. This element is the Common Reference String (CRS): a structured sequence of random bits shared between the parties prior to the protocol. The CRS is what enables the transition from a direct communication model to a unidirectional one, as it implicitly encodes all the mathematical challenges required to establish the validity of the statement. Although the Prover knows these challenges in advance, the system remains secure by virtue of computational complexity: even with prior knowledge of the challenge, it is impossible to construct a valid proof without genuinely possessing the secret (witness). The ultimate function of the CRS is to compel the Prover to embed their solution within a fixed, rigid, and publicly known string. It is worth noting that not all NIZK schemes are equivalent. Protocols differ structurally in how they handle shared randomness and the initial setup phase. NIZKs can therefore be broadly divided into two distinct technological families:

1. ZK-SNARKs
2. ZK-STARKs

ZK-SNARKs

The term SNARK is an acronym standing for Succinct Non-Interactive Argument of Knowledge. It is a well-established protocol used to generate proofs that are extremely compact and efficient to verify [38]. Defining the acronym can help in highlighting the attributes:

- **Succinctness**: the proof size remains extremely compact regardless of the computational complexity or the size of the input data, enabling fast verification with minimal computational resources.

- **Non interactivity**: as discussed, the generated proof enables independent verification without requiring both parties to be available simultaneously. Non-interactivity is made possible by the introduction of the Common Reference String.

- **Arguments of Knowledge**: SNARK schemes do not merely guarantee that the Prover is telling the truth, but also that they possess explicit knowledge of the information substantiating that statement (the witness).

The first step in generating a SNARK proof involves converting high-level code (Python, C++) into an arithmetic circuit, which decomposes complex computations into simple addition and multiplication operations. This circuit is then translated into a system of linear equations known as a Rank-1 Constraint System (R1CS), which constrains the execution of the program. Finally, the matrices are converted into polynomial equations through a Quadratic Arithmetic Program (QAP). The choice of polynomials stems from the fact that two distinct polynomials intersect at very few points: if the Verifier samples a random point and the values match, there is an overwhelmingly high probability that the two polynomials are identical. One of the primary concerns with SNARK systems is the process of creating the CRS, which renders the proof non-interactive. This initialisation phase is known as the Trusted Setup, and relies on private randomness to produce the string. If the random parameters are not properly discarded after use, they become what is commonly referred to as "toxic waste": anyone in possession of these values could exploit them to generate fraudulent proofs. Another known limitation is the computational overhead required to generate the proof. Conversion to R1CS and then to polynomials requires a significant computing power from the prover, which also lengthens the time required for generation. Despite these limitations, SNARKs remain the standard for all cases where it is necessary to minimise the time and computing power required to verify the proof. For these reasons, one of its main applications is in blockchain or distributed solutions.

ZK-STARKs

The STARKs protocol was first introduced in 2018 by Eli Ben-Sasson et al. [39] as an alternative to SNARKs proofs. The proposed solution aims to overcome and limit the critical issues of the previous solution with a new approach. The acronym stands for:

- **Scalable**: This is a protocol that guarantees very low verification times even in cases of complex computations and the use of big data.

- **Transparent**: The security of the system does not depend on any secret generated in a preliminary phase. With this solution, it is therefore possible to completely eliminate the Trusted Setup and all the security criticalities and risks listed previously.

- **Argument of Knowledge**: It is the same concept presented in the previous section. It is a system in which the Prover convinces the Verifier of possessing specific information without revealing it.

The purpose of STARK systems is to make the verification of a complex calculation a fast operation by means of an algebraic problem with low-degree polynomials [40]. The first step is arithmetization, which is the translation of the computer program into a mathematical language. In this phase, an Execution Trace is generated, structured as a table in which each row represents the state of the computation at a specific step.

To ensure the correctness of state transitions, algebraic constraints are defined using Algebraic Intermediate Representation (AIR). If the data in the table satisfies these constraints, the computation is considered valid. In the next step, the data column of the trace is transformed into a polynomial. To amplify any errors and make them easily detectable, the domain of the polynomial is extended well beyond the length of the original trace. This technique is known as Low Degree Extension (LDE). The underlying mathematical idea is that if the Prover has cheated in even a single step, the resulting polynomial will have a very high degree and will not satisfy the expected properties. If, on the other hand, the computation is honest, the polynomial will have a well-defined low degree. To verify that the polynomial is indeed of low degree without having to read it entirely, the FRI (Fast Reed-Solomon Interactive Oracle Proof of Proximity) protocol is used.

Through a series of mathematical reductions and random sampling (managed using the Fiat-Shamir heuristic to remove interaction), the Verifier checks only a few specific points of the polynomial. If these few sample checks are successful, there is cryptographic certainty (with a probability close to 100%) that the entire program execution is correct. The advantages of this solution are manifold. Full transparency and, consequently, the elimination of the Trusted Setup significantly enhance security by removing the risk of toxic waste and the potential for false proof generation. The system is also highly scalable, enabling proof generation over large volumes of data in extremely short time. Recent studies show that, given the same computational domain, STARKs can offer significantly lower proof generation times, up to 10 times faster than zk-SNARKs in specific contexts [41]. Although some tests conducted on commercial hardware demonstrate that the

actual proof generation time can be longer in STARK systems due to poor library optimisation [42]. The STARK protocol also enables extremely fast verification, with a time requirement that is logarithmic with respect to the size of the data. For the reasons just listed, STARK protocols are particularly suited for big data and large-scale databases. By relying exclusively on cryptographic hash functions, STARK systems are considered secure even in a forward-looking perspective, as they can resist quantum attacks unlike SNARK systems.

Feature	**zk-SNARKs**	**zk-STARKs**
Trusted Setup	**Required**. Creates "toxic waste" risks if compromised.	**Not Required** (Transparent). Uses public randomness, eliminating setup risks.
Underlying Cryptography	Elliptic Curves (Pairings).	Hash Functions (Collision-Resistant), Polynomials.
Post-Quantum Security	**No**. Vulnerable to Shor's algorithm.	**Yes**. Relying on hash functions makes them post-quantum secure.
Proof Size	**Succinct** (∼200–800 bytes). Constant size regardless of complexity.	**Large** (40–200 KB). Logarithmic growth relative to computation size.
Scalability	**Limited**. High memory consumption for large circuits.	**High**. Ideal for big data and massive computations.
Prover Speed	Slower.	**Fast** (10× faster). Highly parallelizable.
Verifier Speed	**Constant**. Fast.	**Poly-Logarithmic**. Verified up to 2× faster than SNARKs

Table 3.4: Comparison between zk-SNARK and zk-STARK protocols.

The choice between these two protocols is primarily dictated by the requirements, the application, and the frequency of proof generation and verification. The SNARK

architecture is preferable when the priority is proof compactness. A very common application is on-chain publication: a smaller proof reduces gas costs on shared systems, where storage is generally very expensive. ZK-STARKs, on the other hand, are preferable in contexts where total transparency is a priority and where good scalability on large amounts of data is required. The absence of a trusted setup significantly reduces risks, and quantum-resistant security makes this model more mature for future challenges. Zero-Knowledge (ZK) technologies, while revolutionary, have inherent limitations related to their mathematical nature. The first critical issue is computational asymmetry: while the verification phase is efficient, proving involves massive overhead, often orders of magnitude greater than native execution. The necessary translation of the program into polynomials or arithmetic circuits introduces additional asymptotic complexity, which makes the process particularly burdensome for elaborate computations.

A second significant obstacle is the adoption barrier. Implementing a ZK solution typically requires rewriting the application logic in domain-specific languages for circuit definition (Circuit Engineering), forcing developers to abandon standard programming paradigms. Furthermore, the high computational resource demands for proof generation often necessitate the use of dedicated hardware, effectively limiting the accessibility of the technology.

3.4 Blockchain

Within the information network, the capacity to duplicate and disseminate any online document to other users is a fundamental capability. This situation is ideal for files or content, but it is problematic for money management because value must be moved, not copied. Another issue that must be addressed is that of double spending: under this scenario, the same amount of money could be sent to two different recipients at the same time, creating a conflict over who is the actual owner. In order to address these problems, a variety of strategies have been introduced with a view to facilitating the exchange of value and the recording of transactions. Initially, exchange was carried out using instruments of value (gold, barter), i.e., instruments accepted by the entire community. The advantages of such transactions included the anonymity of the parties involved and the lack of traceability. However, it was necessary for both parties involved to accept the currency of exchange. In the event of the physical instrument being destroyed or lost, the value of the instrument is irretrievably lost.

One method of mitigating these critical issues was the implementation of ledgers, through which a banking institution or a managerial figure maintained a record of income and expenses. As wealth is documented within these voluminous registers, it is not a prerequisite to possess the physical currency, thereby ensuring its value

is maintained. In the analog world, the process of documentation and accounting was performed manually, which rendered transactions susceptible to human error or illicit activities. The advent of information technology facilitated the digitisation of these ledgers, thereby reducing material errors and expediting verification. However, the management of these financial systems remained centralized, with each financial institution maintaining its own version of the ledger. This led to a reliance on an intermediary to approve and coordinate transactions, which required a degree of trust. The aforementioned factors gave rise to expenses, delays, and the foundational issue of trust: in the event of the central ledger being subjected to manipulation, compromise, or mismanagement, the entire system is rendered vulnerable.

To address these issues, in November 2008 Satoshi Nakamoto (a pseudonym) proposed a new way of transferring money that incorporated decentralization, irreversibility, and the complete absence of trust in a single authority [43]. The entire process is based on an innovative technology: the blockchain. The term was initially used in a separated form, where the word "block" referred to the collection of information related to the transaction, and "chain" indicated the connection and the way in which the various blocks were linked together using cryptographic techniques. The design goals of this solution are: peer-to-peer (P2P) distribution, decentralization, immutability, and trust among unknown entities. As seen previously, a system based on a P2P approach eliminates a central and superior entity that, according to predefined rules and protocols, could arbitrarily decide how to manage the entire network. In the case of the Bitcoin blockchain, there is no single server that can write information; instead, all nodes in the network have the right to read, and store a copy of the ledger. Thanks to this structure, it is impossible to alter the data without other entities becoming aware of it. Moreover, due to its decentralized architecture, the problem of the single point of failure is eliminated, where a malfunction, cyberattack, or breakdown could compromise the state of the entire network [44]. Distributed presence also allows for greater speed and shorter response times for read operations, as requests for information will be directed to the nearest node, without the need to query the central server. The immutability and transparency of transactions are guaranteed by the block mechanism. Once a transaction has been entered into a block and confirmed by the network, it becomes immutable and can be viewed by anyone, ensuring continuous control. In order to comprehend this concept, it is imperative to undertake a thorough analysis and understanding of the structural composition of the blocks. To send an amount of money, it is necessary to specify the value and the recipient's address and signing the operation with one's private key. The network receives the request and adds it to the list of pending operations that will be included in a block. Each block is divided into two main sections: the header and the body. The header contains the hash of the previous block's header, the Merkle root

summarizing the current transactions, and the timestamp of creation. The body comprises all transactions that have been validated [45]. The concept of Merkle root bears a strong resemblance to that of Merkle DAG, as previously elucidated in the context of IPFS. In this particular instance, transactions are aggregated in pairs and incremental hashes are calculated until the root sequence, i.e., the final one, is obtained. It is evident that even a minimal change will result in a complete alteration of the hash. This configuration establishes a cryptographic constraint that renders modifications to previously recorded blocks highly computationally intensive. This is due to the fact that recalculating all subsequent blocks would be necessary.

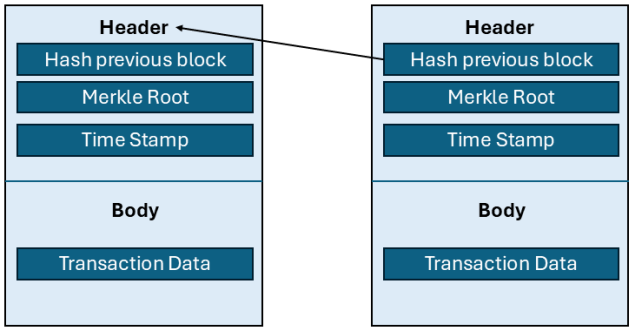

Figure 3.5: Schematic representation of the block structure

To ensure the correctness of the ledger update, consensus mechanisms are used [46] to determine which node has the right to propose the next block and which version of the blockchain is considered valid in case of conflicts. Once a block has been proposed, the other nodes verify its validity by checking the transactions and their compliance with the protocol rules. If valid, the block is accepted and added to the chain. The main consensus mechanisms are:

- **Proof of Work (PoW)**: This solution, devised by Nakamoto for the Bitcoin network, involves solving a complex mathematical puzzle. All nodes begin a competition until they find a valid hash that meets the target. Computing power is fundamental to this protocol: the more powerful the hardware, the

greater the number of attempts per second to find the correct combination first. Once a valid block has been identified, it is transmitted to the network and verified by the other nodes. These nodes then perform a single instantaneous calculation to check its validity, making verification far simpler than finding the solution. To encourage participation and incentivise honest behaviour, the node that identifies the solution to the problem receives a financial reward in bitcoin. The only way to produce blocks faster than the entire network and force a valid alternative chain is to possess more than 51% of the network's computing power (expressed in hashrate); that is, to possess more computational resources than all other nodes in the world combined. This is an extremely improbable scenario, and it has never occurred in the Bitcoin network. The main problems with this solution are the high computational capacity required and the consequent electricity consumption, limited scalability, and hardware centralisation. The competition is so fierce that the majority of blocks are proposed by enormous mining pools, almost entirely excluding smaller nodes from the process.

