Università degli Studi di Messina

Corso di Dottorato di Ricerca
in

CYBER PHYSICAL SYSTEMS

XXXIII Ciclo

Toward Trustless Internet of Things: a Blockchain-based approach

ING-INF/05

Nachiket Tapas

Coordinatore del corso:
Prof. Antonio Puliafito

Tutor:
Prof. Francesco Longo

Co-Tutor:
Dr. Giovanni Merlino

This page is intentionally left blank.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgments.

Nachiket Tapas
January 2021

This page is intentionally left blank.

Acknowledgement

This research would not have been possible without my advisor Prof. Francesco Longo for the continuous support of my Ph.D. study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D. study. I thank him for supporting me while I pursued researching Blockchain, which was at an early stage.

Besides my advisor, I would like to thank Prof. Antonio Puliafito and Prof. Giovanni Merlino for their insightful comments, sharing their vast knowledge, experience, and posing the hard questions that incented me to widen my research from various perspectives.

Outside of the University of Messina, I am eternally grateful to Prof. Yaron Wolfstal, Prof. Asaf Shabtai, and Dr. Yair Allouche for inviting me to visit IBM Israel and the Ben Gurion University of the Negev, Israel, to experience the Israeli research. It was a wonderful experience to meet and work with some of the brilliant people.

Outside of the research community, I thank my friends and fellow labmates Giuseppe Tricomi, Murtadha Arif Bin Sahbudin, Fabrizio De Vita, and Zakaria Benomar for the stimulating discussions and for all the fun we have had in the last three years.

I want to thank my parents and my sister and her family for supporting me spiritually throughout this journey, writing this thesis, and my life in general.

Finally, I am grateful to have truly lived this topic and witness first-hand the rise of Blockchain research from the start of this Ph.D. to its current state at the time of submission.

This page is intentionally left blank.

Abstract

The Internet of Things (IoT) is an innovative paradigm involving both industries and humans' every-day life. With the wide adoption of IoT, security becomes a crucial aspect due to the high level of heterogeneity of the involved devices and the managed information's sensitivity. With multiple and varying devices and entities involved, the system needs to be trustless as the participants involved do not need to know or trust each other or a third party for the system to function. Blockchain is a cryptographically, secure distributed data structure shared across a p2p network peers, where trust among peers is built to achieve consensus between peers. By relocating the trust agency to the cryptographically verifiable system, the need to trust any single entity in the system is removed.

The thesis addresses the challenges related to access control, data acquisition, storage and dissemination, and security patching among various IoT challenges. We address these challenges in a trustless manner for IoT systems. The thesis is organized into three sections. In the first section, we focus on protecting the IoT devices against unauthorized access and information leakage. A distributed access control mechanism is required to protect the devices. The capability of the devices should also be considered while designing the protocol. In the second section, we address the challenges related to protecting the entire data supply chain from sensing to storage and visualization. We need a trustless mechanism to protect a system against misbehavior. There are several trustiness issues in any IoT framework. During the data generation, contributors cannot necessarily be trusted. Data can either be unreliable or can voluntarily be forged. Then there is a single point of trust at the data storage level where the data is collected in the cloud, and again the hosting entity is in charge of secure storage. The same thing is valid for data retrieval and visualization. Also, there is no way for consumers to audit data in the centralized approach and, thus, to be sure that everybody is behaving correctly. In the final section, we focus on the challenges faced to protect an IoT device over an extended period. A distributed, secure, and trustless patching mechanism is required to secure the IoT devices.

The thesis presents the design and implementation of security primitives tailored to IoT application domains focusing on extending trustlessness to access control, data acquisition, storage, visualization, and security patching. The proposed solution's effectiveness is evaluated qualitatively and quantitatively evaluated employing a set of prototypes, case studies, modeling, and real measurements.

This page is intentionally left blank.

Contents

List of Figures

List of Tables

Introduction

Internet of Things (IoT) is a ubiquitous internetwork of intelligent physical objects, called ”Things,” and people. IoT empowers any ”Thing” to connect and communicate, thereby converting the physical world into an enormous information system. Like Cloud Computing and Machine Learning to Data Analysis and Information Modeling, various technologies are quickly becoming an integral part of the IoT fabric. The tremendous advancement in the field of IoT is causing growth in Information and Communication Technology (ICT) business as well. The last decade has seen tremendous growth in the Internet of Things (IoT) services and devices owing to rapid advancements in networking technologies. Gartner, Inc. predicts a 21% growth to 5.8 billion IoT endpoints by 2020 compared to 2019 (1). The extent to which IoT will be part of our day-to-day life can be understood from the fact that 95% of newly introduced products by 2020 will have IoT technology at its core (2). The growth is expected to reach 64 billion devices worldwide by 2025 (3), with the predicted market size to reach $520 billion by 2021 (4). With recent advances in next-generation mobile connection technology 5G, mobile subscriptions are predicted to reach 1.3 billion by 2023 (5). IoT is enabling the development of new business methods, and one of its most essential aspects resides in the data enhancement that will affect the growth in the ICT market.

IoT can be characterized by highly distributed architecture consisting of a combination of entities and devices. Trust is critical to integrate such a highly variable system to operate. With multiple and varying devices and entities involved, the system needs to be trustless; the participants involved do not need to know or trust each other or a third party for the system to function. There is no single entity with authority over the system in a trustless environment, and the consensus is achieved without participants having to know or trust anything but the system itself. Thus, because of the variety of users and devices, a distributed system of trust is required.

Blockchain creates a new, distributed architecture for trust. The trust is not placed on any one actor in the system but, instead, is placed in the system as a whole. Blockchain is a cryptographically secure distributed data structure shared across the peers of the p2p network, where trust among peers to achieve consensus between peers. By relocating the trust agency to the cryptographically verifiable system, the need to trust any single person is removed. It is thus, a system of ”trustless trust.” Blockchain’s technology makes it possible to reliably trust that everyone in the system is sharing the same information; what appears in one ledger will match every other instance. Blockchain can be characterized by the following features: decentralization, immutability, transparency, accountability, and smart contract-based automation. These characteristics make Blockchain a suitable fit for designing security primitives for IoT.

We focus on access control, data acquisition, storage and dissemination, and security patching challenges among the various IoT challenges. We address these challenges in a trustless manner for IoT systems. In the deployment phase, we need to protect the IoT device against unauthorized access and information leakage. With the IoT devices' deployment, a distributed access control mechanism is required to protect the devices. The capability of the devices should also be considered while designing the protocol. We need to protect the entire data supply chain from sensing to storage and visualization in the operational phase. We need a trustless mechanism to protect a system against misbehavior. Further, there are several trustiness issues in any IoT framework. During the data generation, contributors cannot necessarily be trusted. Data can either be unreliable or can voluntarily be forged. Then there is a single point of trust at the data storage level where the data is collected in the cloud, and again the hosting entity is in charge of secure storage. The same thing is valid for data retrieval and visualization. Also, there is no way for consumers to audit data in the centralized approach and, thus, to be sure that everybody is behaving correctly. Finally, in the maintenance phase, the IoT devices need to be protected over an extended period. A distributed, secure, and trustless patching mechanism is required to secure the IoT devices.

We begin by briefly describing the proposals to address the problem of access control and delegation in IoT. We consider two possible approaches: a solution for devices with sufficient resources and one for resource-constrained devices. A higher computational capability of the IoT devices allowed us to address the trustless requirement of IoT-Cloud framework (Stack4Things (6)) using Blockchains. We propose a general-purpose architecture with access control, authorization, and delegation model using Blockchain's smart contract capabilities. We begin by describing three real-world scenarios about access control and delegation in smart environments. Each scenario is characterized by its own set of requirements and limitations. We outline three different approaches for using Blockchain technologies for access control and delegation in IoT, each modeled to suit one of the scenarios mentioned above. Of each approach, we present the design considerations in detail, and we discuss how it deals with the requirements and limitations of the corresponding scenario. Next, we carry out a theoretical analysis of time and space complexity for each approach focusing on the three most essential operations: creating delegation, delete delegation, and check access. We then present performance measurements for each approach. We also increased the number of experiments to strengthen the conclusions that can be drawn from them. Finally, we discuss the results and their significance. We also present a critical discussion about the pros and cons of our proposed system.

Next, we address the problem of access control in resource-constrained devices. As an application domain, we consider the use-case of IIoT environments. We propose an original architecture for auditable authorizations with strong (cryptographic) guarantees suitable to IIoT environments. It represents the first architecture that guarantees: *(i)* the possibility of defining fine-grained access control rules; *(ii)* the support for constrained IoT devices that are characterized by low computing capabilities and by limited or absent Internet connection; *(iii)* the capability of system auditing for demonstrating possible illegitimate accesses. Our proposal allows each authority to release authorizations based on coarse-grained access control rules for its resources, thus satisfying usability. Each authorized party can release fine-grained authorizations autonomously with the only constraint of not violating the original delegation. The authorizations released by each party can be monitored by the authority that can provide evidence of misbehavior in illicit authorizations. The proposed architecture is specifically designed for industrial environments consisting of decentralized authorities collaborating with providers requiring flexible management of authorizations, high-security guarantees, and the capability to detect possible misbehavior. The overall architecture's performance is suitable for industrial environments, even if constrained devices and networks characterize them.

We focus on data acquisition, storage, and dissemination using trustless distributed ledger technology (DLT) - based systems. We begin by addressing the immutability and transparency challenges related to data storage in the cloud. Cloud storage adoption, due to the growing popularity of IoT solutions, is steadily on the rise and ever more critical to services and businesses. In light of this trend, customers of cloud-based services are increasingly reliant, and their interests correspondingly at stake, on the good faith and appropriate conduct of providers at all times, which can be misplaced considering that data is the "new gold," and malicious interests on the provider side may conjure to misappropriate, alter, hide data, or deny access. A key to this problem lies in identifying and designing protocols to produce a trail of all interactions between customers and providers, at the very least, and make it widely available, auditable, and its contents therefore provable. Considering the above scenario, we explore Blockchain as a possible solution to address the above limitations. We present a blockchain-based solution to solve the problem of trust in cloud storage. We describe the use-case scenarios, threat models, architecture, interaction protocols, and security guarantees and discuss how the proposal addresses the literature's challenges.

Next, we extend the previous proposal to secure the data across the data and network plane by focusing on securing the entire supply chain of data from production to storage. We propose building an IoT data collection framework that is decentralized (without any single point of failure), transparent (so that everything could be easily verifiable by everybody), and trustless (no requirement for any involved entity to be fully trusted, for the system to work). Decentralization, transparency, and trustlessness should be built-in at all steps so that there is no possibility of data forging, alteration, and/or censorship. We show how the IoT Data collection platform of the #SmartME project has been revised and extended by including a trustless system engaging each stakeholder (the University of Messina, the Messina Municipality) in the data storage and protection duties. This is achieved by introducing security features at different levels and enabling multiple entities and groups to participate at all levels of the data processing and consumption pipeline, as represented by the icons on the right side of Fig. 4.1. At the data origin, the Arancino platform is introduced that is equipped with a cryptochip partially able to act akin to a TPM (7) chip, e.g., as a secure enclave for keys and on-chip (i.e., host-invisible) crypto operations (signing, authenticating, encrypting, etc.), thus ensuring that known and authorized sources produce each piece of data. At the data storage level, an adapted version of the BigchainDB platform is proposed. Based on blockchain technology, BigchainDB guarantees decentralization (no single entity controls the data), scalability (multiple endpoints to which data can be delivered), and, last but not least, data immutability. In particular, concerning the latter, in the context of Open Data collection for Smart City services, it is of utmost importance that data may be relied upon as pristine along with all phases of the acquisition and storage process. Indeed, immutability lends suitable reliability guarantees to Open Data-based services, possibly extending data usage scope to legally binding processes as well, where any lesser standing in terms of genuine origin for data would be an unacceptable compromise.

Finally, we focus on keeping the devices safe for an extended period. As a solution, we present P^{4}UIoT, a pay-per-piece patch update for IoT software updates using a gradual release, an incentivized distributed delivery network based on Bitcoin's Lightning Network (8). The proposal reduces the transaction fee, as a single transaction with a commitment is mined on the Blockchain. This commitment is known as a channel. Once confirmed, further transactions happen purely peer-to-peer thus, improving the throughput of the system. Also, latency is reduced to network latency. Privacy is preserved thanks to an underlying onion-routing scheme and having just the payments' involved parties aware of a payment made and its terms. The gradual release concept refers to dividing and exchanging the commodity between the sender and receiver in rounds wherein the trust between participants grows with each round. We formally analyze the proposed approach using TLA+ (9) formal modeling language and evaluate the formal specification's correctness. Also, we implement a prototype of the proposed framework on the Bitcoin's lightning network to evaluate its performance. We perform experiments to evaluate the distribution latency and cost metrics in various test scenarios. Finally, we present the proposed framework's advantage by comparing it with existing work in the literature.

Contribution

There are three types of contributions derived from this thesis, which are further described as theoretical, methodological, and technological design contributions.
The theoretical contributions of this thesis involve a systematic survey exploring the application of Blockchain technology to the Internet of Things. This novel research survey forms the basis for the methodological and technological design contributions discussed later.

1. **A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain and iot integration: A systematic survey,"** ***Sensors*****, vol. 18, no. 8, p. 2575, 2018**

The methodological and technological design contributions of the thesis include the following publications.