- **Proof of Stake (PoS)**: This is an alternative to PoW, which aims to reduce its main limitations, such as high electricity costs and the computing power required. Validator nodes are not required to solve complex mathematical problems; they are selected at random by a protocol. The likelihood of selection is directly proportional to the amount of cryptocurrency that is locked (staked) as collateral. The correct execution of the architecture is ensured by the fact that any malicious node attempting to alter the blockchain is punished and may lose part of the amount staked. This phenomenon is known as slashing. Consequently, any attempt to attack the system would be both costly and counterproductive. Furthermore, if a validator does not participate in the process for days or weeks, they will gradually lose their stake. In summary, this system encourages all participants to behave correctly and to collaborate to ensure the blockchain is both honest and reliable. Validators with a larger stake are more likely to be chosen and therefore to receive rewards, creating a centralisation of validation and wealth. This is analogous to the situation previously identified with PoW, where large mining pools centralised the validation process.

- **Proof of Authority (PoA)**: This consensus mechanism requires neither computing power nor locked capital. Block validation is entrusted to pre-established and authorised validators. Each of them must provide real and accurate information about their identity, investing financial resources and personal reputation to ensure long-term trust. These nodes are incentivised to behave correctly so as not to lose the trust they have built over time and, consequently, their right to validate blocks. In this scenario, decentralisation

is compromised because trust is concentrated in a restricted group of entities [47], yet the overall system is more efficient and scalable. This solution is adopted in corporate, governmental, or consortium contexts where greater throughput, low costs, and known accountability are required.

Over time, numerous blockchain-based projects have been proposed, each with entirely different characteristics and functions. The main distinctions can be drawn along two independent dimensions: data access and consensus participation [48]. In public solutions, all parties have the ability to consult the transaction history and access the ledger. In the latter case, only a list of authorised entities has the right to read it. In a permissionless solution, any node may act as a validator, whereas in a permissioned blockchain, only a selected set of approved nodes can validate blocks [49]. The two classifications are independent of one another. A blockchain can be public yet permissioned, or private yet permissionless. The most common configurations in practice are public permissionless and private permissioned. The table below illustrates all possible combinations. It should be noted that the private permissionless configuration, while theoretically possible, is extremely rare in practice.

Type of blockchain	**Data access**	**Consensus**
Public + Permissionless	Open to everyone	Open to everyone
Public + Permissioned	Open to everyone	Selected validators
Private + Permissionless	Restricted	Open to everyone
Private + Permissioned	Restricted	Selected validators

Table 3.5: Classification of blockchains by access and consensus

It is essential to understand the basic principles of blockchain in order to address its main extensions, such as smart contracts and oracles, which expand its application potential.

3.4.1 Smart Contracts

The widespread adoption of blockchain technology and the possibility of recording transactions, content, and legal acts online attracted the attention of researchers and developers, generating numerous implementation ideas. The initial concept of smart contracts was introduced by Nick Szabo in 1994 [50] who envisioned automated transaction protocols capable of executing the terms of a contract. The objective was to reduce or even eliminate the need for trusted intermediaries to guarantee the correctness of the document, introducing concepts such as verifiability

and observability. The idea was groundbreaking, but the technologies required to implement it in practice were not yet available. The advent of blockchains such as Ethereum [51] has paved the way for the practical and feasible implementation of this concept. A smart contract is code that is executed upon the occurrence of specific conditions, known as triggers. The potential applications of this technology are numerous and varied, including property transfers, real estate transactions, automated trading and estate management. The unique characteristics of blockchain, including consensus mechanisms, immutability, and irreversibility, eliminate the need for a third-party notary to validate the contract and associated costs. The most widely used blockchain for smart contract development is Ethereum, and the programming languages employed are Vyper and, most notably, Solidity. The latter, syntactically similar to JavaScript, allows contracts to be written and subsequently compiled into bytecode and executed on the Ethereum Virtual Machine (EVM), a decentralised Turing-complete virtual machine that guarantees the deterministic execution of code across all nodes in the network. In order to understand the structure of smart contracts fully, it is important to introduce the concept of gas. Gas is the economic mechanism used by this type of program and corresponds to transaction costs. The fundamental idea is that every operation (except reading operations) has a price, so the more complex the code, the higher the amount to be paid. This mechanism was also introduced to solve the Halting problem, i.e. the impossibility of determining in advance whether a program will terminate or run indefinitely. Once the available gas has been used up, the contract will finish executing and stop, thus preventing infinite loops. Without this system, a malicious actor could attack the blockchain by sending thousands of spam transactions at no cost, creating programmes with endless loops and executing highly intensive operations, significantly slowing down the entire system. The high economic cost associated with such operations makes attacking the network extremely expensive, thus effectively deterring malicious actors. Each contract must include a parameter setting the maximum amount of gas the sender is willing to spend. This value is known as the gas limit. If the value is set too low, the transaction will fail mid-execution, resulting in the loss of the gas consumed up to that point. Setting a sufficiently high value ensures the transaction completes successfully, even at elevated costs. The gas price is instead determined by two parameters: the base fee and the priority fee. The first value indicates the minimum fee required to include the transaction in a block. It is a mandatory cost that must be paid by all. This value is automatically adjusted by the network based on congestion. The amount is completely burned and permanently removed from total circulation, making Ether (Ethereum's cryptocurrency) potentially deflationary. During periods of high activity, the amount of Ether burned may exceed the amount given as a reward to validators. Base fees are non-negotiable and are determined algorithmically by the network. Priority fees, on the other hand, are an optional value that serves to

incentivise network validators. It is important to note that the higher this value, the faster the confirmation speed will be. This is due to the economic attractiveness of validating the transaction in that block. This approach ensures that only the most profitable and attractive transactions are selected. The following formula should be used to calculate the cost of a transaction:

$$\text{Total cost} = (\text{Base fee} + \text{Priority fee}) \cdot \text{Gas used}$$

The unit of measurement for gas is Gwei, with a value equivalent to 10^{-9} Ether. The most significant impact on transaction costs is related to the complexity of the program and the number of operations it performs. The more complex, lengthy, and computationally intensive the code, the higher the price to be paid for its execution. As well as the costs of executing the functions, it is necessary to consider the deployment costs, i.e. the initial publication of the contract on the blockchain. These costs, borne only once by the developer, include bytecode storage (approximately 200 gas per byte) and the execution of the constructor, which initialises the state variables. Once deployed, the code becomes immutable and permanent on the blockchain, making careful optimisation prior to publication essential. The table below provides an overview of the costs of the main operations [52].

Operation	Gas Cost
ADD, SUB, EQ, AND, OR, XOR	3
MUL, DIV	5
SLOAD (read from storage)	2,100
SSTORE (0 → non-zero)	20,000
SSTORE (non-zero → non-zero)	5,000
SSTORE (non-zero → 0)	5,000 (with possible refund up to 15,000)

Table 3.6: Gas cost of common EVM operations.

Due to these constraints, developers face significant challenges in producing efficient smart contracts that minimise unnecessary costs. The main issues concern the difficulty of identifying all possible cases and scenarios, immature testing frameworks, and a lack of documentation [53]. Some studies [54] have identified that approximately 4.1% of contracts deployed on the Ethereum blockchain are susceptible to unbounded iteration vulnerabilities.

3.4.2 Oracles

As explained in the previous section, smart contracts are programs stored on a blockchain that execute automatically when specific conditions are met. However, a blockchain is a closed system: it can only access data already existing on the chain itself. This constraint significantly limits the applicability of such solutions. For instance, a smart contract cannot natively retrieve external temperatures, verify the completion of a construction project, or access the output of a software application. This isolation effectively prevents the direct use of off-chain triggers. To address this challenge, oracles were introduced as a bridge between off-chain and on-chain domains. These services retrieve, verify, and supply external data to smart contracts through various methods, such as API calls, sensor readings, or database queries. Oracles can be primarily categorized based on three core criteria [55]: the data source, the direction of information flow, and the adopted architecture. The first classification includes:

- **Software Oracles**: they retrieve information from online sources such as websites and servers, and relay it to the smart contract. One of the main advantages of this solution is their ability to fetch real-time data, making them suitable for contracts that need to monitor the price of an asset (gold or stocks), an airplane's delay, or a currency exchange rate.
- **Hardware Oracles**: these oracles have been developed to retrieve and send information about the physical world. Examples of such metrics include measurements from electronic sensors, the Internet of Things (IoT), RFID tags and QR scans.
- **Human Oracles**: in this case, a physical person serves as the data source. The information provided requires human judgment, evaluation, or verification to ensure its accuracy. Examples include the economic appraisal of an object or the quality assessment of a provided service.

While the most prevalent application of an oracle involves capturing external data and delivering it to the blockchain, the inverse operation is also feasible. Consequently, the classification based on the direction of information flow is defined as follows:

- **Inbound Oracles**: this is the most common scenario in which off-chain data is transferred to the smart contract.
- **Outbound Oracles**:In this configuration, information or triggers are conveyed from the on-chain environment to the off-chain world. A practical example is a smart lock that is programmed to unlock only after the required funds have been confirmed on a specific blockchain address.

The final classification concerns the type of architecture adopted:

- **Centralized Oracles**: In this configuration, the system relies on a solitary information source. Implementing such a solution entails significant risks, as the contract's execution depends exclusively on the reliability of a single node; consequently, any data manipulation could fatally jeopardize the smart contract's integrity. Furthermore, this model introduces a critical "single point of failure", where an oracle malfunction or downtime can compromise the entire architecture, significantly undermining overall system resilience.

- **Decentralized Oracles**: This solution is not based on a single source of truth; it queries several separate entities to determine the accuracy and correctness of the information.

Oracles constitute an essential yet delicate component of the blockchain ecosystem. Relying on centralized sources risks undermining the guarantees of decentralization and immutability inherent to blockchain systems, effectively transforming a trustless architecture into one that depends on trust in a single operator. A compromised or malicious centralized oracle may manipulate the data supplied to smart contracts, leading to incorrect behavior and potentially significant economic losses, regardless of the correctness of the on-chain logic. This issue is particularly critical in DeFi (Decentralized Finance) applications, where billions of dollars depend on the reliability of price feeds. For this reason, critical applications should prioritize decentralized solutions that maintain architectural consistency with the founding principles of blockchain technology, accepting higher costs and complexity as a necessary trade-off to preserve the security and trustlessness of the overall system.

The technologies presented in this chapter aim to introduce possible alternatives that can be used in a transparent system. IPFS provides a decentralised storage system, ensuring constant and continuous access to data. Adopting this architecture reduces dependence on central servers and external companies, thus minimising the risk of single points of failure. Trusted Execution Environments (TEE) and Zero-Knowledge Proofs (ZKP), despite their differences in approach and security guarantees, facilitate the validation of the algorithm's correct execution without compromising proprietary information or sensitive data. The use of blockchain, with its smart contracts and oracles, provides a public, tamper-proof ledger that anchors results and cryptographic evidence. It is crucial to emphasize that the adoption of these technologies does not automatically guarantee a transparent and verifiable system. Only through proper integration and well-defined traceability mechanisms is it possible to minimize the level of trust required across the entire architecture. The primary challenge lies in the coordination of these stages and the formulation of an architecture that unifies these tools into a single process. The

following chapter will present the project case study, the system design, and the practical implementation. Furthermore, the proposed architecture will be examined through analysis, benchmarks, and alternative solutions, highlighting the current state of the art, the results achieved, existing limitations, and future developments.

Chapter 4

Implementation of the Verifiable SHM Pipeline

The previous sections introduced the concept of strategic infrastructure and its fundamental role in a territory's socio-economic fabric. The benefits of these works were analyzed, along with the direct link between urban and logistical development and their proper functioning. Safety therefore becomes a priority, managed through structural health monitoring (SHM) systems. However, the analysis revealed several critical issues with these systems, primarily concerning the lack of transparency and verifiability in decision-making processes. These factors reduce stakeholder confidence and compromise data traceability. To address these challenges, Chapter 3 explored the potential of decentralized technologies such as IPFS, Trusted Execution Environments (TEE), and Blockchain to ensure integrity and immutability. This chapter combines the problem analysis and theoretical tools to propose a concrete solution: designing and implementing a verifiable SHM pipeline. This will involve describing the system architecture, from off-chain calculation modules to on-chain smart contracts, and demonstrating through a case study how technological guarantees can be used to ensure the accuracy of structural monitoring.

4.1 System Architecture and Design Overview

SHM systems play a key role in infrastructure safety. They enable constant, real-time monitoring of structural values by taking continuous measurements. This architecture enables interventions to be organised at the first sign of critical issues, eliminating the need for regular scheduled maintenance. Bridges and viaducts are constantly exposed to traffic loads and atmospheric phenomena like temperature variations, wind and rain. Distinguishing genuine structural damage

from Environmental and Operational Variability (EOV) is therefore critical. A structure can show changes in its dynamic response simply due to environmental effects, without any actual deterioration. Damage detection must account for this distinction, otherwise false alarms undermine the reliability of the entire monitoring system.

4.1.1 The Yonghe Bridge Case Study

The reference scenario for this project is the Yonghe Bridge, a cable-stayed structure located in Tianjin, China. The bridge features a main span of 260 meters, flanked by side spans of 25.15 meters and 99.85 meters, for a total length of 510 meters. The deck, which is 11 meters wide, consists of 74 precast concrete segments connected by cast-in-place joints. The supporting towers reach a height of 60.5 meters and incorporate two transverse cross beams. The stay system comprises 88 pairs of steel cables, each containing between 69 and 199 wires depending on the specific load requirements.

Figure 4.1: Picture of the Yonghe Bridge, Tianjin, China [56]

The structure entered service in December 1987. In 2006, after 19 years of operation, cracks up to 2 cm wide were detected in the central span and main deck. Maintenance interventions lasted two years, culminating in the deployment of an advanced SHM system comprising over 150 sensors distributed across the structure. Fourteen uniaxial accelerometers were placed on the road surface to measure vibrations caused by traffic and environmental factors, and a biaxial

accelerometer was installed on the tower to monitor horizontal oscillation and overall stability. Both devices operate at a 100 Hz sampling rate, yielding 360,000 measurements per hour. Data are stored in 17-column matrices, with the first column recording timestamps; columns 2–15 holding the uniaxial accelerometer readings; and the final columns capturing outputs from the south tower's biaxial sensor.

4.1.2 The Trust Gap: From Detection to Auditability

SHM systems possess the ability to measure, store, record, and process large amounts of data in real-time. These instruments are designed to provide precise assessments based on measurements collected by sensors installed on the structure. The collected signals undergo a series of preprocessing and filtering operations with the main aim of eliminating the noise and differentiating the structural damage from EOV.

In the current architecture, the entire data processing pipeline is managed by internal operators or contracted maintenance entities. This centralized control model limits transparency and prevents independent third parties from verifying either the integrity of the collected data or the correctness of the processing procedures. Thus, the overall system does not offer robust guarantees with respect to auditability and trustworthiness. A first critical issue concerns data integrity. Raw acceleration streams generated by the fourteen accelerometers are transmitted to a central server, where they are stored in plain formats (e.g., CSV or binary files) prior to processing. Since the data is neither cryptographically signed at the source nor anchored to an immutable registry, its integrity relies entirely on the trustworthiness of the infrastructure and its administrators. A malicious insider or a compromised privileged account could manipulate specific acceleration values to suppress evidence of structural anomalies (false negatives) or to fabricate non-existent damage (false positives). In the absence of integrity protection mechanisms, such modifications may remain undetected.

A second issue relates to code integrity and algorithmic transparency. The damage detection algorithm executes on the same centralized infrastructure and is not externally verifiable. External stakeholders, such as insurance providers or regulatory authorities, have no cryptographic assurance that the software deployed on the server corresponds exactly to the certified or validated version of the algorithm. An altered implementation could selectively suppress alerts or modify detection thresholds without leaving verifiable evidence. Consequently, the system provides no formal guarantee that the reported outputs genuinely reflect the execution of the approved algorithm over untampered data. Overall, the absence of verifiable guarantees for both data integrity and code integrity introduces a structural trust gap in the monitoring pipeline. This limitation motivates the

need for mechanisms capable of providing cryptographic attestation and tamper-resistance across the entire processing workflow.

4.1.3 System Components and Data Flow

The system presented in the following sections addresses the auditability and security shortcomings identified earlier. The architecture is hybrid, combining off-chain processing with immutable on-chain recording. The most demanding phases run locally through verifiable execution tools, which preserves performance and avoids the prohibitive costs of full on-chain execution. The blockchain acts as a public, permanent ledger for cryptographic proofs and as the execution platform for the damage detection smart contract. The system takes as input the sensor measurements already organized into matrices as described in the previous sections. From there, the pipeline is structured around the following logical layers:

1. **Decentralized Storage Layer (IPFS)** In this phase, the sensor matrices are stored on IPFS, a decentralised file system that guarantees immutability and content identification through a Content Identifier (CID). This approach eliminates dependency on centralised servers or paid cloud services, distributing the load across the peer-to-peer network at no cost. Each file is assigned a unique identifier that is cryptographically bound to its content. Any modification, however minor, produces a completely different CID. This makes data tampering immediately detectable. The CID mechanism also plays a practical role in the on-chain integration. Storing large datasets directly on a blockchain is prohibitively expensive. IPFS solves this by allowing the system to publish only the Identifier on-chain, while the actual data remains accessible on the distributed network.

2. **Off-Chain Computational Layer** This component is the computational core of the SHM pipeline. The main algorithm runs here, processing raw sensor readings through filtering and signal decomposition techniques to remove noise and isolate genuine structural behaviour from Environmental and Operational Variability (EOV). This phase serves two purposes. First, it handles the heavy computational workload, including floating-point operations and large matrix manipulations that would be impossible to execute directly on a blockchain. Second, it generates a cryptographic proof attesting that the final result derives exclusively from the correct execution of the authorised algorithm. Generating this proof is the most critical step in the entire process. By condensing the correctness and integrity guarantee into a compact binary object, it makes on-chain verification of the result both feasible and efficient.

3. **Oracle Connector** The oracle, as discussed in the preceding sections, acts as a bridge between off-chain computation and the smart contract. Its purpose is

to transmit the numerical outputs generated by the algorithm to the blockchain. It consists of a Python script that does not perform any data validation or verification; instead, it manages the pipeline automation. It retrieves the CIDs for the input data, cryptographic proof and algorithm output previously uploaded to IPFS, then builds, signs and broadcasts a transaction to the smart contract. Within the proposed framework, security does not rely on this layer, as the oracle is limited to communicating information that has already been proven. Submitting values to the blockchain that deviate from those processed by the SHM program would render the alteration immediately detectable during the subsequent verification steps performed to ensure the system's correctness

4. **State Evaluation and Settlement Layer** The contract receives the numerical outputs computed off-chain and applies deterministic threshold logic to assess the infrastructure health status, classifying it according to the severity of the detected values. Rather than storing full datasets or heavy logs, the state variables hold only the evaluation result and three identifiers: the CID of the input data, the CID of the algorithm output, and the CID of the Remote Quote. Anyone querying the contract can immediately read the current infrastructure status and, by retrieving the corresponding files from IPFS, independently verify the authenticity and chain of custody of the data.

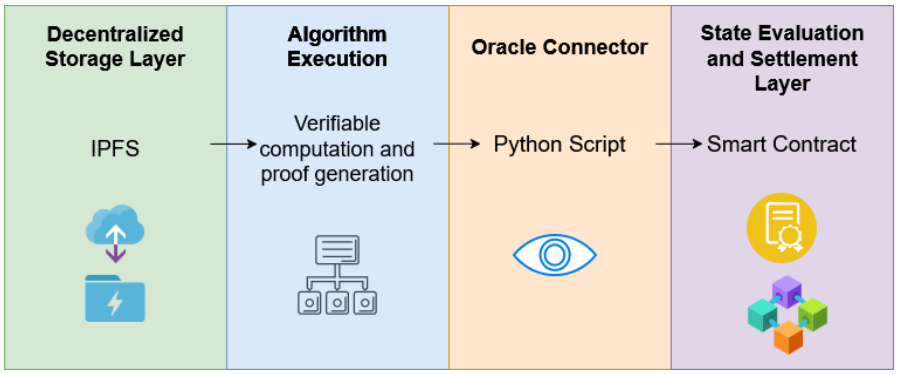

Figure 4.2: Data Flow

4.2 The SHM algorithm Workload

This section covers the full logic and structure of the SHM algorithm. Originally written in MATLAB and later converted to Python, the algorithm processes raw accelerometer data collected from the bridge. Its primary goal is to filter out environmental vibrations and correctly assess the structural condition of the

infrastructure. The software relies on advanced signal analysis techniques operating on floating-point values and large-scale matrix decompositions, a computational load that makes direct on-chain execution unfeasible. The algorithm structure, filtering pipeline and damage identification logic are derived directly from the methodology proposed by Zunino, Casas and Domaneschi in their study on the Yonghe Bridge [57].

4.2.1 Data ingestion

The program extracts and processes a total of 48 hours of data: 24 hours from the month of January and another 24 hours corresponding to the specific day under consideration. The selection of these two time intervals is strictly aligned with the anomaly-detection strategy. The January data are used as a structural baseline: during that period, the bridge was in a state of confirmed structural integrity (undamaged), having recently reopened to traffic. These measurements provide the algorithm with the characteristic dynamic response of the deck, accurately describing how the intact structure reacts to normal environmental and operational variations (EOV), such as thermal excursions and vehicular traffic. The acquisition of this reference dataset is an essential prerequisite for evaluating subsequent measurements. The algorithm uses the January data as a calibration reference to analyze the data streams recorded in August, when an actual structural damage occurred (cracking of the deck and failure of several stay cables). Damage is then identified by computing the deviation of the identified modal frequencies with respect to the initial baseline.

4.2.2 Damage Detection Pipeline

The first operational step of the algorithm is the retrieval of the sensor measurements. The Python program downloads the data directly from IPFS using the content identifier (CID). Associated with the same CID are the matrices containing the January measurements (used as the baseline) and those corresponding to the time under consideration (August). Only after this initial step does the actual mathematical processing begin. All signals acquired from the accelerometers are filtered using a Chebyshev Type-II filter with a cut-off frequency set at 40 Hz. The purpose is to remove noise introduced by traffic and high-frequency components, which are considered negligible for a long-span bridge structure. The filtered signal is then processed by Variational Mode Decomposition (VMD). This technique allows the separation of the different modal components of the bridge, decomposing the vibrations into Intrinsic Mode Functions (IMFs). To each IMF, the Hilbert Transform is applied in order to extract the instantaneous frequencies of the system, which are a key feature for damage identification. To stabilize the results, the

frequency vectors are smoothed using a nonlinear median filter. The key step of the algorithm is the application of Principal Component Analysis (PCA), which isolates and removes Environmental and Operational Variability (EOV) caused by temperature fluctuations and vehicle traffic. The first principal component (PC1) is discarded, as it historically captures over 65% of environmental variance. This preserves only the deviations attributable to genuine changes in structural dynamics. The comparison between baseline frequencies (January) and potentially damaged data (August) is then validated statistically through a two-sample t-test at a 99% confidence level, mathematically confirming the significance of any detected anomaly. The final consolidation relies on symbolic analysis and clustering. Data is aggregated into hourly blocks using median and interquartile range (IQR) metrics, over which an unsupervised K-means algorithm is applied. For each window, the algorithm computes the Difference between Clusters (DC), measuring the distance from the cluster centroids. The Damage Index (DI) is obtained by subtracting a statistical guard threshold called the Confidence Boundary (CB) from the DC. Negative values indicate normal conditions, positive values indicate potential damage. Since the resulting DI values are floating-point, they are natively incompatible with the Ethereum Virtual Machine (EVM) and the smart contract. To overcome this architectural constraint, the final step converts the results to integers through integer scaling, multiplying by a factor of 10^{14}.

Figure 4.3: SHM algorithm pipeline.

This highlights the substantial computational demand of the algorithm. Operations such as VMD decomposition, PCA and clustering on matrices with hundreds of thousands of floating-point samples require dedicated processing power and specialised scientific libraries. These constraints make on-chain execution unfeasible due to the high Gas costs and lack of native support for complex mathematical operations on the EVM. This engineering limitation justifies the hybrid architecture: the intensive computation must run off-chain, using verifiable computation techniques to generate a cryptographic proof certifying its correctness.

4.3 The Trusted Computation Model

The previous sections of this work presented the problem of opacity and the high level of trust required in traditional SHM models. In a closed system, inaccessible from the outside, all pipeline stages occur in a centralized and private manner.

There is no public evidence of the correct algorithm execution, the input data used, or the correctness of the produced results. To address these limitations, two Verifiable Computation alternatives were proposed: Zero-Knowledge proof systems and Trusted Execution Environments. Both guarantee computational integrity, but through different mechanisms and trust models. Having defined the workload structure, the volume of processed values and the required computational power, the two approaches can now be compared in order to identify which architecture best fits the case study. Zero-Knowledge systems present a structural limitation that makes them unsuitable for the purpose of this thesis. This technology operates on finite fields, i.e. integers. Floating-point values are not natively supported and require conversion into arithmetic circuits following the IEEE-754 standard. This process introduces an extremely high overhead, increasing the prover time for a single 32-bit multiplication by approximately 9,000 times [58].

Table 4.1: Constraint overhead introduced by IEEE-754 floating-point conversion in ZK circuits.