1. **N. Tapas, G. Merlino, and F. Longo, "Blockchain-based iot-cloud authorization and delegation," in** ***2018 IEEE International Conference on Smart Computing (SMARTCOMP)*****, pp. 411–416, IEEE, 2018**

2. **N. Tapas, F. Longo, G. Merlino, and A. Puliafito, "Experimenting with smart contracts for access control and delegation in iot,"** ***Future Generation Computer Systems*****, 2020**

3. **L. Ferretti, F. Longo, M. Colajanni, G. Merlino, and N. Tapas, "Authorization transparency for accountable access to iot services," in *2019 IEEE International Congress on Internet of Things (ICIOT)*, pp. 91–99, IEEE, 2019**

4. **N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain-based publicly verifiable cloud storage," in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 381–386, IEEE, 2019**

5. **A. Khare, G. Merlino, F. Longo, A. Puliafito, and O. P. Vyas, "Toward a trustless smart city: the# smartme experience," in *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pp. 204–207, IEEE, 2019**

6. **A. Khare, G. Merlino, F. Longo, A. Puliafito, and O. P. Vyas, "Design of a trustless smart city system: the# smartme experiment," *Internet of Things*, vol. 10, p. 100126, 2020**

7. **N. Tapas, Y. Yitzchak, F. Longo, A. Puliafito, and A. Shabtai, "P4uiot: Pay-per-piece patch update delivery for iot using gradual release," *Sensors*, vol. 20, no. 7, p. 2156, 2020**

Organization

The remainder of the thesis is organized as follows. Chapter 2 provides the background on the problem of access control and delegation. In Chapter 3, the blockchain-based solution for resource sufficient devices is described in detail. Chapter 4 describes the authenticated data structure-based solution for resource constraint devices in detail and discusses the relevant scenario for each of the two approaches. In Chapter 5, the background related to data acquisition, storage, and dissemination is presented in detail. Chapter 6 describes the solution addressing the immutability and transparency challenge related to data storage in the cloud. In Chapter 7, the proposal of Chapter 6 is extended to address the security of data across data and network plane by focusing on securing the entire supply chain of data from production to storage. Chapter 8 provides the background on the problem of security patching to protect the IoT device against attacks. In Chapter 9, details related to a distributed device patching mechanism are presented. Finally, Chapter 10 provides conclusions and delineates the future work.

Part I

Access Control and Delegation

Chapter 1

Introduction

In this chapter, we focus on securing IoT devices against unauthorized access and information leakage. We begin by describing the challenges faced by the research community in extending access control and delegation for a highly distributed environment like IoT. Then, we present the literature survey of the existing approaches addressing the issues and their shortcomings. Next, we present a primer for the technologies used in the proposed solutions. Finally, we conclude the chapter by briefly describing the two proposed solutions and conditions in which they are efficient.

1.1 Problem Statement

Widespread deployment and usage of an IoT infrastructure depend on the system's resilience against unauthorized access and information leaks. Also, a flexible system allows the user-to-user transfer of rights, thereby facilitating cross-domain resource sharing and collaborations. (18; 19; 20; 21; 22; 23). Of the core concepts of data security, access control is central, dictating who can access and use information and resources. Access control policies utilize authentication, verifying the identity of the user, and authorization, ensuring appropriate access to information. User names and passwords, security codes, tokens, and biometrics are common techniques for verifying the identity by the access control mechanism. A combination of the above methods, called multifactor authentication, is also used for user identity verification to make the system robust. Once the user is authenticated, access control ensures appropriate access level and associates suitable actions with that user's identity. Thus, access control can ensure effective monitoring of access and prevention of unauthorized flow of information. Another critical aspect of access control is a user's ability to share access to a resource securely. The one sharing the rights is delegating while the one receiving them is delegated. With the delegation, the user privileges are temporarily elevated to allow the delegated user similar rights as the delegating user. This makes the delegation distributed in nature, with the user deciding who to share her access with compared to access control, which can be performed centrally. To further secure the system, specific models limit the operations which can be shared with the users and the users, which can inherit the privileges. However, there is a possibility of undesired permission propagation and information leakage in a partial or complete delegation of access rights by the user. For instance, if roles r1 and r2 are mutually exclusive, a user who is a member of r1 should not be allowed to receive r2 from others through delegation. Such a security requirement can be enforced using delegation authorization rules. The novel characterstics of IoT makes traditional access control techniques inadequate. The ubiquitous nature of IoT spans accross the Internet using information from the traditional Web to the information covering human, cyber, and physical. Give the broad nature of IoT, the access control can be identified with following features.

1. Volume: According to study presented (24), researchers claimed a 250% increase in M2M traffic in United States in 2011, and would account for half of the total traffic of the Internet by 2020. The volume with it brings unique challenges for data acquisition, storage, and maintenance.

2. Volatility: An IoT environment changes continuously with various entities being added and removed constantly. This dynamic characteristic makes it difficult to foresee user information and permission structure in advance.

3. Confidentiality: With more users getting involved in the IoT environment, data security and privacy becomes critical. Several proposals like privacy by design (25), ISO/IEC 29100:2011 (26), fair information practice principles (27), and general data protection regulation (28) have been made by the researchers. However, some of the principles are focused on individual control instead of protecting the data (29).

4. Cooperation: Various organizations are cooperating by sharing the data among themselves. This helps them to meet the complex application needs.

Traditional access control can prevent unauthorized information flow and ensure effective monitoring of resources. However, with the above listed features, the traditional access control methods and techniques become inadequate. Data in IoT environment is transmitted and shared continuously with users and things to achieve certain objectives (30), (31). Thus, access control and delegation becomes important to ensure secure communication (32). Under this scenario, the challenges of access control include guarantee the access permission, manage the scalable IoT architecture, handle the huge amount of data stream, and so on. Access control and delegation for IoT faces two fundamental challenges: *(i)* Can the existing access control mechanisms be adopted for the IoT use case or new mechanisms need to be designed from scratch? *(ii)* Is a distributed access control mechanism effective for scalable IoT architecture?

Integration of IoT devices with the Internet would require novel architecture keeping in mind the device capabilities. For a constrained environment, lightweight security mechanisms would be required to secure the devices while resource sufficient devices can afford higher security at the cost of complex computation. However, current security standards and access control solutions were not designed with such aspects in mind. They are not able to meet the needs of these incipient ecosystems regarding scalability, interoperability, lightness and end-to-end security. Consequently, a deep revision and adaptation of those mechanisms needs to be considered. A solution to IOT security is to not reinvent the access management experience. The patterns and protocols, which are now available to protect Web resources, should be carried over to IoT. Actually, adapting existing techniques to resource constrained devices, rather than developing new approaches towards the wider IoT requirements, will save time and lets developers focus on service logic rather than on security and authorization issues. Usually, access control is often done within the server side application, once the client has been authenticated. IoT reverses this paradigm by having many devices serving as servers and possibly many clients, taking part in the same application. More importantly, servers are significantly resource-constrained, which results in the minimization of the server side functionality. Subsequently, access control becomes a distributed problem, especially when taking into account the recent efforts of decoupling the sensor network infrastructure from applications (33; 34). With the bankruptcy of Lehman Brothers (35), trust in the internet is over. Building IoT solutions with centralized and collaborative system composed of trusted stockholders is now something of a fantasy. Most access control solutions today enable centralized authorities (e.g. governments or service providers such as Facebook or Google for example) to gain access and control devices by gathering and analyzing user data. A centralized solution suffers from many limitations. *(i)* Lack of end to end security: the inclusion of a central entity prevents end-to-end security to be achieved. *(ii)* Single point of failure : due to the fact that a single entity stores and manages all the data from a set of devices, any vulnerability might compromise a vast amount of sensitive information and even cause disastrous obstruction to the whole system. *(iii)* User is not involved in access control over his own data : When access control logic is located in the cloud or in a central entity, they have a full control over the hosted resources. As a result, user's control can be weakened. *(iv)* Expensive management : managing all IoT devices in a centralized way would be too expensive in the long term. IoT devices are envisioned as low-cost, low-maintenance devices that should run for years and even decades. *(v)* Trust foreign entities: Delegating the authorization logic to an external service requires a strong trust relationship between the delegated entity and the device. Moreover, all communications between them must be secured and mutually authenticated, so that the delegated entity security level is at least as high as if the authorization logic were implemented internally.

Another aspect related to access control and delegation is the nature of the IoT devices under consideration. The devices can be classified as: resource sufficient and resource constrained. The resource sufficient devices are powerful devices capable of high computation, high memory, network, and power availability. In contrast, the resource constrained devices lack the above mentioned features. The selection of suitable approaches in the two situations differs greatly.

1.2 Related work

We survey the literature for access control strategies keeping in mind the two strategies: resource sufficient and resource constrained. Initially, we assume the devices to be sufficiently powerful, and thus, capable of performing complex computations for distributed access control. Typically, a smart environment would involve participants sharing IoT resources to improve the quality of life. With participants being in control of the devices, we can assume steady power and network availability.

1.2.1 Authentication

Traditional access control models, like Role-Based Access Control (RBAC) model (36), depend on a trusted third-party authorization engine to grant access. This motivates strong caution in entrusting the system. Also, such models are based on a centralized model, which does not overlap with the distributed architecture of the IoT. Similar access policies like XACML (37) and CCAAC (38) frameworks result in heavy-weight implementations for resource constrained nodes, such as typical sensor/actuator-hosting boards and can, therefore, be considered unfeasible. This led us to explore blockchain technology as an access control mechanism, due to its distributed nature. Also, given the resource sufficient nature of the IoT devices, it will be possible for the IoT devices to utilize the security guarantees of Blockchain.

1.2.2 Authorization

Smart contract provides a secure and transparent authorization engine to evaluate the access request based on the policies stored on the Blockchain. A.Ouaddah et al. (39) used blockchain to store and audit access control policies. L.Chen and H.Reiser (40) stored access rights for a resource in a blockchain and managed them via transactions. MeDShare proposed by Xia, Qi, et al. (41), for controlling access to sensitive medical data, uses blockchain to store the history of operations performed on the data and smart contracts to enforce access control. (42) proposed a scheme based on Blockchain and smart contract to store and enforce access control policies. However, (42) did not support delegation. (43) proposed similar approach merged with IPFS based file sharing design. Authors discussed the impact of IPFS using an experimental setup. (44) explored permissioned Blockchain (Hyperledger Fabric) based solution to address the access control problem. The authors presented the performance metrics and resources consumption using the Hyperledger Caliper benchmark framework. Similarly, (45) proposed a permissioned Blockchain based access control ecosystem for large data sets. Authors utilized permissioned Hyperledger Fabric Blockchain for system implementation.

The devices that are constrained and thus, limited by the computations to be carried out by the devices. In an industrial internet of things (IIoT) use case, the IoT devices deployed in the industries or remote locations will not have access to a continuous power and network source. Given the constrained nature of devices, the security mechanism should require minimum computation on the devices. Simultaneously, looking into industrial systems, trustworthy industrial systems must ensure people's safety and infrastructure security, operation continuity, and reliability. Any violation of these requirements must be audited with the goal of identifying the causes and attributing the fault to the entities and operations causing problems. Authorization architectures and protocols for distributed systems are governed by multiple authorities, including federated identity systems such as the well-known OpenID (46) or the so-called self-sovereign identity systems (47), and authentication and authorization protocols specifically designed for constrained environments (48; 49). Standard solutions for authorization services protect the environment from illegitimate accesses from unauthorized personnel, but they typically do not consider malicious internal attackers that have high privileges, such as admins that have complete accesses. Detecting violations within outsourced services is related to the capability of detecting integrity of algorithms and of data. Security proposals include expensive protocols for verifiable outsourcing (50), which however are unsuitable because of high computational overhead. Other approaches for detecting correctness of algorithms and protocols are based on secure hardware enclaves, such as SGX (51), that seem still vulnerable to attacks (52).

1.2.3 Delegation

Delegation enables the assignment of authorization to another entity to carry out specific activities. It is the process of distributing and entrusting authorization to another entity. (53) proposed a decentralized access control approach based on Blockchain to overcome the limitation of present access control systems like performance and single point of failure. The proposed system also suggested capability delegation mechanism. The authorization is achieved through a capability token management strategy. The proposed system is implemented on local private Blockchain and Raspberry Pi device. (54) proposed a similar scheme based on Blockchain with a support for delegation. However, (54) limited the depth of the delegation. Finally, authors in (55) proposed a novel cryptographic primitive, named obligation chain, which is based on Blockchain and built-in reputation mechanism to create trust in the system. Authors presented a security analysis for both the obligation chain and the overall architecture and provide experimental tests.

For resource constrained devices, decentralized security solutions designed for multiple root authorities include blockchain technologies (56; 57), secure multi-party computation (58), and transparency logs (59; 60; 61). Blockchain protocols based on consensus and secure multi-party proposals cannot be used in many industrial environments due to device and network constraints. *Transparency logs* (59) based solutions allow the system to outsource a log service to semi-trusted parties without affecting the security of the system. These approaches are based on known cryptographic schemes to guarantee data integrity, such as digital signatures, MAC primitives, and so-called authenticated data structures to enable efficient operations for dynamic workloads (62; 63). Proposals deploying lightweight and flexible authentication and authorization schemes in constrained environments (49; 48) do not guarantee strong auditability. The most popular *transparency log* solution is Certificate Transparency (59) that monitors Certification Authorities by requiring browsers to verify that all certificates fetched from Web sites have been stored in approved authenticated logs. Literature also proposed a more general approach called *key transparency* (60) that prevents the so-called *equivocation*, which is an attack based on binding different cryptographic keys to the same identity. This solution cannot be trivially applied to the considered industrial scenario due to the different characteristics of the delegated authorization protocols, including management of secret information and updates to the authorization policies. Finally, a proposal based on transparency logs has been proposed by the same authors for smart cities (61). It allowed users to detect malicious behaviors of cloud services that act as delegated authorization services, but it cannot be used in federated environments characterizing industrial systems where users and authorization service belong to the same trust domain and might collude to avoid detection. Security solutions that also contribute to guarantee security of IIoT architectures, include network segregation based on firewalls and data diodes (e.g., (64)), and intrusion detection systems (e.g., (65)). Unfortunately, industrial systems are characterized by legacy industrial protocols, network protocols with small packet sizes that cannot be naïvely integrated with security measures, and network-enabled sensors with low computational power that cannot operate many standard security protocols. In literature, there are proposals that are designed for IIoT systems, such as network intrusion detection systems for automotive networks (66) and automation systems (67). Intrusion detection systems can cover a wider range of attacks other than those related to authentication and authorization procedures, but they are unable to provide strong evidence of misbehaviors. Moreover, their adoption in possibly disconnected networks can delay audit and detection of threats, and there are useless in the case of encryption network protocols. Other proposals include scalable identification systems, such as physical unclonable functions (68), strong authentication protocols to control remote access (69), architectures for offloading expensive cryptographic operations (70) and lightweight asymmetric cryptographic primitives and protocols (71; 72). Additional challenges are related to the security of the software operated by the IoT devices, including the capability of verifying whether a device has been compromised and the possibility of fixing vulnerabilities through software patching. Major proposals include remote attestations to verify the integrity of the executed software (73), and scalable and reliable architectures for long-term software updates (74).

1.3 Preliminaries

In this section, we present the background technologies used in the proposed solutions. We begin by introducing the OpenStack cloud infrastructure which serves as a backbone. Next, we present the architecture of Stack4Things framework, an extension to OpenStack for management of fleet of IoT devices. We illustrate details about how the already existing authorization and delegation mechanisms works. Subsequently, we present blockchain technologies, focusing on Ethereum (75) and smart contracts as an enabler of one of the solutions. Finally, we present authenticated data structures (76), which form the basis of our second proposed solution.

1.3.1 Openstack

OpenStack[1] (77) is a set of open-source software tools for building and managing cloud computing platforms. OpenStack is a centerpiece of infrastructure Cloud solutions for most commercial, in-house, and hybrid deployments, as well as a fully Open Source ecosystem of tools and frameworks. Currently, OpenStack allows to manage virtualized computing/stor-

[1]See https://rb.gy/h5c79a.

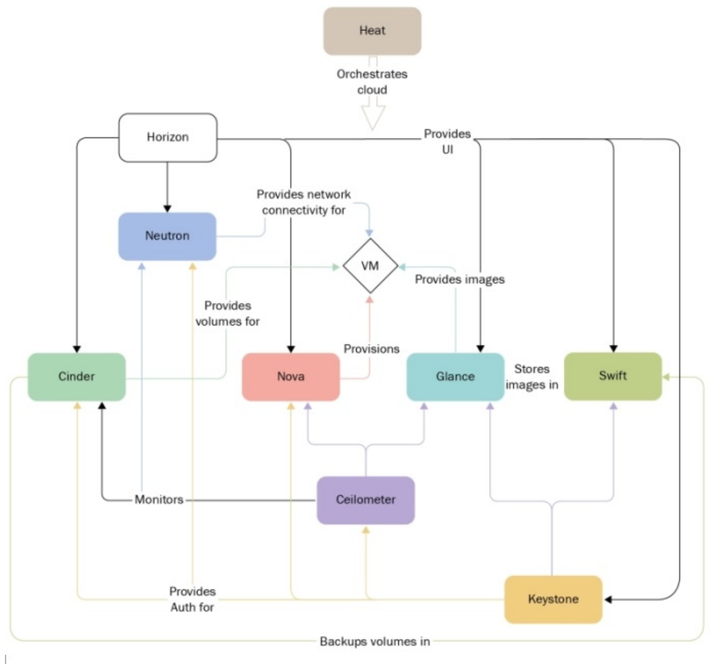


Figure 1.1: Openstack Architecture.

age resources, according to the infrastructure Cloud paradigm. In Figure 1.1, a conceptual architecture of OpenStack depicting components, as boxes, and the services they provide to other components, with arrows, are shown, respectively. Nova, the compute resource management subsystem, lies at the core of OpenStack and provisions VMs, with the help of a number of subsystems that provide core (e.g., networking in the case of Neutron) and optional services (e.g., block storage, in the case of Cinder) to the instances. Horizon is the dashboard and as such provides either a (web-based) UI or even a command-line interface to Cloud end users. Ceilometer, the metering and billing subsystem, like most other components of the middleware, cannot be fully analyzed on its own, as it needs to interface to, and support, Nova. In particular, while both Nova and any of the aforementioned subsystems exploit a common bus, the former alone dictates a hierarchy on participating devices, including their role and policies for interaction. Indeed, Nova requires a machine operating as Cloud controller, i.e., centrally managing one or more Compute nodes, which are typically expected to provide component-specific services (e.g., computing) by employing a resource-sharing/workload-multiplexing facility, such as an hypervisor.

1.3.2 Stack4Things authorization and delegation

The overall architecture of Stack4Things is defined in (6). It highlights interactions and communication facilities between end users, the Cloud, and (possibly mobile) sensor- and actuator-hosting IoT nodes. On the IoT node side, the *Stack4Things lightning-rod* represents the point of contact with the Cloud infrastructure allowing the end users to manage node-hosted resources even when nodes are behind a NAT or a strict firewall. This is ensured by WebSocket-based tunneling and WAMP-based[2] messaging between the Stack4Things lightning-rod and its Cloud counterpart, namely the *Stack4Things IoTronic service.* The Stack4Things IoTronic is designed as an OpenStack service, providing end users with the possibility to manage one or more nodes, remotely. The main goals of IoTronic lie in extending the OpenStack architecture towards the management of mobile/embedded system-hosted sensing and actuation resources. In Stack4Things, authentication mechanisms are implemented by leveraging the OpenStack Keystone subsystem. Keystone is an implementation of the OpenStack Identity service, providing authentication and high-level authorization mechanisms through the use of credentials and authentication tokens. To authorize an incoming request, Keystone validates a set of credentials supplied by the issuing user. Initially, the credentials are represented by a username and a password. As soon as such credentials are validated by Keystone, an authentication token is released that can be exploited by the user in subsequent requests.

OpenStack ensures API protection by exploiting the role-based access control (RBAC) model. Each token issued by Keystone includes a list of roles for the user. When the user calls a service, that service interprets the set of user roles and determines which operations or resources each role grants access to. However, delegation mechanisms, i.e., temporarily granting a user permissions to perform an API call on a specific subset of the resources available as a whole and then revoking them, could be quite hard to implement in terms of Keystone APIs, due to the requirement to dynamically associate API calls to roles, a feature which requires a number of non-trivial modifications to policy (JSON) files. If a user wants to perform an API call on a resource, the user needs to be a part of a specific project, and role with required permissions need to be setup and assigned to the project. This means that to protect an object or a small sub-set of objects by custom rules, one or more roles with required permissions are defined, a grouping of objects is done using projects, and an association between roles and project is set. For similar reasons, several OpenStack services, e.g., Swift, implement their authorization and delegation mechanisms on resources without using the standard policy JSON files. For such a reason, we devised for Stack4Things its own authorization and delegation mechanisms to access resources without resorting to the standard OpenStack approach.

Figure 1.2 shows the Cloud-side Stack4Things architecture with specific focus on authentication, authorization, and delegation mechanisms. While authentication is implemented by fully relying on Keystone, a specific IoTronic agent, namely the *Stack4Things IoTronic delegation agent*, deals with authorization and delegation duties. Whenever an API call is issued by a user on a specific IoT-node and after the authentication token is validated by interacting with Keystone, the Stack4Things IoTronic APIs contacts the *Stack4Things IoTronic delegation agent* to check if the user is granted access to the specific operation on that specific IoT-node or not. If authorization is granted, the Stack4Things IoTronic APIs deliver the request to the *Stack4Things IoTronic conductor* which triggers all the steps that are needed to fulfill it, involving other Stack4Things IoTronic agents if required.

Figure 1.3 depicts the entity-relationship model for authentication and delegation mechanisms in Stack4Things. The *Node* entity represents the set of IoT nodes managed by the

[2]See `http://wamp.ws`.

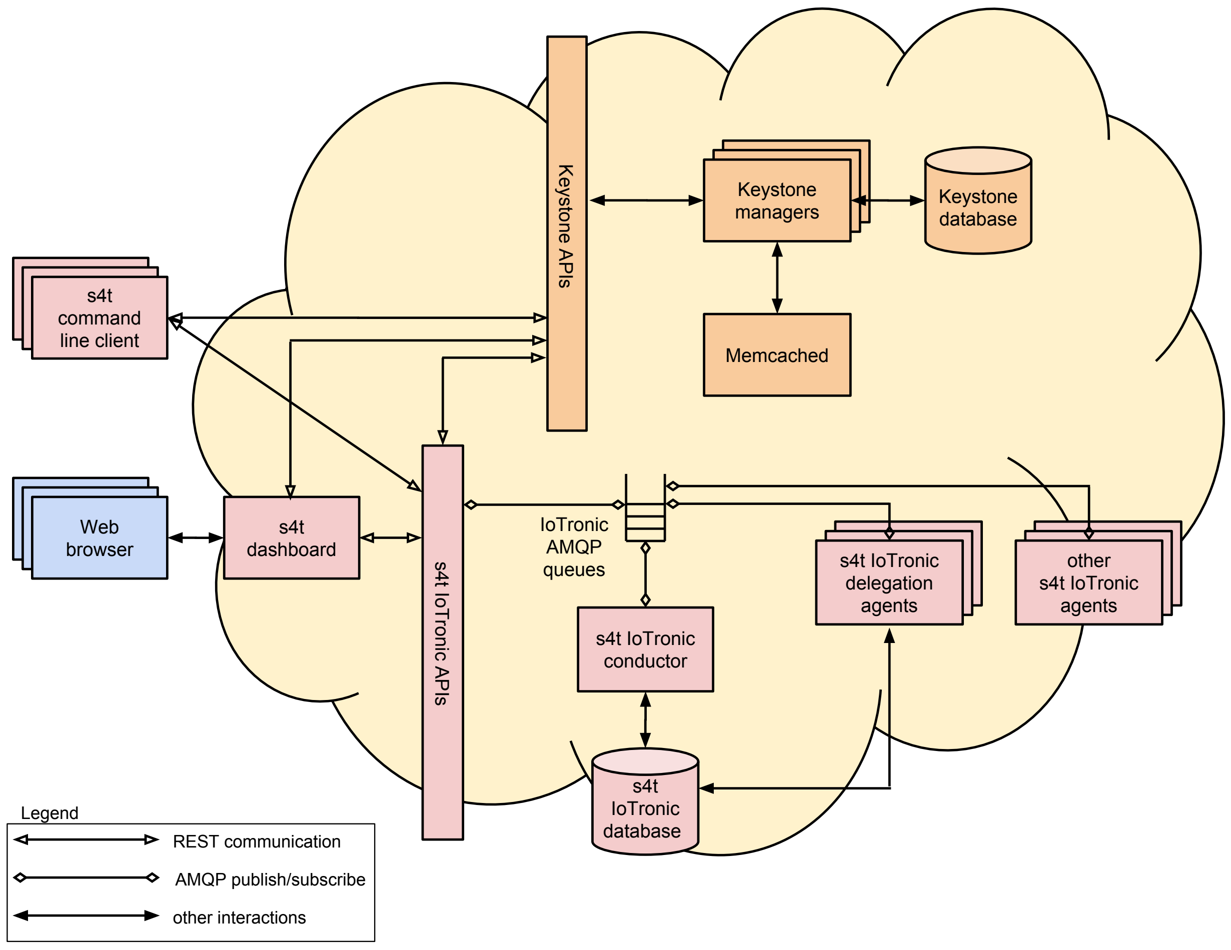


Figure 1.2: Cloud-side Stack4Things architecture with focus on authentication, authorization, and delegation mechanisms.

IoTronic service, while the *User* entity represents the set of users that are somehow allowed to interact with IoT nodes. The *Operation* entity models the set of API operations that needs to be authorized on specific IoT nodes. The *Role* entity represents the set of IoTronic roles that are currently defined in the system.