Operation	**IEEE-754 Overhead (R1CS constraints)**
Addition 32-bit	2,456×
Multiplication 32-bit	8,854×
Addition 64-bit	15,637×
Multiplication 64-bit	44,899×

The structural monitoring of the Yonghe Bridge generates 360,000 samples per hour per sensor, which require analysis windows of 600 seconds (60,000 samples) to ensure statistical accuracy. Considering that the VMD phase can necessitate up to 10,000 iterations on each data block, the total number of constraints would reach orders of magnitude that are only executable with extremely powerful hardware, requiring hundreds of gigabytes of RAM. Moreover, the time required for test generation would be prohibitive. The VMD process is the most mathematically complex step and represents the true bottleneck of the entire algorithm. The operations performed on the values measured by a single sensor can be estimated with the following formula:

$$\text{Ops} = I \times 4K \times \frac{N}{2} \times \log_2(N) \times 4$$

Using the parameters identified by the SHM system presented above:

$$\text{Ops} = 10{,}000 \times 4 \times 6 \times \frac{60{,}000}{2} \times \log_2(60{,}000) \times 4 \approx 4.61 \times 10^{11}$$

Applying the overhead introduced by the conversion of floating-point values to integers according to the IEEE-754 standard, the number of constraints in Zero-Knowledge systems becomes:

$$\text{Ops}_{\text{FP32}} = 10{,}000 \times 4 \times 6 \times \frac{60{,}000}{2} \times \log_2(60{,}000) \times 4 \times 8{,}854 \approx 4.07 \times 10^{15}$$

$$\text{Ops}_{\text{FP64}} = 10{,}000 \times 4 \times 6 \times \frac{60{,}000}{2} \times \log_2(60{,}000) \times 4 \times 44{,}899 \approx 2.07 \times 10^{16}$$

As demonstrated, the computational overhead required for ZK proof generation is prohibitive. Although this technology represents the highest level of cryptographic verifiability, completely eliminating the need for trust through succinct and non-interactive mathematical proofs, it remains currently inapplicable in many practical contexts. The maturity level of ZK systems makes them unsuitable for situations requiring the management, manipulation and analysis of large volumes of data, as in the case of structural monitoring. Trusted Execution Environments base the security and verifiability of execution at the hardware level rather than on cryptographic and mathematical proofs. As introduced in Chapter 3, solutions such as Intel SGX allow code to be executed in secure enclaves, isolated from the rest of the operating system. This guarantees the integrity of the execution flow and the confidentiality of data in use. Unlike ZK circuits, which operate on finite values, TEEs execute floating-point operations natively following the IEEE-754 standard, with performance close to that of a standard execution. The computational costs introduced by TEE are mainly related to context switching operations and the paging of encrypted memory (EPC). In a well-optimized system the overhead is approximately 1.5x–2x compared to native execution, with peaks up to 16x in non-optimized systems [59]. The overhead values are highly variable and depend on the program context and the type of operations performed.

Table 4.2: Technical comparison between Zero-Knowledge Proofs (ZKP) and Trusted Execution Environments (TEE) in the context of Structural Health Monitoring.

Feature	**Zero-Knowledge Proofs (ZKP)**	**Trusted Execution Environments (TEE)**
Trust Model	**Minimal (Trustless):** Based exclusively on mathematical and cryptographic assumptions.	**Medium:** Requires trust in the hardware manufacturer (Intel) and the cloud provider (Azure).
Proof Generation Time	**Extremely High:** Hours or days for complex circuits like VMD (10^{15} constraints).	**Near-Instantaneous:** The attestation report occurs in milliseconds.
Proof Object	**Logical Integrity:** Proves that a specific input produces an output according to a fixed mathematical circuit.	**Binary Integrity:** Guarantees that a specific code hash was executed within an isolated area.
Scalability	**Low:** Heavily limited by the RAM required for the Prover and the circuit depth.	**High:** Near-native scalability; handles data volumes limited only by available RAM/EPC.
Floating-point Support	**Poor:** Requires software emulation (IEEE-754) with computational overhead up to 9000×.	**Native:** No performance loss.

The correctness of program execution within an isolated environment is guaranteed by the generation of a cryptographic proof called quote. It consists of a binary file containing identifying information about the software used (MRENCLAVE), the identity of the enclave signer (MRSIGNER) and an additional field called Report Data, available to the user. ZK systems offer verifiability based on the mathematics of computation, cryptographically binding input, execution and output. TEE architectures guarantee execution integrity through environment isolation. For the Yonghe Bridge case study, the use of Trusted Execution Environments allows

computational efficiency to be maintained, avoiding the floating-point issues of ZK systems described above. It is important to note that solutions such as Intel SGX do not guarantee the correctness of the results, nor the use of the declared input data, but only that the binary associated with its identifier has been executed.

4.3.1 Gramine Shielding and SGX Deployment

Based on the computational limitations discussed above, the design solution adopted to guarantee SHM monitoring integrity is the TEE architecture. The main challenge consists in adapting native Python code to run within an Intel SGX enclave. By design, enclaves are built to execute compiled binaries, typically written in C or C++, making the execution of interpreted languages a significant obstacle. Rewriting the entire algorithm would be a complex engineering challenge and could also result in the loss of some specific scientific libraries (such as vmdpy or scipy), with a consequent risk of numerical discrepancies and performance degradation. The adopted solution was to integrate the Python code with Gramine, an open-source Library Operating System that allows Linux applications to run inside a Trusted Execution Environment with minimal code modifications [60].

The integration of Gramine is based on a layered structure that separates the trusted code, running inside the enclave, from the untrusted host operating system. The Python code operates exactly as in a standard environment, invoking the necessary functions. The interpreter runs inside the enclave, with all data structures and the execution stack protected by the hardware AES-128 encryption implemented by the CPU. The primary function of LibOS is to manage and translate syscalls, i.e. standard Linux system calls (e.g. reading from IPFS). Gramine intercepts the request and, when possible, attempts to handle it internally or forwards the request to the host operating system via controlled transitions (OCALL). This approach preserves the POSIX semantics expected by Python applications while ensuring the cryptographic isolation required by SGX.

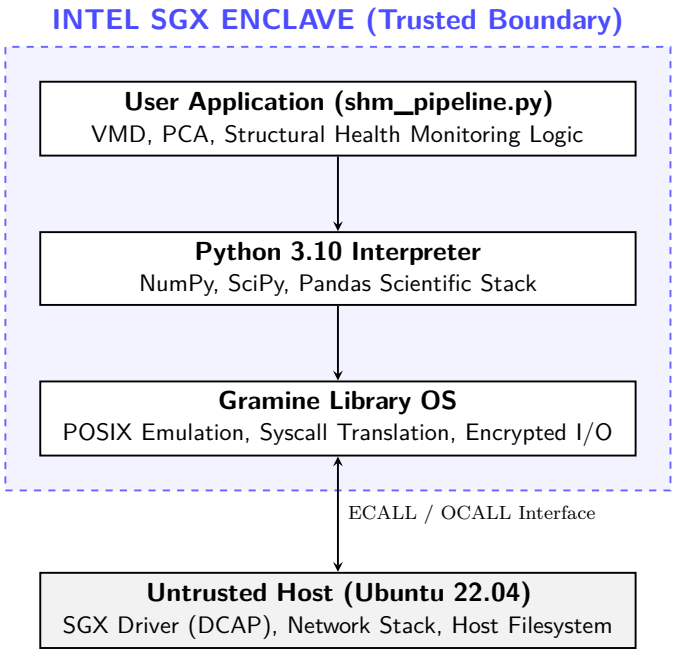

Figure 4.4: Layered architecture of the SHM monitoring system protected by Intel SGX and Gramine LibOS.

Deploying a Python application in Gramine requires the definition of a manifest file, a TOML document that specifies all enclave parameters and dependencies. The manifest contains the list of all files declared as trusted, whose hash is verified at enclave boot time, ensuring that only approved versions are accessible. For the specific case study, it is generated from a template using the following command:

```
gramine-manifest \
  -Dlog_level=error \
  -Dentrypoint=$(realpath $(which python3)) \
  -Darch_libdir=/lib/x86_64-linux-gnu \
  python.manifest.template python.manifest
```

This step uses placeholders that allow dynamic values to be resolved at build time, making the structure reusable across different environments without manual modifications. The manifest also includes a dedicated section for enclave memory configuration.

```
sgx.enclave_size = "32G"
sgx.max_threads = 64
sgx.remote_attestation = "dcap"
```

The enclave is configured with 32 GB of memory, allowing it to handle the large volume of data and processes required by the algorithm. This value must be a power of 2 and cannot exceed the available RAM on the host machine. The sgx.max threads parameter enables NumPy parallelization, while the selected attestation method is DCAP (Data Center Attestation Primitives). The manifest contains the list of all trusted files authorized to operate inside the enclave. This includes the Gramine kernel, system libraries (LibC, libpthread), the Python interpreter and the main algorithm script. During boot, Gramine computes the SHA-256 hash of each file and compares it against the expected value: any modification, even a single byte, causes verification to fail and execution to halt. Allowed files, meanwhile, can be read without integrity verification. These include known files such as the dependency list (containing thousands of entries), the .env files read by the program and system files.

```

sgx.trusted_files = [
  "file:{{ gramine.libos }}",
  "file:{{ gramine.runtimedir() }}/",
  "file:{{ arch_libdir }}/",
  "file:/usr/{{ arch_libdir }}/",
  "file:/usr/lib/",
  "file:/usr/local/lib/",
  "file:Thesis_algorithm_tee.py",
  "file:{{ entrypoint }}",
]
sgx.allowed_files = [
  "file:venv/",
  "file:requirements.txt",
  "file:input.env",
  "file:/etc/nsswitch.conf",
  "file:/etc/ethers",
  "file:/etc/hosts",
  "file:/etc/group",
  "file:/etc/passwd",
  "file:/etc/resolv.conf",
  "file:/tmp",
  "file:/etc/sgx_default_qcnl.conf",
  "file:output/",
```

Once the template is configured, the manifest signing step can be executed.

```
gramine-sgx-sign --manifest python.manifest --output python.manifest.sgx
```

The first component (gramine-sgx-sign) reads the manifest and computes the MRENCLAVE value: a hash that incorporates the hashes of all trusted files, the

enclave configuration parameters (such as enclave size and attestation type), and the memory layout, meaning the arrangement of code, data and stack sections in the enclave's virtual address space. The result is a unique string. Changing even a single parameter will cause the entire hash to change completely. Gramine digitally signs the manifest using an RSA-3072 private key. This signature produces a second identifier called MRSIGNER, which is the hash of the public key and identifies the entity that authorized the enclave. Signing is a fundamental step for remote attestation because, when the enclave is launched, the SGX processor reads the manifest, recomputes MRENCLAVE using the same logic, and compares the expected value against the computed one, rejecting enclave creation if any discrepancy is found. Finally, both MRENCLAVE and MRSIGNER are stored in hardware-protected registers accessible only to the enclave itself. During the remote attestation phase, the enclave can generate a quote signed by the processor, containing MRENCLAVE and MRSIGNER. The signature transforms the manifest from a simple configuration file into a non-repudiable cryptographic commitment of the running software.

4.3.2 Remote Attestation Mechanism

One of the main advantages of adopting Gramine as the LibOS is the ability to run Python code without any rewriting or adaptation. The entire SHM algorithm, from raw data ingestion to output values, remains unchanged from the original version. The only additions introduced concern the generation of the cryptographic proof, which do not affect the logical and mathematical flow. To make the sensor data verifiable, an identifying hash was computed. The string is built by concatenating the IPFS dataset CID with the execution date, to which a SHA-256 function is applied. This process cryptographically binds the identity of the input data and the execution timestamp, making each program run uniquely identifiable.

```
1 date_string = date.today().isoformat()
2 ROOT_CID = os.getenv("CID")
3 input_data_proof = ROOT_CID + date_string
4 input_hash  = hashlib.sha256(input_data_proof.encode('utf-8')).hexdigest()
5 input_hash_bytes = hashlib.sha256(input_data_proof.encode('utf-8')).digest()
```

At the conclusion of processing, the output values are written to a CSV file, for which a hash is calculated. The decision to hash the file rather than the values directly is deliberate. The CSV file will be uploaded to IPFS, making it easier to reproduce independently of any external verifier.

```
1  with open(csv_filename, "rb") as f:
2  file_bytes = f.read()
3 output_hash_bytes = hashlib.sha256(file_bytes).digest()
```

```
file_hash = hashlib.sha256(file_bytes).hexdigest()
binding_hash = input_hash_bytes + output_hash_bytes
```

The two values obtained, both 32 bytes, are concatenated into a single binding hash, which is inserted into the Report Data field of the quote, whose maximum capacity is 64 bytes. As previously discussed, a TEE system guarantees that the execution of a given program, identified by its MRENCLAVE value, has taken place correctly. There are no further guarantees from a semantic standpoint or regarding how the Report Data field was populated. Nothing would prevent a malicious actor from running the algorithm on different data than those declared, then inserting ad hoc hashes into the Report Data, still obtaining a hardware valid proof. The quote attests that the binary executed within the SGX enclave and that those 64 bytes were present in the Report Data, but cannot independently guarantee that those 64 bytes were constructed according to the intended logic.