Finally, the *Delegation* entity contains the list of roles assigned to users for each IoT nodes. Delegations can be distinguished in two main categories. First level delegations represent roles that are assigned to users by default or by an administrator, e.g., the role IoT-node-owner is automatically assigned to the user that creates a node in the system, representing the situation in which a user contributes a node to the system and therefore has all the permissions on that node. Lower level delegations represent roles that are temporally assigned to a user by another user that is already granted with that role, or with a role that includes all the considered permissions. For example, a node owner has various permissions (list, login, read, write). This is a first-level delegation created by an administrator. Another user wanting to access read API can request read access from the node owner. The user,

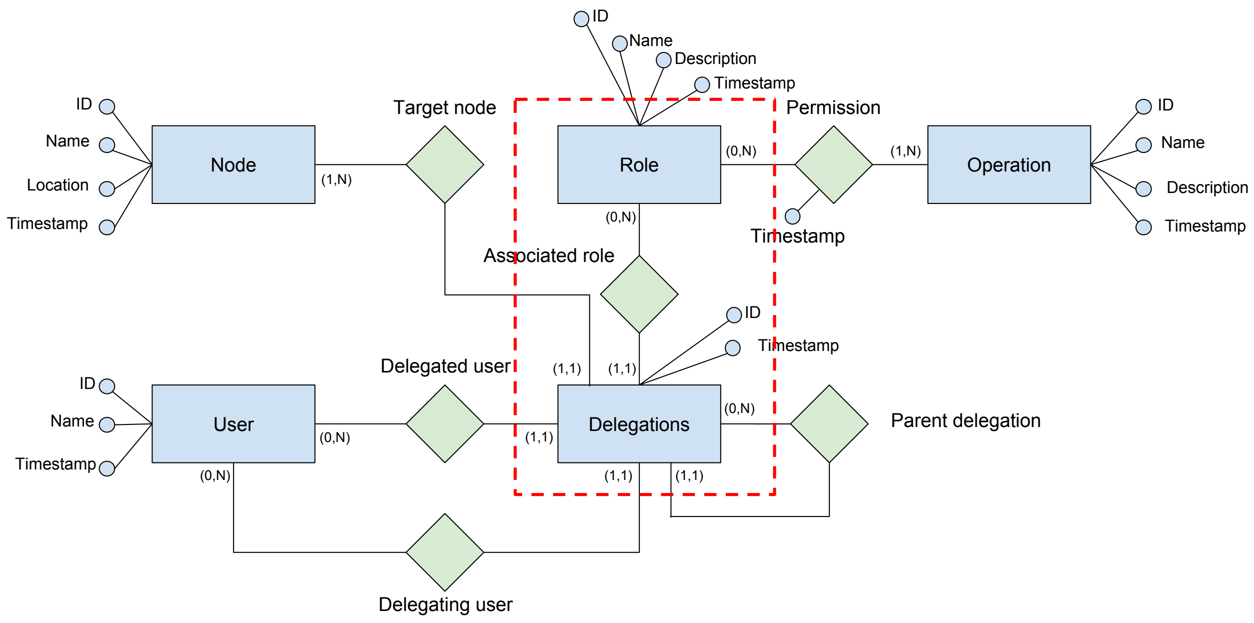


Figure 1.3: The entity-relationship model for the authorization and delegation mechanisms in Stack4Things.

in this case, will have only read permission. This is a second-level delegation created by the node owner. In this sense, they represent an operation of the delegating user entrusting to the delegated one. Each lower level delegation contains the information about the parent delegation from which it derives. In this way, the IoTronic implements a multi-level delegation scheme in which, if a delegation is revoked, all its child delegations are revoked (cascade revocation). More details about the legacy mechanisms that have been enhanced in this work can be found in (78). The red box, in Figure 1.3, highlights the portion of the data model that has been implemented on-chain, while migrating toward blockchain-based mechanisms.

1.3.3 Blockchain, Ethereum, and Smart Contracts

In 2008, Satoshi Nakamoto (79) proposed a cryptocurrency named Bitcoin in a completely decentralized environment. Blockchain is the enabling technology behind Bitcoin. A Blockchain is a public distributed ledger that stores all transactions ever executed in the network. It is a sequence of blocks in which each block is linked to the previous block via its cryptographic hash. The relationship is established by storing the parent block's hash in a child's block header (Figure 1.4). Transactions are hashed and stored in the block. The hashes of two transactions are taken and hashed further to generate another hash. This process eventually provides a single hash from all transactions stored within the block. This hash is known as Transaction Merkle root hash and stored in Block's header. Ethereum uses SHA256 for all its hashing needs.

The design of the Blockchain is such that it resists any modification to data. Once the

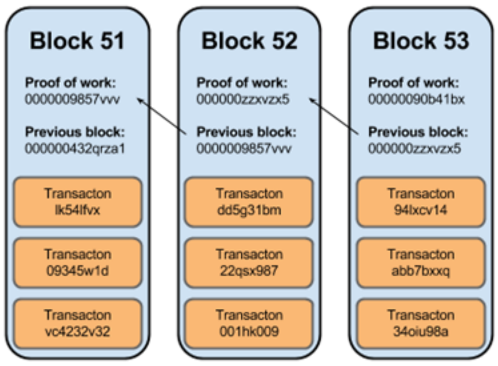


Figure 1.4: The relationship between blocks.

data has been recorded in a block on the chain it cannot be altered without changing all subsequent blocks. This immutable nature of blockchain leads to an open, distributed ledger capable of efficiently recording transactions between couple of entities and allows public verification. Blockchain can be particularly useful in situations requiring maintenance of unmodifiable and verifiable audit trail.

Ethereum is a generalized variant of a decentralized transaction ledger (75). The key elements of Ethereum are Blockchain as a backbone, a Turing-complete language, and an unlimited publicly verifiable storage. It can be viewed as a transaction-based state machine where each valid transaction causes the state machine to move to the next state, thus, forming a chain of states. Another important feature of Ethereum is Smart Contract. Ethereum implements DApps through smart contracts running on top of the EVM. In 1994, Nick Szabo - a legal scholar, and cryptographer - realized that a decentralized ledger could be used for smart contracts[3]. Smart Contracts are a formal generalization of a transaction-based state-machine. Also known as self-executing contracts, Blockchain contracts, or digital contracts, smart contracts enable the exchange of cryptocurrency, or any tangible asset in a fair, conflict-free way, while eliminating any third party. A sample smart contract is presented in Data Structure 1. The smart contract consists of member variables (storedData) which hold the data while member functions (get, set) which interact with the blockchain. Smart contracts can be characterized by three properties namely autonomy, decentralization, and auto-sufficiency. Autonomy implies once the contract is on the Blockchain, no interaction is required from the initiator of the contract. Smart contracts are decentralized in nature as they exist on the Blockchain. Auto-sufficiency points towards the automatic execution of contracts based on certain conditions. Smart contracts particularly solve the problem of a trusted third party in any system. The smart contract triggers automatically on certain programming conditions and is publicly verifiable on the Blockchain. For example, based on the price of a car, the insurance premium for the car is calculated and inserted into the Blockchain via a transaction. In contrast, the transaction is initiated by the smart contract instead of by a user. It can eliminate the need for a trusted entity. Smart contracts can find

[3]See `http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html`.

application in various scenarios e.g., real estate law, financial transactions, legal processes, crowdfunding, insurance premiums.

Data Structure 1: Sample smart contract.

```
pragma solidity ^0.4.0;
contract SimpleStorage {
    uint storedData;
    function set(uint x) public {
        storedData = x;
    }
    function get() public view returns (uint) {
        return storedData;
    }
}
```

1.3.4 Authenticated Data Structures

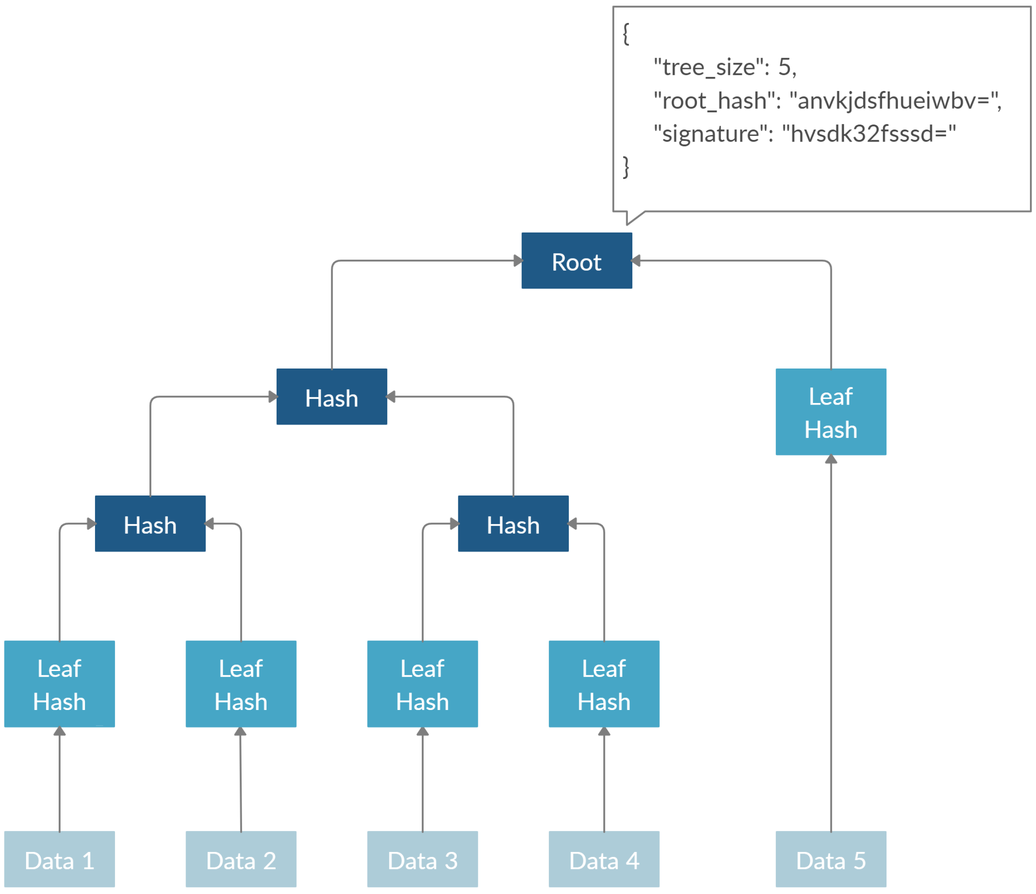
Trust is critical for any information sharing system. On the Internet, trust can be in the form of websites trusting CA for certificates, administrators trusting package distributors for the binaries of the software. Authenticated data structures integrate transparency and public auditing to the trust model, allowing entities to participate in a trustless manner. By adding transparency to services, trust can be verified by the ecosystems that depend upon them.

Verifiable Logs: The first structure we describe is an append-only log. As the name suggests, the logs are appended to the data structure. Once a log is accepted, it is cryptographically added to the log store and thus, cannot be altered. Periodically the log will publish a signed tree head which includes a root hash of all entries for a given log size. Clients of the log can: *(i)* Efficiently verify that a specific entry is included in the log. *(ii)* Efficiently detect split-view attacks. *(iii)* Efficiently verify the append-only property of the log. *(iv)* Enumerate all entries held in the log.

Verifiable Maps: The next structure we describe is a verifiable map. A verifiable map is a map from a set of keys to a corresponding set of values. Periodically the map will publish a signed tree head which includes a root hash of all 2^{256} entries. Clients of the map can: *(i)* Efficiently retrieve the value (or indication of nonpresence) for a key, and verify that this is included (or not present) in the map at a point in time. *(ii)* Efficiently detect split-view attacks. *(iii)* Enumerate all key/value pairs held in the map.

1.4 Proposal Summary

In this section, we briefly describe the proposals to address the problem of access control and delegation in IoT. We begin by proposing the solution for resource sufficient devices. A higher computational capability of the IoT devices allowed us to address the trustless



```
{
    "tree_size": 5,
    "root_hash": "anvkjdsfhueiwbv=",
    "signature": "hvsdk32fsssd="
}
```

Figure 1.5: Verifiable logs Merkle tree.

requirement of IoT-Cloud framework (Stack4Things (6)) using Blockchains. We propose a general purpose architecture with access control, authorization, and delegation model using the smart contract capabilities of Blockchain. We begin by describing three real world scenarios pertaining to access control and delegation in smart environments. Each scenario is characterized with its own set of requirements and limitations. We outline three different approaches for the use of Blockchain technologies for access control and delegation in IoT, each modeled to suit one of the above mentioned scenarios. Of each approach, we present in details the design considerations and we discuss how it deals with the requirements and limitations of the corresponding scenario. Next, we carry out a theoretical analysis of time and space complexity for each approach focusing on the three most important operations, namely create delegation, delete delegation, and check access. We then present performance measurements for each approach. We also increased the number of experiments to strengthen the conclusions that can be drawn from them. Finally, we discuss the results and their significance. We also present a critical discussion about the pros and cons of our proposed system.

Next, we address the problem of access control in resource constrained devices. As an

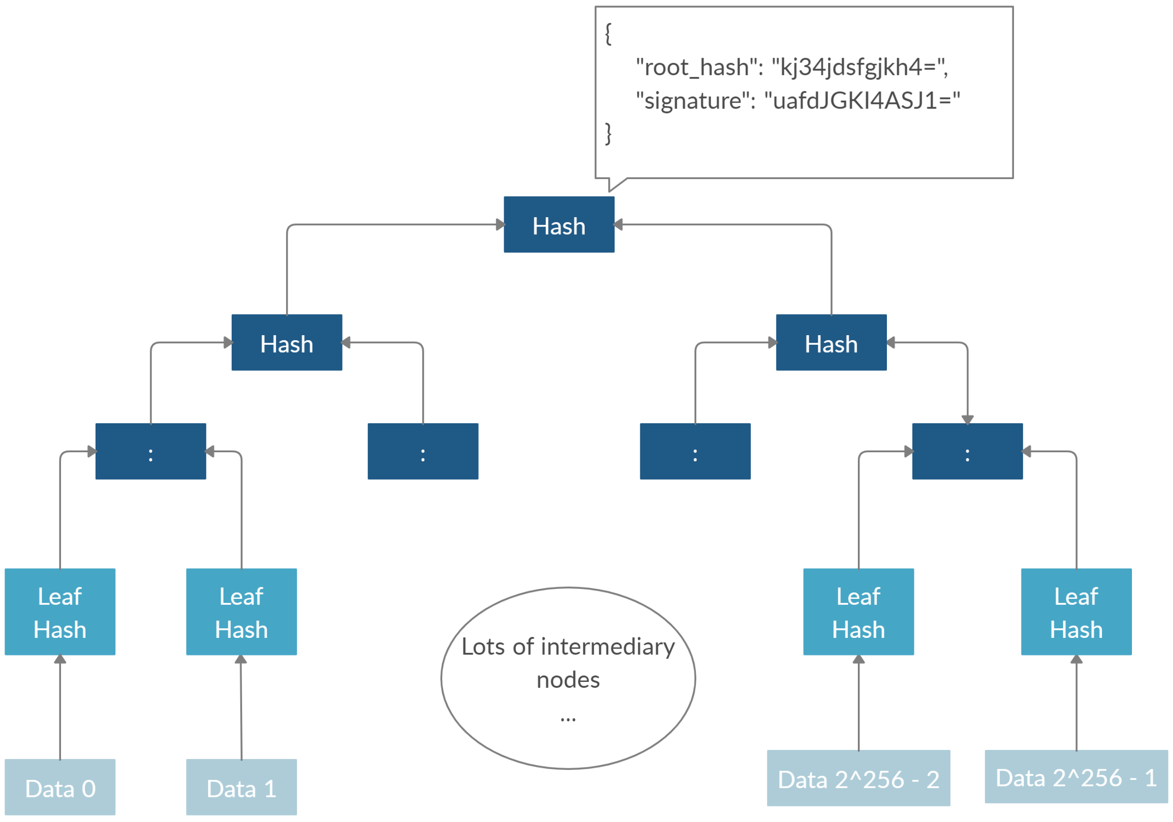


Figure 1.6: Verifiable maps Merkle tree.

application domain, we consider the use-case of IIoT environments. We propose an original architecture for auditable authorizations with strong (cryptographic) guarantees that is suitable to IIoT environments. It represents the first architecture that guarantees: *(i)* the possibility of defining fine-grained access control rules; *(ii)* the support for constrained IoT devices that are characterized by low computing capabilities and by limited or absent Internet connection; *(iii)* the capability of system auditing for demonstrating possible illegitimate accesses. Our proposal allows each authority to release authorizations based on coarse-grained access control rules for its resources thus satisfying usability. Each authorized party has the capability of releasing fine-grained authorizations autonomously with the only constraint of not violating the original delegation. The authorizations released by each party can be monitored by the authority that can provide evidence of misbehavior in the case of illicit authorizations. The proposed architecture is specifically designed for industrial environments consisting of decentralized authorities collaborating with providers that require flexible management of authorizations, high security guarantees, and the capability of detecting possible misbehaviors. The performance of the overall architecture is suitable to industrial environments even if they are characterized by constrained devices and networks.

The proposed system enforces security by considering covert security assumptions with public auditability (80; 81). It does not prevent misbehaviors of the delegated parties, but guarantees accountability of their actions. Hence, the audit mechanism acts as a deterrent for third parties with a reputation. An authority can detect and show evidence of wrong or malicious behaviors of the authorized partners, possibly allowing early detection of cyber attacks and violations. Alternative solutions based on intrusion detection systems cannot produce publicly verifiable proofs of misbehaviors, and their efficacy might be hindered by encryption protocols (67). Other proposals based on cryptographic protocols (82; 83) and public distributed ledgers (56; 11) cannot be deployed in IIoT environments because of their high requirements in terms of computation, storage and network traffic. Solutions adopting the general transparency approach (59; 60; 61) cannot be immediately deployed in the context of authorization management for IIoT because they are designed either to handle public information or for different architectures and threat models.

Chapter 2

Access control and delegation for resource sufficient devices

In this chapter, we describe the proposal to address access control and delegation in resource sufficient IoT devices. We present a general-purpose architecture with access control, authorization, and delegation model using Blockchain's smart contract capabilities. We describe the proposed access control and delegation solution (12; 84) and the design choices involved. We begin with the motivation for the proposed methodology and then briefly describe the development process flow. It provides the reader with an overview of the steps involved. This is followed by a brief description of the time and space complexity evaluation process. Then, we present details about how the implementation of the Blockchain-based authorization and delegation mechanisms in Stack4Things has been conducted. Further, we show how smart contracts have been designed to focus on the data structures and their interfaces. The description of proposed approaches is presented incrementally, with each one customized to tailor one of the discussed use cases. Finally, we present the algorithms' pseudocode and their theoretical analysis in terms of time and space complexity.

2.1 Motivations

The IoTronic service, discussed in the chapter 2 (Section 1.3.2), is supposed to be fully trusted by end users that expect it to manage IoT resources and provides access to them in a fair way. However, malicious administrators and/or unfair internal policies could affect IoTronic supposed behavior by, e.g., forcing it to deny access to specific operations to users with full rights. Moving authentication and delegation mechanisms to the blockchain allows moving part of the trust to the deployed smart contracts and it can be considered as a first step in the migration toward a fully decentralized version of the Stack4Things framework. The aim of the present work is to leverage blockchain technologies with the following motivations: *(i) Trustiness*: Ethereum smart contracts are deployed and run on top of all the nodes in the network and consensus is reached each time a smart contract is invoked. The consensus mechanism used in Ethereum is based on Proof-of-Work (PoW) (79) algorithm. In PoW, the election of an actor as a leader is based on a hard mathematical problem. The actor able to solve the problem becomes the leader. The leader then decides the next block to be added to the blockchain. All other actors present in the network are able to verify the solution of the leader. In this way, a consensus is achieved in a distributed manner. Moreover, provided that the user is able to check that the bytecode, that is deployed on the blockchain, is actually coming from the claimed source code (this is actually possible and similar services are even provided online by, e.g., the Etherscan Contract Verification tool[1]), he/she can reasonably be sure that access control is performed the way it is documented. *(ii) Auditing*: However, this is not enough to reach a complete trustiness in the system because the user should also be sure that the Stack4Things IoTronic delegation agent is actually interacting with the smart contracts to check for user permissions. Thus, another reason for the use of blockchain is the possibility to implement a public auditing system. Thanks to blockchain (and specifically via Ethereum events and logs), each operation can be logged on the blockchain and the end user is able to check why access to a certain operation/resource has been granted or denied.

The type of operations associated with a role is dependent on the type of system in which RBAC policy is implemented. We have considered the following operations to be logged on the blockchain: *Create* to create a new node, *Delete* to delete an existing node, *List* to show information about a node, *Login* to create an SSH tunnel to a node, *Read* to read a value from a sensor on a node, *Write* to write a value to an actuator on a node, *Config* to configure a node, and *Inject* to inject custom code in the node. Note that, in the present work, privacy is not taken into consideration, as a requirement. In fact, maintaining the public auditing system and guaranteeing user privacy at the same time could be complex. Future work will be devoted to investigating this aspect.

2.2 Use Cases

This section discusses the use cases and their typical characteristics. The requirements of each use case are different as it is implemented. The use cases differ in terms of the number of users and resources present in the system, the frequency with which the users enter and leave the system, and the response times of the system. Also, we outline the characteristics of the methodology suitable for each use case.

The first use case is Smart Home. The use case involves a limited number of resources being used by a limited number of users. The users of the system enter and leave the system rarely. This scenario involves a central authority, like an administrator, creating few delegations with all operations for a set of users (family members) and these delegations are deleted rarely. Thus, this is a static scenario wherein users are delegated once and they never leave the system. Based on the characteristics we can infer that since the use case involve a limited number of delegation additions and even fewer delegation deletions, a methodology with linear time complexity for delegation additions and quadratic time complexity for delegation deletions can be suitable. Also, the number of operations/events related to smart services will be small in number. Thus, a methodology with linear to quadratic time complexity for access decisions can be suitable.

[1]See `https://etherscan.io/verifyContract2`.

The next use case is Smart Hotel. This use case also involves a limited number of resources being used by a limited number of users. However, the users of the system enter and leave the system frequently. This case is similar to the first use case in terms of the total number of delegations in the system which are kept low as users enter and leave the hotel frequently. This case is characterized by a highly dynamic scenario wherein delegations are created and deleted very frequently in a small system. The use case involves rapid creation and deletion of delegation. Besides, the number of operations will be limited in number related to smart services. Therefore, a methodology having linear time complexity for delegation addition and deletion, and linear to quadratic time complexity for access decision can be suitable.

The third use case is that of a Smart City. The typical characteristics of the use case include the presence of a lot of delegations in the system and the necessity to add/delete them frequently. Smart City exhibits these characteristics due to the rapid movement of its inhabitants in and out of the system and a large number of smart services being offered to the inhabitants of the city. Thus, a methodology with linear time complexity for all three operations namely delegation creation, delegation deletion, and access decision can be suitable.

The final use case involves highly interactive small systems. The typical characteristic of such a system is immediate response and the limited number of delegations. Smart Museum or Smart Immersive Environment exhibit such characteristics. As the emphasis is on the response time, the methodology needs to exhibit quick access decision. Also, since the number of delegations is limited, a linear to quadratic time complexity for delegation creation and deletion can be suitable.

2.3 Development process flow

We begin with the design of a smart contract by analyzing the data structure that needs to be stored on the blockchain. We identify the critical operations of access control and delegation like creation, verification, and deletion of access policies. Each step in the process optimizes the performance of one of the operation identified above. The initial solution is designed keeping in mind the best practices for smart contract development [2]. After the initial design, we perform the time, space and cost analysis to understand the performance of key operations. As part of the next step, we identify the bottleneck and make a change in the algorithm to improve the performance. We repeat the process to make further improvements and systematically present the analysis in the further sections.

2.4 Time and space complexity.

The time complexity of three basic operations of access control and delegation are analyzed namely creation, verification, and deletion of delegation. The time complexity of a single delegation creation is dependent on the time complexity of subset check i.e. if the beneficiary's role is a subset of delegator's role. The time complexity of deletion of a single delegation is

[2]See `https://consensys.github.io/smart-contract-best-practices/`.

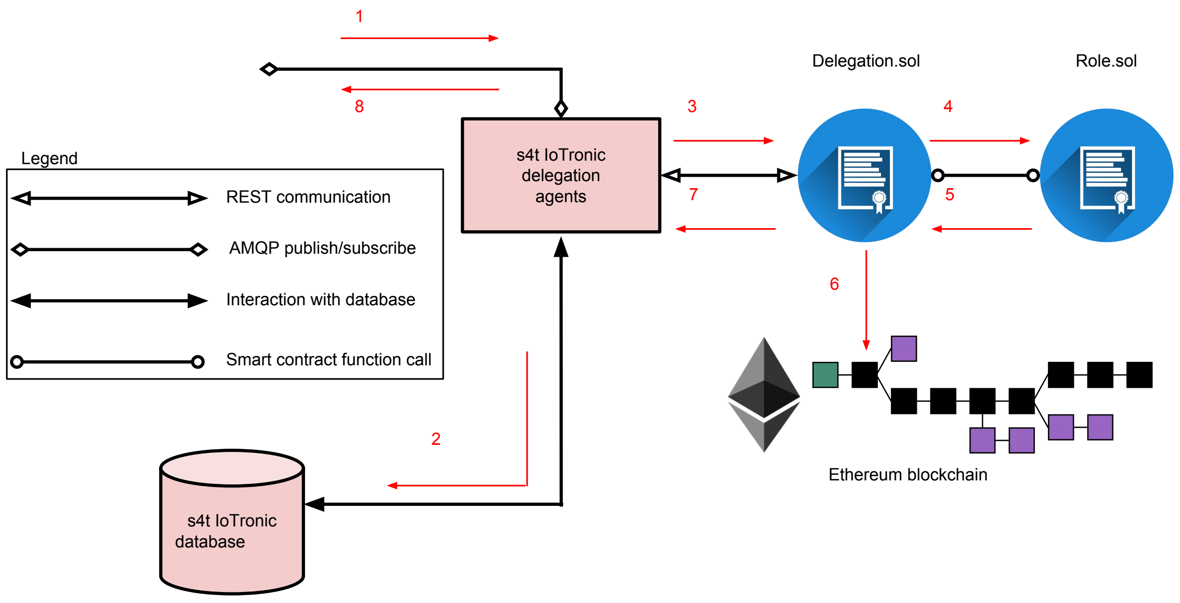


Figure 2.1: Interaction between s4t iotronic delegation agent and smart contracts on the ethereum blockchain.

dependent on the collection and deletion of its child delegations. Finally, the time complexity of delegation verification is dependent on number of roles in the system and time to verify if a role is authorized to perform an operation. The space complexity analysis identifies the need of additional memory or disk space to perform the basic operations. It is represented in terms of number of delegations in the system.