To close this gap, an independent external auditor would need to perform a series of checks. The first step involves inspecting the algorithm's source code, verifying its logic, compiling it and publishing the expected MRENCLAVE value. The auditor should also certify that the Report Data field is populated with the correct hash concatenation rather than arbitrary values. Only the presence of such an entity would make the chain of trust complete and independent from the operator. The overall trust level in the architecture would be significantly reduced, though at the cost of sharing the source code with third parties, partially compromising its confidentiality. At the end of execution, in addition to the binary Remote Quote, a JSON file is generated. It contains all relevant process values in human-readable form and is intended to assist the verifier during the audit process. It should be noted that this file does NOT constitute a proof, as it is neither signed nor verifiable.

```
   metadata = {
        "execution_date": date_string,
        "input_cid": ROOT_CID,
        "input_proof_string": input_data_proof,
        "input_proof_hash": input_hash,
        "output_hash": file_hash,
        "binding_hash_hex": binding_hash.hex()
    }
```

4.4 On-Chain Logic and State Management

This section analyzes the role of blockchain technology within the Structural Health Monitoring (SHM) pipeline. The infrastructure acts as a public and immutable ledger, ensuring full visibility and transparency of the structural status after each monitoring iteration. As the final layer of the architecture, the blockchain serves a consolidating function: it does not merely store the classification result, but acts as an anchor point for all preceding off-chain phases. Leveraging the immutability properties of the ledger, the IPFS CIDs of the datasets used and the execution proofs, such as the SGX quote generated in the Trusted Execution Environment, are uploaded on-chain. This allows an external verifier to independently reconstruct and validate the entire process. In summary, this phase collects and crystallizes the information generated in the previous steps, transforming an isolated computational analysis into a certified, auditable and time-persistent record.

4.4.1 Smart Contract Architecture

The smart contract represents the core of the on-chain component. The code was developed in Solidity and is responsible for processing the values received from the local computation and updating the infrastructure risk profile. The variables employed in the code are the following:

- **input_data**: stores the CID of the raw data recorded by the sensors
- **lastCID**: stores the CID of the data cleaning algorithm output
- **proof_execution**: stores the CID of the Remote Quote generated by the TEE
- **counter**: counter that quantifies the number of values outside the threshold
- **status**: saves as a string, based on the counter value, the health status of the infrastructure

These variables are permanently stored on the blockchain, but get overwritten every time the contract is called, so only the most recent execution is kept. To avoid losing track of past analyses, the StatusUpdated event was introduced. The emission of this event within the main function generates immutable logs in the transaction. Unlike state variables, logs are not overwritten and serve as a historical register of the entire SHM pipeline. This mechanism ensures full system auditability, enabling an external verifier to retrieve the CIDs and health statuses of each individual analysis performed over time, while simultaneously optimising on-chain storage costs. The contract's operational core is represented by the threshold function,

which is designed to receive as arguments an array of integers corresponding to the scaled Damage Index (DI) and the related IPFS pointers (input, output, and proof). The function operates deterministically through an iterative loop that processes each element of the array: any value greater than or equal to zero is interpreted as a structural anomaly detected by the sensors and increments an internal counter. Once all values have been processed, the contract classifies the infrastructure health status according to the counter value. The possible infrastructure states are as follows:

- **false**: the positive value counter is equal to zero, indicating no damage
- **check**: anomalies are present but remain contained (between one and three values), indicating that an inspection of the structure is warranted.
- **alert**: The number of out-of-threshold values exceeds three, signalling a potentially damaged infrastructure.

Once this process is complete, the contract updates the state variables and emits the StatusUpdated event. This operation permanently anchors the values to the blockchain, making them immutable, transparent and publicly verifiable.

```
for (uint256 i = 0; i < values.length; i++) {
    if (values[i] >= 0) {
        count++;
    }
}

counter = count;
lastCID = outputCID;
proof_execution = proofCID;
input_data = inputCID;

if (count == 0) {
    status = "false";
} else if (count >= 1 && count <= 3) {
    status = "check";
} else {
    status = "alert";
}
```

To minimize smart contract deployment and execution overhead, on-chain operations were strictly limited. Cryptographic validation of the remote attestation was therefore not included as a blocking requirement for contract execution. While Solidity technically supports on-chain cryptographic verification, the computational cost of processing an SGX proof directly within the EVM (Ethereum Virtual

Machine) would be prohibitive and poorly scalable for a continuous monitoring system. A traditional on-chain DCAP verification costs approximately 4 million gas units [61], making the operation economically unsustainable on public networks. However, recent advances in ZKP are opening new possibilities. It is now feasible to generate off-chain a zero-knowledge proof attesting to the correct verification of the SGX remote quote. In this architecture, the oracle no longer submits the raw SGX proof, but rather a ZK proof confirming the attestation's validity. On-chain verification of a ZK proof is significantly cheaper, with gas consumption dropping to approximately 200,000–300,000 units.

$$\text{Verification}_{SGX} = \underbrace{4{,}000{,}000 \times 0.79 \times 10^{-9}}_{\text{Gas used} \times \text{Gas price (ETH)}} \times \underbrace{2268.58}_{\text{ETH price (\$)}} \approx \$7.168^{1}$$

$$\text{Verification}_{ZKP} = \underbrace{300{,}000 \times 0.79 \times 10^{-9}}_{\text{Gas used} \times \text{Gas price (ETH)}} \times \underbrace{2268.58}_{\text{ETH price (\$)}} \approx \$0.537$$

The on-chain computational cost is closely linked to the dynamics of the Gas Price, a parameter characterised by high intrinsic volatility. Its value fluctuates according to multiple factors, including global transaction volume on the Ethereum network and the market price of the cryptocurrency itself. Such variations can produce sudden and substantial swings in operational costs. In a peak scenario, such as that recorded on 10 October 2025, the costs would have been:

$$\text{Verification}_{SGX} = \underbrace{4{,}000{,}000 \times 18.20 \times 10^{-9}}_{\text{Gas used} \times \text{Gas price (ETH)}} \times \underbrace{3837}_{\text{ETH price (\$)}} \approx \$\mathbf{279.33}$$

$$\text{Verification}_{ZKP} = \underbrace{300{,}000 \times 18.20 \times 10^{-9}}_{\text{Gas used} \times \text{Gas price (ETH)}} \times \underbrace{3837}_{\text{ETH price (\$)}} \approx \$\mathbf{20.95}^{2}$$

Due to this extreme volatility in execution costs, it was decided to minimize on-chain computation. The Smart Contract is executed at each oracle trigger, recording the CIDs and updating the infrastructure status without verifying the authenticity of the remote quote. On-chain verification could be integrated in future developments of the pipeline, particularly if new optimisations succeed in reducing the gas consumption required.

[1]Source: Etherscan with values as of 2026/02/01

[2]Source: Etherscan with values as of 2025/10/10

4.4.2 The Oracle Connector

The blockchain operates within an isolated, deterministic system, which is incapable of interacting with the external environment. For this reason, smart contracts cannot make external calls such as downloading files from IPFS or querying databases. Oracles act as intermediaries, enabling communication between off-chain and on-chain environments. In this case study, the oracle was designed specifically for the contract, transmitting the same variables previously defined in Solidity. For this reason, despite the oracle intervention occurring before the blockchain update in the operational flow, its architecture is discussed in this section, following the definition of the on-chain control logic. The oracle consists of a Python script leveraging the web3.py library. Its operational flow is structured into three fundamental steps:

1. **Initialization and data retrieval**: the script loads environment variables from a configuration file containing the contract address and access keys. It then retrieves the CID of the raw sensor data, the CID of the file generated by the SHM algorithm, and the CID of the cryptographic proof produced by Intel SGX along with the metadata file.

2. **IPFS interaction and data formatting**: the oracle downloads the SHM algorithm output file using the previously retrieved CID. The extracted values are then converted into an array to ensure compatibility with the Smart Contract.

3. **On-Chain execution**: in the final step, the script establishes a connection with a Sepolia blockchain node. The values previously fetched from IPFS are submitted on-chain through a transaction signed with the associated private key

The utilisation of a centralised software connector introduces a number of vulnerabilities. Since the script runs in a standard environment without any cryptographic proof, it is susceptible to manipulation and represents a potential attack surface. The Smart Contract performs no validation on the received values, accepting them by default. A targeted attack on the script could alter the value array prior to transmission, causing the blockchain to record a false alarm or conceal critical damage. The entire architecture relies on unique identifiers—specifically CIDs and hashes—associated with the input data, output results, and the execution proof. Any tampering with the values submitted to the contract would produce a mathematical mismatch. An external auditor, upon downloading the files from IPFS, would immediately notice that the hashes recomputed over the data do not match the binding hash contained in the Report Data field of the Remote Quote. Although this step is not independently verifiable, any potential data manipulation

would be detected at the subsequent verification stage, making cryptographic verification of the oracle execution unnecessary.

Chapter 5

Practical Implementation and Results Analysis

This chapter aims to demonstrate the practical implementation of the entire SHM pipeline, examining complications, execution times and associated costs. The complete process is described step by step, and a transparency verification of the architecture is conducted by simulating the operations of an external auditor. While the previous chapters presented the theoretical foundations of the architecture and its components, the following sections focus on the concrete implementation aspects. A performance overhead analysis resulting from the adoption of TEE is carried out, alongside a cost and timing estimation. Through an examination of the on-chain component, the trade-offs between security, transparency and scalability are discussed, demonstrating in practice the design decisions taken during the construction of the pipeline.

5.1 Hardware and Software Configuration

The first fundamental step was selecting hardware with enough computational power for the algorithm and, above all, with native TEE support. A Microsoft Azure Virtual Machine with a DC8s v3 instance was chosen for this purpose. The machine runs an Intel(R) Xeon(R) Platinum 8370C server processor, compatible with Intel SGX, and has 64 GB of RAM and 32 GB of dedicated EPC memory, which allows processing large volumes of data. The DC4s v3 instance was initially used, with 32GB of RAM and 16GB of EPC memory, but it proved insufficient to run the VMD phase, generating an ArrayMemoryError due to the hardware EPC limits.

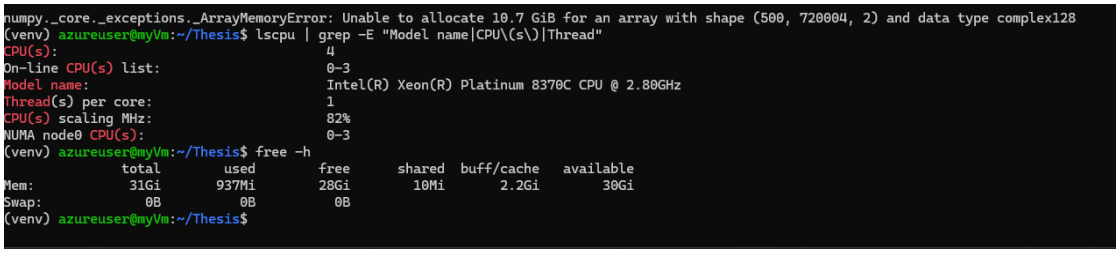

Figure 5.1: ArrayMemoryError on a Standard DC4s v3 instance, showing the failure to allocate 10.7 GiB due to Intel SGX hardware memory (EPC) limits.

The image shows how standard RAM plays a marginal role in processes running inside a Trusted Execution Environment. All operations take place within the EPC, which is therefore the key parameter to consider when selecting the VM instance. Another important factor is the physical location of the datacenter: the greater the distance from Microsoft's hardware, the higher the latency. Each region also has different hourly pricing, so the choice requires balancing performance and cost based on the specific use case. Lower latency means faster execution, but the nearest datacenter may not be the cheapest option. For this case study, the Italy North region was selected, as it is the closest to the execution site and among the most cost-effective options available. The following table compares the hourly pricing of the DC8s v3 instance across different regions:

Table 5.1: Azure DC8s v3 hourly pricing (Linux PAYG, Mar 2026) [62]

Region	Location	Price (USD/hr)
Italy North	Milan	$0.448
East US	Virginia	$0.768
North Europe	Dublin	$0.856
West Europe	Amsterdam	$0.920

The final configuration of the Microsoft Azure Virtual Machine is reported below:

Table 5.2: Specifications of the VM used for TEE deployment.

Parameter	Value
Cloud platform	Microsoft Azure
Region	Italy North
VM type	Standard DC8s v3
Processor	Intel(R) Xeon(R) Platinum 8370C
Architecture	Ice Lake
Base frequency	2.8 GHz
vCPU	8 cores
RAM	64 GB
TEE technology	Intel SGX v2
EPC memory	Up to 32 GB
Operating system	Ubuntu 24.04 LTS
Hourly cost	$0.448

Once the VM was configured, the necessary software stack for the algorithm execution had to be installed. The only fundamental constraint at this stage is the use of a Linux kernel with DCAP support, as it is the only one compatible with Gramine. Another aspect to consider is consistency across executions. Python versions can change over time, and with them the individual libraries. To maximize compatibility, it is good practice to run the pipeline with fixed software versions, avoiding discrepancies in the results. For this reason, the entire Python environment was managed with all dependencies listed in a dedicated requirements file.

Table 5.3: Software stack used for the SHM pipeline execution.

Component	**Version**
Operating System	Ubuntu 24.04.3 LTS
Python	3.12.3
Gramine	1.9
numpy	2.4.1
scipy	1.17.0
scikit-learn	1.8.0
vmdpy	0.2

5.2 IPFS Integration

The sensor data recorded on the bridge are uploaded to IPFS, generating a unique Content Identifier. It is important to specify that a computer on which IPFS has been downloaded becomes a node forming part of the decentralized network. If no other node has copied the file, it may be removed through garbage collection or become unavailable when the local device is offline. To address this issue and ensure continuous data availability, dedicated pinning services are used. These are platforms with hundreds of nodes that keep files permanently accessible, regardless of the local node status. For the study project, Pinata was utilized because it has a free plan that allows for constant pinning of 500 files up to 1GB of storage. In a larger scale deployment, the adoption of a dedicated IPFS node network or a paid pinning service should be evaluated based on data volume and availability requirements.