2.5 Design choices

This section deals with the considerations made during the design of the proposed approaches and the logical steps from ideation to implementation. We present various development environments and the motivation behind considering those environments. In order to tune smart contracts' code size to satisfy gas requirements, we designed two separate contracts. `Role.sol` contract captures the association of each role with the corresponding allowed operations, while a second contract, named `Delegation.sol`, represents the relationship between users, resources, and roles. Composition design strategy is chosen over inheritance with the goal of reducing coupling between the two smart contracts, as both the paradigms are supported by Solidity. Such a decision presents a second advantage: inheritance would lead to a single contract being actually deployed to the Blockchain as the inherited class code will be copied to the inheriting class. Also, it would lead to gas consumption crossing the gas limit threshold. We tested the contracts on the Ethereum public testnets (i.e. Rinkeby) - which implements a proof-of-authority based consensus algorithm - to observe real world performance of our system. Proof of Authority (PoA) is a consensus algorithm where a leader's identity, responsible for the addition of block to the blockchain, is at stake. Staking identity means voluntarily disclosing who you are in exchange for the right to validate the blocks. Identity placed at stake can serve as a great equalizer, understood and valued the same by all actors. Individuals whose identity (and reputation by extension) is at stake for securing a network, are incentivized to preserve the network. Performance parameters have been recorded to present a comparative study of our experiments.

Figure 2.1 represents the sequence of interactions that takes place when access to a specific operation on a specific IoT resource needs to be granted to a user. In step 1, the user requests for a particular operation on a specific IoT resource and the request is redirected to the S4T IoTronic delegation agent by the S4T IoTronic conductor. In step 2, the agent checks the S4T IoTronic database for validating the user request (e.g., the existence of the IoT resource is checked). In step 3, after successful validation, the S4T IoTronic delegation agent calls the `Delegation.sol` contract by using the Web3 client. A User identifier, an operation identifier, and IoT resource identifier are passed as parameters. In step 4, the `Delegation.sol` contract requests the roles granting access to the considered operation to the `Role.sol` contract. In step 5, such a list of roles is returned and the `Delegation.sol` contract checks if the user is associated with one of them for the considered IoT resource. In step 6, the `Delegation.sol` contract logs the result in the blockchain and finally, in step 7, it sends the result back to S4T IoTronic delegation agent that, in step 8, sends the result back to the S4T IoTronic conductor. Eventually, access to the user is granted or not depending on the result.

In order to create an audit trail for users to verify the behavior of the authorizing agent, we decided to create event-based logs. The logs created by OpenStack are event-based which encouraged us to adopt event-based logging. In event-based permission delegation, permissions are delegated to a specific user in the response of an event. This triggering of event initiates the logging process. Another strategy for logging includes query-based logging. In query-based permission delegation, the user requests/queries for permission on a resource from the owner. In contrast, the query causes the initiation of logging. The event-based logging enables the user to verify the decision communicated to him/her by the authorizing agent and thus, report any unfairness. This is a major challenge: since the logs are stored on the Blockchain, the access verification request has to wait till the mining is complete. The long waiting time for access request is highly undesirable and renders the system useless. To overcome this challenge, we looked at the concept of pending transactions in a transaction pool waiting to be mined. In the life-cycle of a transaction, when a transaction is selected for addition to the block and before the block is added to the Blockchain (consensus), the transaction is verified and added to the pending block. Once consensus is reached, the pending block is added to the Blockchain. Thus, we are capturing the pending block to know whether the user is having the access or not. We know that the verification of the transaction is quick but consensus takes time. Thus, using the pending block, we are able to reduce the response time of access request. The limitation to this approach being the possibility that the transaction fails due to reasons like not enough ether on originating account to perform transfer and pay for gas, or transaction gas consumption exceeding block gas limit. These issues can be handled by keeping enough balance in the account, and the operation of validation simple and bounded.

2.5.1 Data structures and interfaces

This subsection presents the design of the proposed solutions. It describes each solution in detail and presents the challenge that motivated the development of the approach following it. The description contains pseudo-code for three most important operations of the system namely insertion of delegation, deletion of delegation, and authenticating the access request. The description is followed by the theoretical analysis of time and space complexity in the average and worst case.

Data Structure 1: Data structure of AllowedOperations.

```
struct AllowedOperations {
    bytes32 id;
    bytes32 role_name;
    bytes32[] operations;
    uint index;
}

mapping ( bytes32 => AllowedOperations ) public roles;
bytes32[] public chkRole;
```

As per Figure 2.1, the smart contracts are designed in two parts. The first contract, named `Role.sol`, captures the functionality of RBAC systems wherein the information of operations associated to a particular role are stored on the blockchain. The second contract, named `Delegation.sol`, is responsible for associating the role with the user, allowing user to perform associated operations on the resource. It also stores information related to delegation of role, wherein an authorized user enables a requesting user to perform operations on the resource.

Data Structure 2: Data structure of DelegationInfo.

```
struct DelegationInfo {
    bytes32 id;
    bytes32 delegatingUserID;
    bytes32 beneficiaryUserID;
    bytes32 nodeID;
    uint index;
}

mapping (bytes32 => DelegationInfo) public delegations;
bytes32[] public chkDeleg;
```

The `AllowedOperations` (Data Structure 1) represents the data structure for storing information related to role. The structure stores a universally unique identifier (UUID) of the role, the name of the role, the operations that are allowed for the role, and an index. The index holds the position of the id in the `chkRole` array. The mapping `roles` holds information about a role with its id as the key. The array `chkRole` is used to confirm the existence of the role as the mapping data structure in Solidity initializes all the keys associated with it. The position of the UUID of the role in the `chkRole` is stored in the index field, thereby confirming the existence of the role. Similarly, `DelegationInfo` (Data Structure 2) represents the data structure for storing information related to delegation. The structure holds the delegation id, the id of the delegation providing temporary access, the id of the user receiving the access, the node (resource) id, the role assigned to the user and an index. All ids stored are UUIDs. Also, the index acts similar to the one in `Role.sol`. Except for the indices (which are of *integer* data type), most of the fields are *bytes32* as in Solidity the operations involving *bytes32* data type are computationally cheaper.

Algorithm 1: Algorithm to add delegation to the system

```
input : _id, _delegator_id, _beneficiary_id, _node_id, _roleid_benef
output: The delegation is added to the blockchain
1 if !isSubset ( _roleid_benef, delegations[ _delegator_id ].roleid ) then
2     exit;
3 else
4     temp_delegation ← delegation( _id, _delegator_id, _beneficiary_id,
5         _node_id, _roleid_benef );
6     index ← chkDeleg.push( _id );
7     temp_delegation.index ← index;
8     delegations[ _id ] ← temp_delegation;
```

The Algorithm 1 outlines the process to add a delegation to the system. The steps include: (i) checking if the user has authority to delegate the role. A check is made to confirm if the delegated role is a subset of the role of the delegator. (ii) adding the delegation to the Blockchain based on the output of step (i). The isSubset function takes two roles as input and checks if first role is a subset of the second role. The id of the role is passed as an input, and the function returns true if the role is a subset, otherwise it returns false. The time complexity of the algorithm is quadratic in nature, as each operation in a role needs to be verified against each operation of the other role. Finally, the addDelegation function adds the delegation to the Blockchain based on the output of the step (i). The time complexity of the algorithm depends on the number of delegations in the system. For few delegations in the system, the quadratic nature of isSubset function dominates and thus, the addDelegation runtime complexity exhibits' quadratic nature, proportional to maximum number of operations possible on a resource. For large number of delegations, the linear search for presence of delegation dominates the time complexity and thus, a linear time complexity is observed. The Algorithm 2 deletes a delegation from the system. Deletion of a parent delegation also results in deletion of all child delegations. The algorithm functions in the following manner: (i) the id of the delegation to be deleted is stored in an array, (ii) in each of next iterations, a scan is made over all the delegations to check if the parent of a

Algorithm 2: Algorithm to delete a single delegation from the system

```
input  : _id
output: A single delegation is deleted from the blockchain
1 index ← delegations[_key].index;
2 lastindex ← delegations[chkDeleg[chkDeleg.length]].index;
3 swap( chkDeleg[index], chkDeleg[lastindex] );
4 decrease chkDeleg length by 1;
5 delete lastindex to reset the structure;
```

delegation matches the id of the delegation to be deleted, (iii) finally, all the collected ids are deleted in a single iteration. Instead of deleting a delegation while searching, we are deleting delegation at the end to reduce the multiplicative effect in terms of time complexity. The storage of ids for deletion, in turn, introduces a linear space complexity. The function deleteSingleDelegation deletes a single delegation, whose id is passed as a parameter. The complexity of the algorithm is constant in both average and worst case. A behaviour worth mentioning is that at the time of deletion, the operation being performed on the resource will not be interrupted as the access control check is triggered based on events. Once the access is revoked, the next access control check will deny the operation.

Algorithm 3: Algorithm to delete a delegation tree from the system

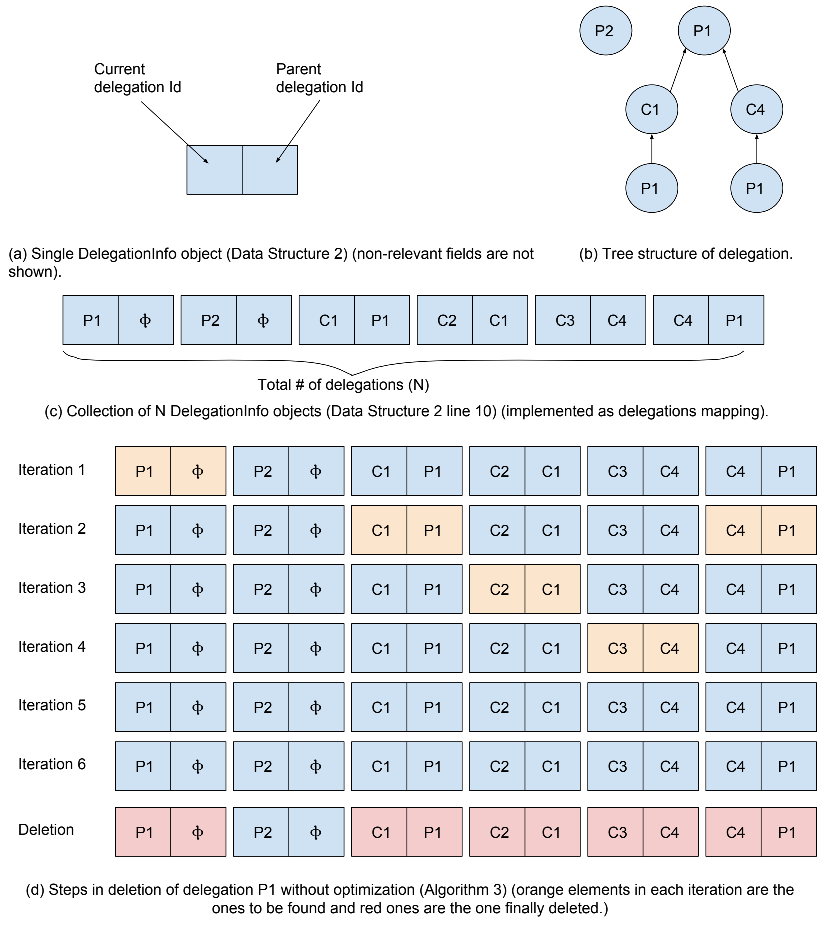
```
input  : _id
output: The delegation tree is deleted from the blockchain
1 Candidate.push(_id);
2 cnt ← 2;
3 for $i$ ← 1 to $N$ do
4     if delegations[chkDeleg[j]].delegatingUserID == tempId then
5         Candidate[cnt] ← delegations[chkDeleg[j]].id;
6         cnt ← cnt + 1;
7 for $j$ ← 1 to $N$ do
8     deleteSingleDelegation( Candidate[j] );
9 Candidate ← [ ];
```

The algorithm to delete an entire delegation tree is presented in Algorithm 3, in which all the delegations are deleted one by one. The decision to delete the delegations separately rather than during identification is taken to reduce the multiplicative effect to an additive effect. The complexity of the algorithm to delete a delegation is presented in Figure 2.2. The (a) part of the figure presents the basic architecture of a node. It consists of delegation id and id of the delegator in addition to other information. Part (c) consists of a sample snapshot of the system at that instant with (b) representing the tree structure of the system at that instant. Part (d) emphasizes the process involved in finding the delegation to be deleted at each iteration and deleting all the identified delegations in single step.

Finally the algorithm to check if the user has access to perform a particular operation is

Algorithm 4: Algorithm to check if user is having access to perform operation

```
input  : _beneficiaryid, _nodeid, _operationid
output: boolean true or false
1 N ← chkRole.length;
2 for $i$ ← 1 to $N$ do
3     if _beneficiaryid == delegations[i].beneficiaryid and _nodeid ==
        delegations[i].nodeid then
4         if checkRole( delegations[i].roleid, _operationid ) then
5             log(_beneficiaryid, _nodeid, _operationid, true);
6             return true;
7 log(_beneficiaryid, _nodeid, _operationid, false);
8 return false;
```



(a) Single DelegationInfo object (Data Structure 2) (non-relevant fields are not shown).

(b) Tree structure of delegation.

(c) Collection of N DelegationInfo objects (Data Structure 2 line 10) (implemented as delegations mapping).

(d) Steps in deletion of delegation P1 without optimization (Algorithm 3) (orange elements in each iteration are the ones to be found and red ones are the one finally deleted.)

Figure 2.2: Complexity analysis of first method.

presented in Algorithm 4. The function makes use of a subroutine to check whether a particular role includes the operation specified. Based on the userid and nodeid, the corresponding roleid is found from the delegation and the presence of operation is validated in that role. The process is repeated till either the role with the operation is found or all the delegations for that particular userid and nodeid combination are exhausted. The run time of the algorithm is in the order of the number of delegations in the system.

Table 2.1 present the summary of time and space complexity of each algorithm in average and worst case. We see that for isUserAllowed algorithm, the complexity is either quadratic in terms of number of operations associated with a role or linear in terms of number of delegations. In fact, the early part of algorithm execution is quadratic in nature as the number of delegations are comparable to or less than the square of the operations associated with the role. Once this threshold is passed, the nature changes to linear.

		addDelegation	**deleteDelegation**	**isUserAllowed**
Time	worst	$O(N)$	$O(N^2)$	$O(K^2)$ or $O(N)$
	average	$\theta(N)$	$\theta(N^2)$	$\theta(K^2)$ or $\theta(N)$
Space	worst	$O(1)$	$O(N)$	$O(1)$
	average	$\theta(1)$	$\theta(N)$	$\theta(1)$

Table 2.1: Time and space complexity analysis for the first approach.

Data Structure 3: Modified data structure of DelegationInfo.

```
...
        bytes32 firstChild;
        bytes32 firstSibling;
...
```

The deleteDelegation algorithm is the only one requiring additional array to store the delegations to be deleted. We see that the deletion algorithm has quadratic time complexity. This limitation is significant in a Smart City scenario where the number of delegations increases rapidly and the creation and deletion of delegations is frequent. This brings us to the next approach. The second approach is similar in terms of creating a delegation or checking the access request of a user. But in order to efficiently handle the deletion of a delegation tree, we utilize two pointers, namely firstSibling pointer and firstChild pointer. To delete an entire delegation tree, the previous algorithm took quadratic time complexity. One solution to this problem is to store the pointers to all child delegations in the root node. But in the worst case, the storage space required can go up with the number of delegations in terms of space complexity. Thus, we used two pointers in each node. Each node stores one pointer to the first child and one pointer to the first sibling. In this way, the space complexity is maintained linear. Thus, to efficiently delete a delegation, we need to traverse down the tree using the firstChild and firstSibling pointers (Data Structure 3). In any iteration, we add the node to be deleted to a temporary array. We delete an element from the temporary array on a FIFO basis. Before deletion, we check for the firstChild and firstSibling pointers of the node to be deleted and if found, then push them into the temporary array. We repeat this process till there are no more nodes to be deleted in the array. This entire process is completed by a single pass on the delegation array and thus, we reduce the time complexity to linear. Also, we need to adjust the pointers in case a partial tree is deleted. If the first child of a tree is deleted, then we make the first sibling of the node to be deleted as the first

Algorithm 5: Algorithm to delete the delegation tree in linear time

```
input  : _id
output: delete the entire delegation tree

1  tempArray[0] ← _id;
2  cnt ← 1;
3  for i ← 1 to N do
4      tempId ← tempArray[i];
5      if tempId != 0 then
6          if delegations[tempId].firstChild != 0 then
7              tempArray[cnt] ← delegations[tempId].firstChild;
8              cnt ← cnt + 1;
9          if i != 0 then
10             if delegations[tempId].firstSibling != 0 then
11                 tempArray[cnt] ← delegations[tempId].firstSibling;
12                 cnt ← cnt +1;
13     else
14         break;
15 if delegations[_id].delegatingUserID != 0 then
16     tempIdOld ← delegations[_id].delegatingUserID;
17     tempIdNew ← delegations[tempIdOld].firstChild;
18     while tempIdNew != _id do
19         tempIdOld ← tempIdNew;
20         tempIdNew ← delegations[tempIdNew].firstSibling;
21     if tempIdOld == delegations[_id].delegatingUserID then
22         delegations[tempIdOld].firstChild ← delegations[tempIdNew].firstSibling;
23     else
24         delegations[tempIdOld].firstSibling ← delegations[tempIdNew].firstSibling;
25 for j ← 1 to tempArray.length do
26     deleteSingleDelegation(tempArray[j]);
```

child of the parent. Otherwise, we need to adjust the firstSibling pointer to point to the next sibling. The algorithm is presented in Algorithm 5. The complexity of the algorithm to delete a delegation is presented in Figure 2.3. The (a) part of the figure presents the basic architecture of a node. It consists of delegation id and id of the first child and the first sibling delegation in addition to other information. Part (c) consists of a sample snapshot of the system at that instant with (b) representing the tree structure of the system at that instant. Part (d) emphasizes the process involved in finding the delegation to be deleted in single iteration and deleting all the identified delegations in next iteration.

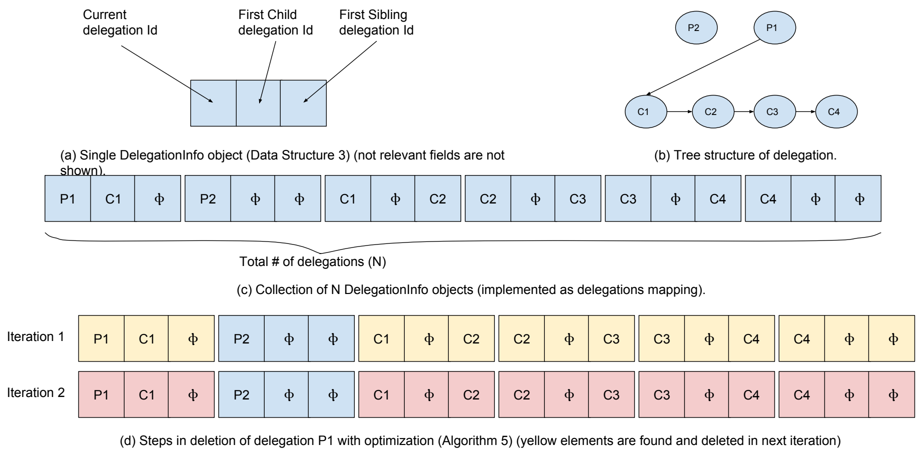

Current delegation Id
First Child delegation Id
First Sibling delegation Id
(a) Single DelegationInfo object (Data Structure 3) (not relevant fields are not shown).
P2
P1
C1
C2
C3
C4
(b) Tree structure of delegation.
P1 C1 ϕ P2 ϕ ϕ C1 ϕ C2 C2 ϕ C3 C3 ϕ C4 C4 ϕ ϕ
Total # of delegations (N)
(c) Collection of N DelegationInfo objects (implemented as delegations mapping).
Iteration 1
Iteration 2
(d) Steps in deletion of delegation P1 with optimization (Algorithm 5) (yellow elements are found and deleted in next iteration)

Figure 2.3: Complexity analysis of second method.

		addDelegation	**deleteDelegation**	**isUserAllowed**
Time	worst	$O(N)$	$O(N)$	$O(K^2)$ or $O(N)$
	average	$\theta(N)$	$\theta(N)$	$\theta(K^2)$ or $\theta(N)$
Space	worst	$O(1)$	$O(N)$	$O(1)$
	average	$\theta(1)$	$\theta(N)$	$\theta(1)$

Table 2.2: Time and space complexity analysis for second approach.

Data Structure 4: Modified data structure of AllowedOperations.

```
...
mapping ( bytes32 => bool ) operation;
bytes32[] ops;
...
```

In Table 2.2, we present the summary of time and space complexity of each algorithm in average and worst case. The deleteDelegation algorithm is this case is linear and thus, the time complexity is reduced from quadratic to linear. The algorithm still requires additional array to store the delegations to be deleted. We see that the time complexity of isUserAllowed is similar to last approach. In scenarios, where the number of operations associated to role are comparable to number of delegations, we see a quadratic time complexity and thus, the approach is not suitable for highly interactive environments. This leads to the next approach.

The final approach is centered around reducing the complexity of the test operation isUserAllowed. We achieve this by using the mapping structure to store the operations of a role as a key-value pair (Data Structure 4 line 1). The retrieval of a single value takes constant time and thus, the isSubset function's complexity reduces from the order of K^2 to the order of K. The resulting change in the isSubset algorithm is presented in Algorithm 6.

The mapping data structure consists of the key-value pair. Leveraging the mapping data

Algorithm 6: Algorithm to find if a role is subset of other

```
input  : _roleid_benef, _roleid_deleg
output: boolean true or false
1  k1 ← $roles[\_roleid\_benef].ops.length$;
2  k2 ← $roles[\_roleid\_deleg].ops.length$;
3  if k1 > k2 then
4      return false;
5  else
6      cnt ← 0;
7      for $i \leftarrow 1$ to $k1$ do
8          if roles[roleid_deleg].operation[roles[roleid_benif].ops[i]] then
9              cnt ← cnt + 1;
10 if cnt == k1 then
11     return true;
12 else
13     return false;
```

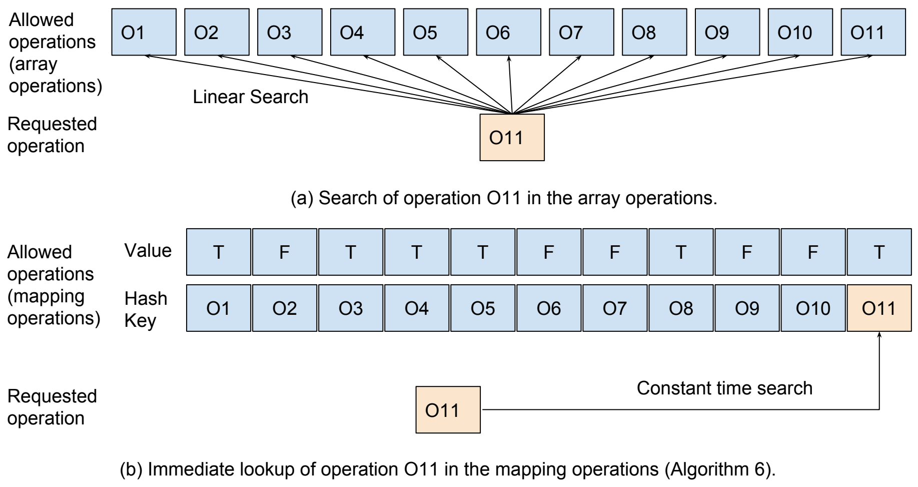

Allowed operations (array operations)
O1
O2
O3
O4
O5
O6
O7
O8
O9
O10
O11
Linear Search
Requested operation
O11
(a) Search of operation O11 in the array operations.
Allowed operations (mapping operations)
Value
T
F
T
T
T
F
F
T
F
F
T
Hash Key
O1
O2
O3
O4
O5
O6
O7
O8
O9
O10
O11
Requested operation
O11
Constant time search
(b) Immediate lookup of operation O11 in the mapping operations (Algorithm 6).

Figure 2.4: isUserAllowed advantage in terms of search.

structure, it can be found that a role is a subset of another role in a single pass. This reduces the overall complexity of isUserAllowed which uses this subroutine. The reduction is effective if the number of delegation is comparable to the number of operations associated with the role. Beyond that, the complexity is linear in any case. The comparison among searching the access policy in an array and searching the access policy using the mapping data structure of the Solidity programming language is presented in Figure 2.4. The (a) part of the figure represents the linear search algorithm in an array. Part (b) represents the advantage of the mapping data structure of Solidity. The mapping data structure acts as a hash-map data structure. Thus, the search time of the algorithm reduces to a constant complexity.

		addDelegation	**deleteDelegation**	**isUserAllowed**
Time	worst	$O(N)$	$O(N)$	$O(N)$
	average	$\theta(N)$	$\theta(N)$	$\theta(N)$
Space	worst	$O(1)$	$O(N)$	$O(1)$
	average	$\theta(1)$	$\theta(N)$	$\theta(1)$

Table 2.3: Time and space complexity analysis for third approach.

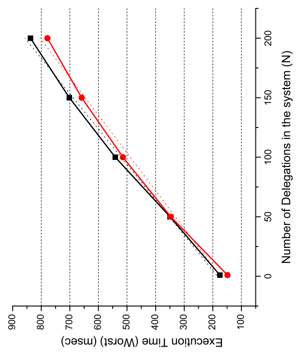
In Table 2.3, we report the time and space complexity of each algorithm in the average and the worst case. The isUserAllowed algorithm in this case is linear and thus, the time complexity is reduced from quadratic to linear. The approach thus, becomes suitable for highly interactive environments.

2.6 Results and Discussion

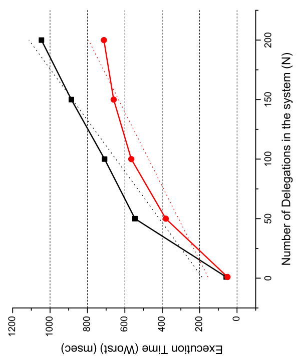
This section details the simulation setup and presents the results obtained from the simulation. The section also presents advantages as well as limitations of the obtained results.