Table 5.4: IPFS upload statistics for the Yonghe Bridge dataset.

Parameter	**Value**
Local file size	536.67 MB
IPFS size	536.77 MB
Upload time	$\approx$ 2 min
Root CID	`bafybeige56soo3pgpzdscmi5ymd7birbordgusafxj2egnsdujiec3wpxi`
Pinning service	Pinata

The storage size is nearly identical to the original file. The small difference is due to the Merkle DAG structure, which adds metadata for each 256 kB block generated during the chunking process.

5.3 Algorithm execution via TEE

Once the input data were uploaded to IPFS and the corresponding CID obtained, the SHM algorithm was executed. The Microsoft Azure DC8s v3 VM was used with Gramine as the LibOS to ensure execution within a trusted environment. The configuration and signing of the manifest occurred as explained in the previous chapters. The Python code remained almost entirely unaltered from the original version, with the exception of an additional function for generating the Remote Quote. To keep the MRENCLAVE value constant across executions, the input CID was parameterized through a system environment file read by the program

at runtime. This approach allows system parameters to be updated without touching the source code, avoiding any alteration of the MRENCLAVE value and guaranteeing coherent remote attestation for all subsequent executions. At the end of the execution, three files were generated:

1. **output_2026-02-01.csv**: it contains the algorithm output, which includes the Damage Index (DI) values.

2. **quote_2026-02-01.bin**: it is the binary file containing the SGX Remote Quote, which includes key fields for the entire architecture such as MRENCLAVE, MRSIGNER and the Report Data field.

3. **metadata.json** : JSON file containing the values stored in Remote Quote in a readable and organized format

This last generated file serves only as a preliminary overview of the data and holds no validity in the verification or transparency process. To estimate the overhead introduced by execution within a trusted environment, the algorithm was run twice under identical conditions: once with TEE and once natively.

Table 5.5: Performance overhead introduced by TEE isolation: native execution vs SGX via Gramine.

Execution Mode	**Real Time**	**User Time**	**Sys Time**
Native (no TEE)	7m 28s	54m 37s	0m 24s
SGX via Gramine	10m 28s	57m 16s	1m 12s
Overhead	+40%	+5%	+200%

The values indicate that the most significant percentage increase is observed in system time, with an increase of 200%. This is due to the OCALLs introduced by Gramine, which handle all calls from code inside the enclave to the untrusted host operating system. User time shows minimal variation, confirming the high computational load sustained by the CPU throughout execution. The overall overhead based on real time is approximately 40%, confirming the feasibility of TEE for programs with an intensive workload. Once the algorithm is complete, the artifacts are published on IPFS and pinned via Pinata, generating the corresponding Content Identifiers.

Table 5.6: IPFS Content Identifiers generated after TEE execution.

File	CID
Algorithm output (CSV)	`bafkreifku4pevurgpal7b3o27l caeyxfffmpv75zqx77gkyjlhuox ppod4`
Proof and metadata folder	`bafybeicbjcgy6swfnheouxqrsd 6vzvuyqkonexwxybqrkvni5mfkw ysj54`

The CIDs generated by the algorithm, in addition to those of the raw files recorded by the sensors, complete the IPFS references that the oracle will subsequently submit on-chain.

5.4 Oracle and On-Chain Settlement

The final stage of the architecture concerns the on-chain data submission, the Smart Contract execution and the infrastructure status update. The oracle, through a system file, reads the CID values calculated in the preceding steps and sends and signs the transaction to the Sepolia node. It also extracts the output values produced by the SHM algorithm and stores them in an array to facilitate EVM processing. The trigger for the Solidity code therefore consists of the oracle execution and the submission of the transaction containing the required data. Once this phase is complete, the bridge status can be monitored on the blockchain through the threshold function described in Chapter 4. In the case study, the smart contract counter detected 4 Damage Index values exceeding the threshold, classifying the infrastructure as potentially damaged. The analyzed data referred to the month of August, a period in which the Yonghe Bridge presented documented structural damage, confirming the correctness of the obtained result.

Parameter	**Value**
Contract address	`0x7EB8a545F9D22f0851C31bD1e2c6eeFa1aC15e89`
First transaction hash	`0xca7ce754a8ed66fd958c0fdc26b3fb5672b55ff44c7c86e193717d26447ee831`
Second transaction hash	`0x390bbaa08fc658d7a04a0198989e5815e42bf018e75008693cdcba4bfdb9cdea`

Table 5.7: Contract address and transaction hashes

Two transactions were submitted during the case study: the first served as a test execution to initialize the state variables, while the second carried the actual SHM output and produced the final infrastructure classification. The first one shows a gas consumption approximately three times higher than subsequent ones. This is due to the SSTORE operation and how the EVM handles state variable writes. When a contract is executed for the first time, all state variables contain a default value of zero. Writing a non-zero value into an empty storage slot costs around 20,000 gas per variable. From the second transaction onwards, the variables already hold non-zero values, so only an overwrite is required, costing around 5,000 gas per operation. The costs to consider for this phase are those related to the one-time contract deployment and those associated with each individual transaction.

Operation	**Gas Used**	**Cost (ETH)**	**Cost (USD)**
Contract deployment	805,162	0.000636 ETH	$1.44
First transaction	307,657	0.000243 ETH	$0.55
Subsequent transactions	96,650	0.000076 ETH	$0.17

Table 5.8: Gas consumption and estimated costs (Gas price: 0.79 Gwei average, ETH: $2268.58, Feb 01 2026). Source: Etherscan.

To estimate the overall cost of the pipeline, the VM rental cost and the on-chain transaction cost must be combined. The IPFS pinning service used in the case study was the free tier, which comes with storage limitations. In a larger scale deployment, a paid service or a dedicated network of IPFS nodes should be considered. Assuming a weekly algorithm execution with a duration of less than one hour, the annual cost can be estimated:

$$\text{Annual cost} = \underbrace{0.448 \times 52}_{\text{Cloud VM}} + \underbrace{1.44}_{\text{Deployment}} + \underbrace{0.17 \times 52}_{\text{Transactions}} = \$33.57$$

The figure obtained demonstrates the scalability and economic sustainability of the pipeline presented. The implementation of transparency mechanisms and cryptographic proofs adds a minimal cost compared to the benefits gained in terms of auditability, traceability and reduced trust requirements across the entire architecture.

5.5 Transparency Audit Simulation

The purpose of the architecture is to increase the transparency and, above all, the verifiability of an SHM system. The only information required to initiate the audit process is the Smart Contract address. By querying the blockchain, it is possible to directly read the transaction logs where the CIDs of the sensor data, the output of the SHM algorithm, and the cryptographic proof are stored. These Content Identifiers can then be used to retrieve the associated files. The first and most critical part of the verification concerns the Remote Quote, which attests to the correct execution of the algorithm. There are several approaches to checking the validity of this file, each with different trade-offs. One option is to rely on cloud-based solutions such as Microsoft Azure Attestation (MAA), which drastically reduces architectural complexity by delegating the validation process to a managed service. This approach receives the Quote, validates it against specific security policies, and issues a JWT (JSON Web Token) that can be easily inspected by any application. This is the simplest and most straightforward approach, though it introduces an additional trust assumption, the cloud platform itself. The alternative is a local and independent verification process. Using native DCAP libraries or tools integrated directly into Gramine, it is possible to query Intel's servers to confirm the validity of the proof. In this study, a Python script was implemented using the dcap_qvl library [63] to validate the Quote generated by Intel SGX. In addition to verifying the correctness of the hardware state (TCB), the code extracts the fundamental execution parameters in plain text. The output provides immediate visibility of the MRENCLAVE, MRSIGNER, and Report Data fields, thereby ensuring definitive cryptographic confirmation of the TEE environment's authenticity. The second verification concerns the binding hash. The verifier retrieves the raw dataset CID from the blockchain and concatenates it with the execution date, which can be obtained from the metadata.json file. Applying a SHA-256 function to this string produces the first of the two hashes. The verifier then downloads the output CSV file using its CID and computes its SHA-256 hash. Concatenating the two 32-byte values yields exactly the content of the Report

Data field in the Remote Quote. A match constitutes cryptographic proof that the enclave processed precisely the declared dataset and produced precisely that output, with no possibility of substitution. In the real verification run, the values obtained are those shown in the table below.

Parameter	**Value**
Input CID (from blockchain)	`bafybeige56soo3pgpzdscmi5ymd7birbordgusafxj2egnsdujiec3wpxi`
Execution date (from metadata.json)	`2026-02-01`
Concatenated string	`bafybeige56soo3pgpzdscmi5ymd7birbordgusafxj2egnsdujiec3wpxi2026-02-01`
Input proof hash (SHA-256)	`de16d9d0aa223823f8f82a0c266a19d52b443763f8c4611f2bb3f8cf26fc5315`
Output CID (from blockchain)	`bafkreifku4pevurgpal7b3o27lcaeyxfffmpv75zqx77gkyjlhuoxppod4`
Output hash (SHA-256)	`aaa71e4ad2267817f0eddafac40262e52958faffb985fff32b0959e8ebbdee1f`

Table 5.9: Binding hash computation performed by the verifier.

Concatenating the two strings yields the expected binding hash value, which must then be compared against the Report Data field. After running the Python script, the values read from the remote quote are:

Parameter	Value
TCB Status	`SWHardeningNeeded`
Advisory ID	`INTEL-SA-00615`
MRENCLAVE	`918b521753002ddc91f6881dc1d485015a012d5ca31d506c476040ad8f97b601`
MRSIGNER	`205e2b2e1cf69c19abdb73593ead8ef1d0021e848cba26e386cb13b67e63954a`
Report Data	`de16d9d0aa223823f8f82a0c266a19d52b443763f8c4611f2bb3f8cf26fc5315aaa71e4ad2267817f0eddafac40262e52958faffb985fff32b0959e8ebbdee1f`

Table 5.10: Remote Quote fields extracted via dcap_qvl verification script.

The correspondence confirms that the enclave processed the declared dataset exactly as specified, producing the stated output. The status returned by the DCAP verification is SWHardeningNeeded, associated with advisory INTEL-SA-00615. This result does not indicate any enclave compromise, nor does it invalidate the verification performed. It means that the processor used is subject to known side-channel vulnerabilities for which Intel recommends the adoption of additional software mitigations. The final step in the audit process is to verify the Smart Contract input data. The on-chain code is responsible for processing the values derived from the SHM algorithm output and, consequently, updating the infrastructure status. This process can be verified by opening the CSV file containing the Damage Index values and comparing it with the values read by the Contract via Sepolia. The simulation demonstrates that the entire pipeline can be verified publicly and independently by any auditor, with no barriers arising from specific hardware requirements or privileged access to data. The level of trust required in the operator is drastically reduced, transforming the system from opaque to transparent.

5.6 Critical Discussion and Future Developments

The system described above significantly reduces the required level of trust, but presents several limitations that must be acknowledged. The most critical point

is the oracle. The Python script responsible for reading the CIDs, extracting the output values and submitting the transaction operates in an unverified environment. There are no guarantees that the values sent on-chain actually correspond to the output of the algorithm executed within the TEE. Any manipulation of these values would still be detectable by an external auditor during the verification phase, but would not be prevented in real time. The system, which does not include automatic checks, would therefore not be able to identify this type of malfunction, placing complete trust in the verifier. The implementation of a decentralised oracle, such as Chainlink, would eliminate this critical issue by adopting a system distributed across multiple nodes, increasing overall transparency. The Pinata pinning service was used in its free tier, which is sufficient for the case study but not scalable in a real operational context. A large-scale deployment would require a paid plan or, alternatively, the management of a dedicated IPFS node network. Running a self-managed network would remove the dependency on an external service, significantly enhancing the system's overall resilience.

The adoption of a Trusted Execution Environment (TEE)-based approach made it possible to execute the SHM algorithm, characterised by intensive mathematical operations, while maintaining acceptable performance overhead and generating cryptographic proof capable of validating its execution. However, an intrinsic limitation of this technology is the absence of a direct mathematical link between the input data, the program logic and the output values. This critical issue currently makes the involvement of a qualified external auditor indispensable. To protect the intellectual property of the software, the source code is not made public but disclosed exclusively to this certifying body. The auditor's role is to privately inspect the code, validate its semantics and the correct population of the Report Data field, and then publish the MRENCLAVE deemed valid on-chain. While this approach ensures the functioning of the system, it introduces a new trust node, the auditor, without which the end user has no guarantee regarding the actual correctness of the binary running inside the enclave. To address this limitation, ZKP systems would need to be integrated. Unlike hardware attestation, these cryptographically bind input, program logic and output, producing a trustless proof without relying on any external authority. Once generating such proofs becomes computationally feasible for floating-point workloads like those used in this study, the need for auditors and trusted hardware can be eliminated entirely. The system also lends itself to targeted architectural enhancements aimed at optimising the pipeline and making it more scalable and suited to real-world deployment. While these additions are not strictly necessary to ensure the transparency of the base model, their adoption would meaningfully improve the efficiency, privacy and responsiveness of the infrastructure. The sensor readings collected from the bridge are currently uploaded to IPFS in plaintext, making them publicly visible. While this approach ensures maximum inspectability, it may not be compatible with

the security policies of strategic infrastructures. A future development could therefore involve the introduction of asymmetric encryption. This would allow the data to be encrypted before being uploaded to the decentralised network, preserving the benefits of IPFS while adding a stronger security layer. The SGX enclave, leveraging secure key management mechanisms, could then decrypt the data exclusively within its protected environment during algorithm execution, keeping the raw measurements hidden from the public without compromising the integrity proof of the computation.

In terms of code execution, using Gramine requires careful configuration of the host environment, making the system very sensitive to changes in the operating system or Python libraries. The use of Confidential Containers would facilitate the creation of Docker images encompassing the operating system, dependencies, and algorithm. This would fully standardise the architecture, producing a system that can run natively on any cloud server or local machine equipped with SGX hardware, minimising setup time and version incompatibility risks. Finally, the role of the blockchain could be expanded beyond its function as an immutable ledger. In the event of an alert status, the contract could automatically trigger notifications to external systems or initiate escalation procedures towards maintenance personnel, transforming the system from a passive monitoring tool into an active component of infrastructure management.

Chapter 6

Conclusion

The introduction of SHM systems has profoundly changed the way infrastructures are monitored. Installing a sensor network enables continuous data collection, ensuring constant surveillance of the structure. This approach overcomes the limitations of visual inspections and allows damage to be identified before it becomes visible to the human eye. The widespread adoption of these systems has helped reduce costs compared to traditional monitoring methods, improving safety and extending the operational lifespan of the infrastructure. Conventional SHM systems still suffer from limitations related to the transparency and auditability of the entire pipeline. Sensor data is collected, processed and interpreted within proprietary infrastructures entirely controlled by the operator, with no mechanism allowing external parties to verify its authenticity. This situation gives rise to a Trust Gap, a condition in which institutions, citizens and auditors are forced to place their trust in a single entity, leaving open the possibility of result manipulation.

The aim of this work was therefore to propose an alternative that, while preserving the advantages of SHM systems, would guarantee a level of control and transparency sufficient to confirm correct execution and eliminate the information asymmetry. Three macro-phases were identified: data measurement and storage, algorithm execution, and Damage Index computation. Each of these phases relies on complementary technologies that make it possible to verify correct execution. Sensor data storage is handled via the InterPlanetary File System (IPFS), a decentralised network that makes the values publicly accessible, associates them with a unique identifier and, crucially, eliminates the risk of a single point of failure. The algorithm runs inside an Intel SGX Trusted Execution Environment, generating a tamper-resistant hardware proof that cryptographically links input and output via the binding hash. The resulting values are then processed by a Smart Contract on the Ethereum network, which computes the infrastructure status and stores it immutably on the blockchain.

The system was tested on a real case study using datasets from the Yonghe Bridge,

validating the entire pipeline in a practical context, demonstrating the feasibility of the approach and estimating realistic costs. The most technically complex aspect was the off-chain attestation of algorithm execution. Intel SGX, running through Gramine LibOS, made it possible to avoid any modifications to the native code, considerably simplifying the procedure. Performance analysis revealed a TEE overhead of approximately 40%, primarily due to the management of system calls between the trusted and untrusted environments. This is a modest and acceptable difference, confirming the viability of the TEE approach for computationally intensive workloads. The main constraint of this approach is the need for Intel SGX-compatible hardware: in the case study, an Azure DC8s v3 VM in the Italy North region was used, at a cost of $0.448 per hour.

The IPFS integration demonstrated reliable data availability and consistency, with upload and pinning operations completed within the expected timeframes. Damage Index analysis and the consequent infrastructure status update are performed on-chain via a Smart Contract. The analyses were carried out on the Sepolia test network, but with costs reflecting the Ethereum mainnet. Each transaction consumes approximately 97,000 gas units, corresponding to roughly $0.17 per execution, confirming the economic sustainability of the approach for periodic monitoring use cases. The audit simulation in section 5.5 demonstrated that an independent third-party verifier, using exclusively publicly available data on the blockchain and IPFS, was able to cryptographically reconstruct and verify the entire pipeline execution, from the raw sensor input through to the final Damage Index, without requiring any collaboration from the system operator. The system as a whole demonstrates the feasibility of a transparent pipeline, but also presents certain limitations. TEE systems offer no guarantees regarding program semantics or the link between input and output. For this reason, a trusted third party must validate the logic and confirm the structure, introducing an additional entity into the system that requires trust.

On-chain data submission is handled by an oracle implemented as a Python script. This design introduces a centralised element into an otherwise decentralised system, representing a potential single point of failure and a residual trust assumption. The system performs no validation on the data, which is only checked manually at a later stage by the auditor. Furthermore, data persistence on IPFS depends on active pinning by a dedicated node: should this node become unavailable, the Content Identifiers registered on-chain would no longer be resolvable. Future developments are therefore focused on reducing these issues. Replacing the Python oracle with a decentralised network such as Chainlink would eliminate the centralised element, distributing the responsibility for data submission across multiple independent nodes. On the storage side, integrating a decentralised pinning service would guarantee long-term data persistence without relying on a single operator-controlled node. A further development concerns the on-chain cryptographic verification of

the Remote Quote. Currently the system implements no automatic checks, leaving the detection of erroneous executions entirely to the auditor. Embedding this verification in the Smart Contract would make the system reactive, automatically rejecting non-compliant transactions without any manual intervention. Finally, the maturation of Zero-Knowledge Proof systems represents the most significant frontier: unlike TEEs, which attest exclusively to the integrity of the execution environment without guaranteeing program semantics, ZKPs cryptographically bind input, computational logic and output, eliminating the need for an external auditor and making the system fully trustless. The following case study was also the subject of an article that will be presented at IABMAS 2026. All programs used are available for consultation at a GitHub repository [1].

In conclusion, this work demonstrates that implementing a transparent and independently verifiable SHM system is both technically feasible and economically sustainable, removing the need to place unconditional trust in the infrastructure operator. The proposed system represents a concrete, deployable step towards a new generation of infrastructure monitoring solutions, capable of meeting growing demands for accountability, transparency and public safety.

[1] https://github.com/GiorgioMarengo/Thesis

Bibliography

[1] Hoong Chen Teo, Alex Mark Lechner, Grant W. Walton, Faith Ka Shun Chan, Ali Cheshmehzangi, May Tan-Mullins, Hing Kai Chan, Troy Sternberg, and Ahimsa Campos-Arceiz. «Environmental Impacts of Infrastructure Development under the Belt and Road Initiative». In: *Environments* 6.6 (2019). ISSN: 2076-3298. DOI: 10.3390/environments6060072. URL: https://www.mdpi.com/2076-3298/6/6/72 (cit. on p. 3).

[2] Zili Zhang, Xiangyang Li, and Hengyun Li. «A quantitative approach for assessing the critical nodal and linear elements of a railway infrastructure». In: *International Journal of Critical Infrastructure Protection* 8 (2015), pp. 3–15. ISSN: 1874-5482. DOI: https://doi.org/10.1016/j.ijcip.2014.11.001. URL: https://www.sciencedirect.com/science/article/pii/S1874548214000651 (cit. on p. 4).

[3] *Directive (EU) 2022/2557 of the European Parliament and of the Council of 14 December 2022 on the resilience of critical entities and repealing Council Directive 2008/114/EC*. Official Journal of the European Union, L 333/164. Art. 2(4). Dec. 2022. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32022L2557 (cit. on p. 5).

[4] United Nations Office for Disaster Risk Reduction. *Definition: Critical infrastructure*. Accessed: 2025-09-25. 2017. URL: https://www.undrr.org/terminology/critical-infrastructure (cit. on p. 5).

[5] The White House. *National Security Memorandum/NSM-22: Critical Infrastructure Security and Resilience*. Accessed: 2025-10-30. Apr. 2024. URL: https://bidenwhitehouse.archives.gov/briefing-room/presidential-actions/2024/04/30/national-security-memorandum-on-critical-infrastructure-security-and-resilience/ (cit. on p. 5).

[6] Giacomo Buffarini, Paolo Clemente, Sonia Giovinazzi, Chiara Ormando, Maurizio Pollino, and Vittorio Rosato. «Preventing and Managing Risks Induced by Natural Hazards to Critical Infrastructures». In: *Infrastructures* 7.6 (2022). ISSN: 2412-3811. DOI: 10.3390/infrastructures7060076. URL: https://www.mdpi.com/2412-3811/7/6/76 (cit. on p. 5).

[7] S.M. Rinaldi, James Peerenboom, and T.K. Kelly. «Identifying, understanding, and analyzing critical infrastructure interdependencies». In: *Control Systems, IEEE* 21 (Jan. 2002), pp. 11–25. DOI: 10.1109/37.969131 (cit. on p. 5).

[8] RFI. «Rete Ferroviaria Italiana S.p.A 2018 ANNUAL REPORT». In: (2018), p. 180. URL: https://www.rfi.it/content/dam/rfi/rfi_en/about-us/bilancio/RFI%2031-12-2018%20Annual%20Report.pdf (cit. on p. 6).

[9] Assoporti. «CROLLO PONTE MORANDI: PER L'AUTOTRASPORTO DANNO DI OLTRE 116 MILIONI DI EURO IN QUASI DUE MESI». In: (2018), p. 6. URL: https://www.assoporti.it/media/3655/analisi-impatto-crollo-ponte-morandi-per-lautotrasportodocx.pdf (cit. on p. 6).

[10] Claudio Ferrari and Marta Santagata. «Vulnerability and robustness of interdependent transport networks in north-western Italy». In: *European Transport Research Review* 15.1 (2023), p. 6. DOI: https://doi.org/10.1186/s12544-023-00580-7 (cit. on p. 6).

[11] Luca Capacci, Fabio Biondini, and Dan M. Frangopol. «Resilience of aging structures and infrastructure systems with emphasis on seismic resilience of bridges and road networks: Review». In: *Resilient Cities and Structures* 1.2 (2022), pp. 23–41. ISSN: 2772-7416. DOI: https://doi.org/10.1016/j.rcns.2022.05.001. URL: https://www.sciencedirect.com/science/article/pii/S2772741622000205 (cit. on p. 6).

[12] World Class Maintenance and FME. *Smart Infrastructure Maintenance – a Necessity and Opportunity for Europe*. Tech. rep. Report presented to the Dutch Minister of Infrastructure and Water Management. World Class Maintenance, Dec. 2023. URL: https://www.worldclassmaintenance.com/politics/rapport-smart-infrastructure-maintenance-a-necessity-and-opportunity-for-europe-overhandigd-aan-minister-harbers/ (cit. on p. 6).

[13] S. Kelly. «Estimating economic loss from cascading infrastructure failure: a perspective on modelling interdependency». In: *Infrastructure Complexity* 2.1 (2015), p. 7. DOI: 10.1186/s40551-015-0010-y. URL: https://doi.org/10.1186/s40551-015-0010-y (cit. on p. 7).

[14] Zehra Irem Turksezer, Chiara Iacovino, Pier Francesco Giordano, and Maria Pina Limongelli. «Development and Implementation of Indicators to Assess Bridge Inspection Practices». In: *Journal of Construction Engineering and Management* 147.12 (2021), p. 04021165. DOI: 10.1061/(ASCE)CO.1943-7862.0002195. URL: https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29CO.1943-7862.0002195 (cit. on p. 7).

[15] Francesca Brighenti, Valeria Francesca Caspani, Giancarlo Costa, Pier Francesco Giordano, Maria Pina Limongelli, and Daniele Zonta. «Bridge management systems: A review on current practice in a digitizing world». In: *Engineering Structures* 321 (2024), p. 118971. ISSN: 0141-0296. DOI: https://doi.org/10.1016/j.engstruct.2024.118971. URL: https://www.sciencedirect.com/science/article/pii/S0141029624015335 (cit. on p. 7).

[16] S. Dorafshan and M. Maguire. «Bridge inspection: human performance, unmanned aerial systems and automation». In: *Journal of Civil Structural Health Monitoring* 8.3 (2018), pp. 443–476. DOI: 10.1007/s13349-018-0285-4. URL: https://doi.org/10.1007/s13349-018-0285-4 (cit. on p. 8).

[17] M. Moore, B. Phares, B. Graybeal, D. Rolander, and G. Washer. *Reliability of Visual Inspection for Highway Bridges, Volume I: Final Report*. Tech. rep. FHWA-RD-01-020. Research report. Federal Highway Administration, Turner-Fairbank Highway Research Center, June 2001. URL: https://www.fhwa.dot.gov/publications/research/nde/pdfs/01020a.pdf (cit. on p. 8).

[18] Christoph Kralovec and Martin Schagerl. «Review of Structural Health Monitoring Methods Regarding a Multi-Sensor Approach for Damage Assessment of Metal and Composite Structures». In: *Sensors (Basel)* 20.3 (Feb. 2020), p. 826. ISSN: 1424-8220. DOI: 10.3390/s20030826. URL: https://doi.org/10.3390/s20030826 (cit. on p. 8).

[19] Bangcheng Zhang, Yuheng Ren, Siming He, Zhi Gao, Bo Li, and Jingyuan Song. «A review of methods and applications in structural health monitoring (SHM) for bridges». In: *Measurement* 245 (2025), p. 116575. ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2024.116575. URL: https://www.sciencedirect.com/science/article/pii/S0263224124024606 (cit. on p. 9).

[20] Yixin Zhou, Zepeng Ma, and Lei Fu. «A Review of Key Signal Processing Techniques for Structural Health Monitoring: Highlighting Non-Parametric Time-Frequency Analysis, Adaptive Decomposition, and Deconvolution». In: *Algorithms* 18.6 (2025). ISSN: 1999-4893. DOI: 10.3390/a18060318. URL: https://www.mdpi.com/1999-4893/18/6/318 (cit. on p. 9).

[21] Consiglio Superiore dei Lavori Pubblici. *Linee Guida per la classificazione e gestione del rischio, la valutazione della sicurezza e il monitoraggio dei ponti esistenti*. Tech. rep. Accessed: 2025-10-30. Rome, Italy: Ministero delle Infrastrutture e dei Trasporti, May 2020. URL: https://www.mit.gov.it/sites/default/files/media/notizia/2020-05/1_Testo_Linee_Guida_ponti.pdf (cit. on p. 10).

[22] Emin Aktan, Ivan Bartoli, Branko Glišić, and Carlo Rainieri. «Lessons from Bridge Structural Health Monitoring (SHM) and Their Implications for the Development of Cyber-Physical Systems». In: *Infrastructures* 9.2 (2024). ISSN: 2412-3811. DOI: 10.3390/infrastructures9020030. URL: https://www.mdpi.com/2412-3811/9/2/30 (cit. on p. 10).

[23] Ali Mardanshahi, Abhilash Sreekumar, Xin Yang, Swarup Kumar Barman, and Dimitrios Chronopoulos. «Sensing Techniques for Structural Health Monitoring: A State-of-the-Art Review on Performance Criteria and New-Generation Technologies». In: *Sensors* 25.5 (2025). ISSN: 1424-8220. DOI: 10.3390/s25051424. URL: https://www.mdpi.com/1424-8220/25/5/1424 (cit. on p. 11).

[24] Giulio Mariniello, Chiara Gragnaniello, and Domenico Asprone. «Enhancing structural health monitoring data management and damage detection in digital twins with blockchain and smart contracts». In: *Automation in Construction* 180 (2025), p. 106558. ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2025.106558. URL: https://www.sciencedirect.com/science/article/pii/S0926580525005989 (cit. on p. 11).

[25] Yilin Chen. «Comparative Analysis of the Centralized and Decentralized Architecture of Cloud Computing in terms of Privacy Security». In: *Applied and Computational Engineering* 145 (Apr. 2025), pp. 51–56. DOI: 10.54254/2755-2721/2025.21867 (cit. on p. 13).

[26] Arif Sari and Murat Akkaya. «Fault Tolerance Mechanisms in Distributed Systems». In: *International Journal of Communications, Network and System Sciences* 8 (Dec. 2015), pp. 471–482. DOI: 10.4236/ijcns.2015.812042 (cit. on p. 13).

[27] Weiyu Zhong, Ce Yang, Wei Liang, Jiahong Cai, Lin Chen, Jing Liao, and Naixue Xiong. «Byzantine Fault-Tolerant Consensus Algorithms: A Survey». In: *Electronics* 12 (Sept. 2023), p. 3801. DOI: 10.3390/electronics12183801 (cit. on p. 14).

[28] Cong T. Nguyen, Dinh Thai Hoang, Diep N. Nguyen, Dusit Niyato, Huynh Tuong Nguyen, and Eryk Dutkiewicz. «Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities». In: *IEEE Access* 7 (2019), pp. 85727–85745. DOI: 10.1109/ACCESS.2019.2925010 (cit. on p. 14).

[29] Juan Benet. «IPFS - Content Addressed, Versioned, P2P File System». In: *CoRR* abs/1407.3561 (2014). arXiv: 1407.3561. URL: http://arxiv.org/abs/1407.3561 (cit. on p. 16).

[30] Justus Wendroth and Benedikt Jaeger. «A Brief Overview on HTTP». In: *Network* 59 (2022) (cit. on p. 17).

[31] Petar Maymounkov and David Mazières. «Kademlia: A Peer-to-Peer Information System Based on the XOR Metric». In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65. ISBN: 978-3-540-45748-0 (cit. on p. 18).

[32] Alfonso De la Rocha, David Dias, and Yiannis Psaras. «Accelerating content routing with bitswap: A multi-path file transfer protocol in ipfs and filecoin». In: *San Francisco, CA, USA (2021)* 11 (2021) (cit. on p. 18).

[33] Trinh Viet Doan, Yiannis Psaras, Jörg Ott, and Vaibhav Bajpai. «Toward Decentralized Cloud Storage With IPFS: Opportunities, Challenges, and Future Considerations». In: *IEEE Internet Computing* 26.6 (2022), pp. 7–15. DOI: 10.1109/MIC.2022.3209804 (cit. on p. 19).

[34] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. «Pinocchio: nearly practical verifiable computation». In: *Commun. ACM* 59.2 (Jan. 2016), pp. 103–112. ISSN: 0001-0782. DOI: 10.1145/2856449. URL: https://doi.org/10.1145/2856449 (cit. on p. 20).

[35] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. «Trusted Execution Environment: What It is, and What It is Not». In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. 2015, pp. 57–64. DOI: 10.1109/Trustcom.2015.357 (cit. on p. 22).

[36] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. «The knowledge complexity of interactive proof systems». In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. DOI: 10.1137/0218012 (cit. on p. 26).

[37] Manuel Blum, Paul Feldman, and Silvio Micali. «Non-interactive zero-knowledge and its applications». In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM. 1988, pp. 103–112. DOI: 10.1145/62212.62222 (cit. on p. 28).

[38] Ryan Lavin, Xuekai Liu, Hardhik Mohanty, Logan Norman, Giovanni Zaarour, and Bhaskar Krishnamachari. «A survey on the applications of zero-knowledge proofs». In: *arXiv preprint arXiv:2408.00243* (2024) (cit. on p. 28).

[39] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Scalable, transparent, and post-quantum secure computational integrity*. Cryptology ePrint Archive, Paper 2018/046. 2018. URL: https://eprint.iacr.org/2018/046 (cit. on p. 29).

[40] Aleksander Berentsen, Jeremias Lenzi, and Remo Nyffenegger. «A walk-through of a simple zk-stark proof». In: *Available at SSRN 4308637* (2022) (cit. on p. 30).

[41] Li Wei, Liang Peili, and Li Fei. «zk-STARKs based scheme for sealed auctions in chains». In: *IET Blockchain* 4.4 (2024), pp. 344–354 (cit. on p. 30).

[42] Ayush Nainwal, Atharva Kamble, and Nitin Awathare. «A Comparative Analysis of zk-SNARKs and zk-STARKs: Theory and Practice». In: *arXiv preprint arXiv:2512.10020* (2025) (cit. on p. 31).

[43] Satoshi Nakamoto. «Bitcoin: A peer-to-peer electronic cash system». In: *Available at SSRN 3440802* (2008) (cit. on p. 33).

[44] Luyao Zhang, Xinshi Ma, and Yulin Liu. «Sok: blockchain decentralization». In: *arXiv preprint arXiv:2205.04256* (2022) (cit. on p. 33).

[45] Bela Shrimali and Hiren B. Patel. «Blockchain state-of-the-art: architecture, use cases, consensus, challenges and opportunities». In: *Journal of King Saud University - Computer and Information Sciences* 34.9 (2022), pp. 6793–6807. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2021.08.005. URL: https://www.sciencedirect.com/science/article/pii/S131915782100207X (cit. on p. 34).

[46] Sisi Zhou, Kuanching Li, Lijun Xiao, Jiahong Cai, Wei Liang, and Arcangelo Castiglione. «A Systematic Review of Consensus Mechanisms in Blockchain». In: *Mathematics* 11.10 (2023). ISSN: 2227-7390. DOI: 10.3390/math11102248. URL: https://www.mdpi.com/2227-7390/11/10/2248 (cit. on p. 34).

[47] Manuel Adelin Manolache, Sergiu Manolache, and Nicolae Tapus. «Decision Making using the Blockchain Proof of Authority Consensus». In: *Procedia Computer Science* 199 (2022). The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 2021): Developing Global Digital Economy after COVID-19, pp. 580–588. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2022.01.071. URL: https://www.sciencedirect.com/science/article/pii/S1877050922000710 (cit. on p. 36).

[48] Stefano De Angelis, Gilberto Zanfino, Leonardo Aniello, Federico Lombardi, and Vladimiro Sassone. «Blockchain and cybersecurity: a taxonomic approach». In: *EU Blockchain Observatory, October* (2019) (cit. on p. 36).

[49] HTM Gamage, HD Weerasinghe, and NGJ Dias. «A survey on blockchain technology concepts, applications, and issues». In: *SN Computer Science* 1.2 (2020), p. 114 (cit. on p. 36).

[50] Nick Szabo. «Formalizing and Securing Relationships on Public Networks». In: *First Monday* 2.9 (Sept. 1997). DOI: 10.5210/fm.v2i9.548. URL: https://firstmonday.org/ojs/index.php/fm/article/view/548 (cit. on p. 36).

[51] Vitalik Buterin et al. «A next-generation smart contract and decentralized application platform». In: *white paper* 3.37 (2014), pp. 2–1 (cit. on p. 37).

[52] Gavin Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. Yellow Paper. Ethereum Project, 2014. URL: https://ethereum.github.io/yellowpaper/paper.pdf (cit. on p. 38).

[53] Weiqin Zou, David Lo, Pavneet Singh Kochhar, Xuan-Bach Dinh Le, Xin Xia, Yang Feng, Zhenyu Chen, and Baowen Xu. «Smart Contract Development: Challenges and Opportunities». In: *IEEE Transactions on Software Engineering* 47.10 (2021), pp. 2084–2106. DOI: 10.1109/TSE.2019.2942301 (cit. on p. 38).

[54] Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. «MadMax: surviving out-of-gas conditions in Ethereum smart contracts». In: *Proc. ACM Program. Lang.* 2.OOPSLA (Oct. 2018). DOI: 10.1145/3276486. URL: https://doi.org/10.1145/3276486 (cit. on p. 38).

[55] Abdeljalil Beniiche. «A Study of Blockchain Oracles». In: *CoRR* abs/2004.07140 (2020). arXiv: 2004.07140. URL: https://arxiv.org/abs/2004.07140 (cit. on p. 39).

[56] Yanbo Niu, Jun Li, Shukang Zhou Shukang Zhou, Gaoyang Liu, Yiqiang Xiang, He Zhang, and Jiangpeng Shu. «Dynamic displacement estimation and modal analysis of long-span bridges integrating multi-GNSS and acceleration measurements». In: *Journal of Infrastructure Preservation and Resilience* 4 (Apr. 2023). DOI: 10.1186/s43065-023-00077-6 (cit. on p. 43).

[57] Leonardo Zunino, Joan R. Casas, and Marco Domaneschi. «Bridge damage assessment under traffic and environmental variability: A case study on Yonghe cable-stayed bridge». In: *Engineering Structures* 343 (2025), p. 120965. ISSN: 0141-0296. DOI: https://doi.org/10.1016/j.engstruct.2025.120965. URL: https://www.sciencedirect.com/science/article/pii/S0141029625013562 (cit. on p. 47).

[58] Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Yinuo Zhang. «Succinct Zero Knowledge for Floating Point Computations». In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS '22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 1203–1216. ISBN: 9781450394505. DOI: 10.1145/3548606.3560653. URL: https://doi.org/10.1145/3548606.3560653 (cit. on p. 49).

[59] Ilaria Battiston, Lotte Felius, Sam Ansmink, Laurens Kuiper, and Peter Boncz. «DuckDB-SGX2: The Good, The Bad and The Ugly within Confidential Analytical Query Processing». In: *Proceedings of the 20th International Workshop on Data Management on New Hardware*. 2024, pp. 1–5 (cit. on p. 50).

[60] Chia-Che Tsai, Donald E Porter, and Mona Vij. «Gramine: A library OS for SGX enclaves». In: *Proceedings of the 2017 USENIX Annual Technical Conference*. 2017 (cit. on p. 52).

[61] Automata Network. *Releasing DCAP Library v4: A Unified Interface to Verify TEEs On-Chain*. Medium Blog. Accessed: 2026-03-01. Sept. 2024. URL: `https://blog.ata.network/releasing-dcap-library-v4-a-unified-interface-to-verify-tees-on-chain-2f27538babb4` (cit. on p. 59).

[62] *Azure Pricing Calculator*. `https://azure.microsoft.com/en-us/pricing/calculator/`. Accessed: Mar 2026 (cit. on p. 63).

[63] Automata Network. *dcap-qvl: Python wrapper for Intel SGX DCAP Quote Verification Library*. `https://pypi.org/project/dcap-qvl/`. Accessed: 2026-03-07. 2026 (cit. on p. 69).