The performance evaluation of the proposed model is based on the measurement of execution time. The execution time can be captured in two possible ways: the timestamp of the block and through external timekeeping. The limit of the block timestamps is that they depend on the miner and thus, can be inaccurate. We are recording the timestamp on request initiation and request fulfillment. The execution time is calculated as the difference of the aforementioned two timestamps.

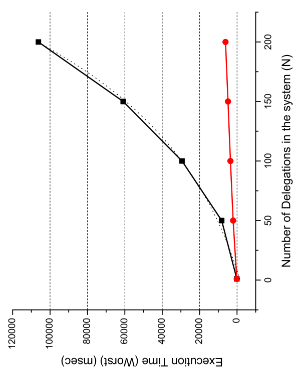
The experiments are carried out on a local blockchain (Ganache), as well as on public test network (Rinkeby). The Rinkeby test network is based on a proof-of-authority (POA) consensus algorithm. The experiments consist of three operations for which the execution time is recorded: inserting delegation into the system, deleting the delegation from the system, and checking the access rights of a user. The process of inserting, deleting and access checking is repeated 100 times to average out the randomness of the mining process. Based on the structure of the algorithms, an average and a worst case is identified for each of the proposed approaches. For each of the identified worst and average case, the operations are performed several times and the results are tabulated. Finally, the average and worst cases for each of the operation and in each approach is compared using graphs.



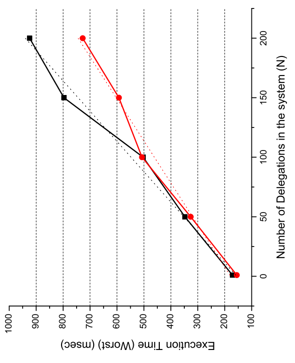
(a) Insert approach one



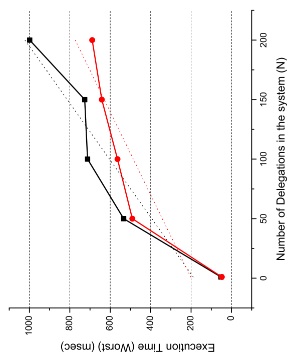
(b) Check approach one



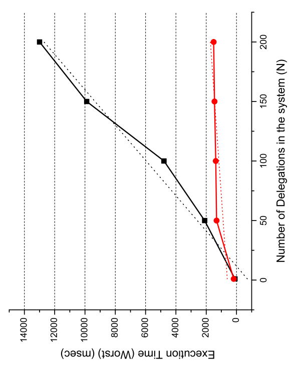
(c) Delete approach one



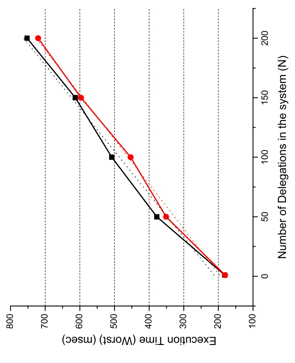
(d) Insert approach two



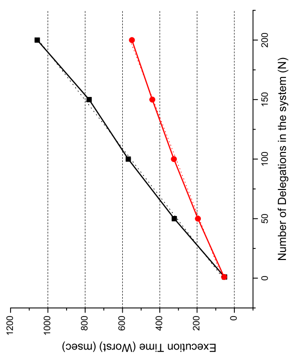
(e) Check approach two



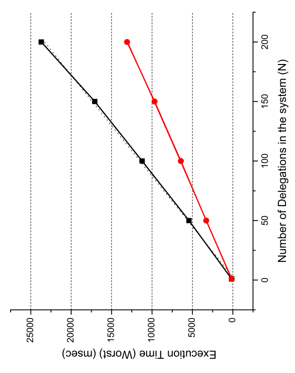
(f) Delete approach two



(g) Insert approach three



(h) Check approach three



(i) Delete approach three

Figure 2.5: Graphs of experimental results of three approaches in Ganache.

Operation	No. of Delegations	First Approach	Second Approach	Third Approach
Insertion	1	175.31±4.3	171.39±4.87	181.61±4.41
	50	349.62±7.53	347.88±11.28	377.76±8.91
	100	541.24±9.53	502.94±13.75	507.87±10.07
	150	701.7±18.89	796.95±14.22	613.05±17.43
	200	837.52±8.47	924.36±19.82	752.33±14.38
Deletion	1	111.66±3.27	102.8±3.15	148.24±3.89
	50	8191.9±122.9	2104.47±31.88	5425.56±34.83
	100	29401.84±240.84	4784.6±73.75	11224.12±56.92
	150	60869.42±377.09	9888.82±251.06	17081.85±86.41
	200	106246.36±596.03	12996.66±144.93	23701.65±302.98
Check	1	57.55±2.1	51.78±1.7	52.73±2.25
	50	545.86±5.04	533.68±4.15	322.3±5.27
	100	708.45±10.35	712.26±8.41	569.17±6.06
	150	886.13±7.13	726.35±27.19	779.02±5.61
	200	1046±16.49	998.74±20.89	1056.97±3.42

Table 2.4: Experimental results in worst case of three approaches in Ganache (msec).

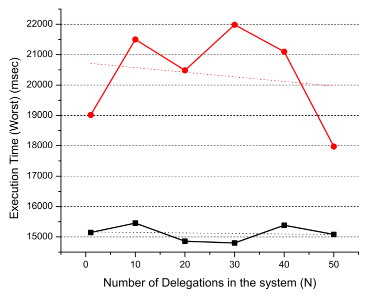
Figure 2.5 consists of nine plots where each plot compares worst and average case of each operation in each approach. The figure summarizes the results for Ganache. Each row of the figure represents an approach; for example, like the first row represents approach one. Each plot is plotted between the number of delegations in the system before the operation is executed and the execution time of the operation. Each execution value is averaged out for 100 iterations. The first column of the plot shows the delegation insertion for all the three approaches. We can see that Plots 2.5a, 2.5d, and 2.5g exhibit a linear trend, i.e. the delegation insertion varies linearly with the number of delegations in the system. The second column represents the operation of access check. We observe that Plots 2.5b and 2.5e show quadratic growth for the initial number of delegations, while linear for the remaining. The behavior is as per the theoretical analysis we presented in the previous section. Plot 2.5h shows a linear variation of the execution time of access check with the number of delegations in the system. The last column shows the deletion of delegation for all the approaches. We observe that Plot 2.5c shows that the execution time of the deletion process varies quadratically with the number of delegations in the system. In case of Plots 2.5f and 2.5i, we observe a linear trend between the execution time and the number

Operation	No. of Delegations	First Approach	Second Approach	Third Approach
Insertion	1	148.45±5.46	155.23±4.9	180.74±4.39
	50	346.94±7.22	326.47±7.78	351.12±7.15
	100	514.28±10.36	506.98±11.91	453.26±7.74
	150	658.56±19.25	592.87±9.07	596.72±9.89
	200	778.41±8.85	727.39±10.83	720.45±10.36
Deletion	1	117.78±2.23	182.49±13.24	147.76±4.02
	50	2045.06±8.86	1314.45±20.44	3290.35±25.52
	100	3515.77±37.68	1369.58±57.53	6431.63±38.39
	150	4823.59±33.43	1450.21±55.25	9686.29±54.78
	200	6125.66±32.55	1509.4±39.28	13087.61±96.89
Check	1	50.73±1.35	47.84±1.6	55.08±1.64
	50	380.95±2.31	490.54±11.72	195.41±4.67
	100	565.53±7.51	564.16±6.22	324.43±4.76
	150	659.65±6.49	641.7±3.8	440.01±5.02
	200	712.11±4.74	689.24±15.05	549.3±5.35

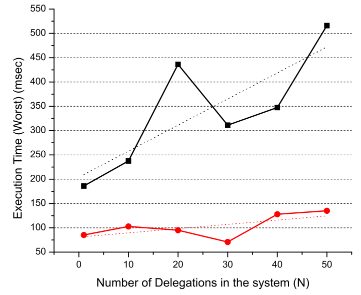
Table 2.5: Experimental results in average case of three approaches in Ganache (msec).

of delegations in the system. Table 2.4 and Table 2.5 lists the worst and average execution time of all the operations for all the approaches with 95% confidence range. All the values are in milliseconds. We can observer a relatively small confidence interval thus, strengthening our theoretical analysis.

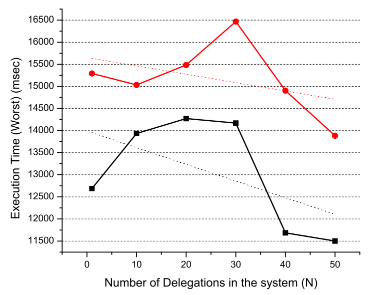
Similarly, Figure 2.6 shows the variation of the execution time of operations with respect to the number of delegations in the system for Rinkeby. Plots in the first column represent the insertion of delegation in the system, plots in the second column represent the check access operation, and plots in the third column represent the deletion of delegation from the system for all the three approaches. We observe that Plots 2.6b, 2.6e, and 2.6h are of quadratic nature as the number of delegations in the system are comparable to the number of roles. Thus, in such situations the time complexity of the system is quadratic as we mentioned in the theoretical analysis. For the Plots 2.6a, 2.6d, 2.6g, 2.6c, 2.6f, and 2.6i, we observe that the execution time of the system is the same as the block confirmation time of the blockchain. On the average, the plots are around the average block confirmation time, which is 14 sec for Rinkeby. Table 2.7 lists the average execution time of all the operations for all the approaches, with a 95% confidence range. All the values are in milliseconds.



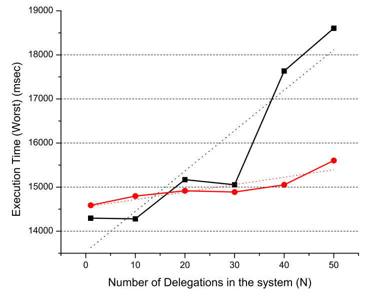
(a) Delegation insertion using approach one.



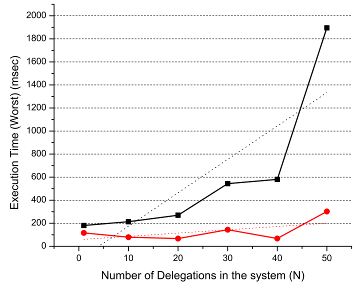
(b) Check access request using approach one.



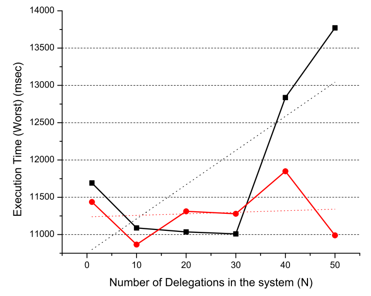
(c) Delete delegation using approach one.



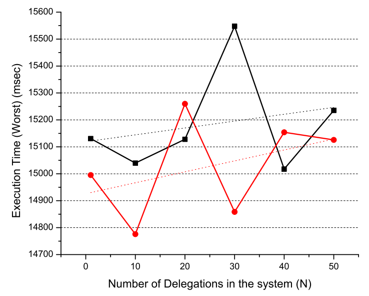
(d) Insert approach two



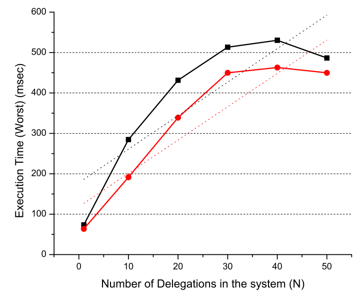
(e) Check approach two



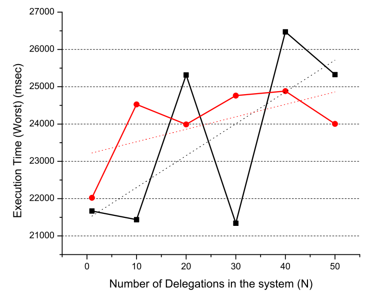
(f) Delete approach two



(g) Insert approach three



(h) Check approach three



(i) Delete approach three

Figure 2.6: Graphs of experimental results of three approaches in Rinkeby.

Operation	No. of Delegations	First Approach	Second Approach	Third Approach
Insertion	1	15.15±0.47	14.3±0.47	15.13±0.47
	10	15.46±0.8	14.28±0.67	15.04±0.36
	20	14.86±0.44	15.17±0.43	15.13±0.45
	30	14.8±0.38	15.05±0.28	15.55±0.54
	40	15.38±0.46	17.63±1.14	15.02±0.18
	50	15.08±0.34	18.6±1.52	15.24±0.49
Deletion	1	12.69±1.47	11.69±1.1	21.67±1.72
	10	13.94±2.47	11.09±1	21.44±2.17
	20	14.27±1.56	11.04±0.9	25.32±2.36
	30	14.17±1.61	11.01±0.99	21.34±1.82
	40	11.69±1.28	12.84±1.4	26.47±2.96
	50	11.5±1.13	13.77±1.51	25.33±2.03
Check	1	0.19±0.1	0.18±0.04	0.07±0.03
	10	0.24±0.1	0.21±0.04	0.28±0.05
	20	0.44±0.09	0.27±0.07	0.43±0.04
	30	0.31±0.06	0.54±0.08	0.51±0.06
	40	0.35±0.07	0.58±0.08	0.53±0.05
	50	0.52±0	1.9±0.79	0.49±0.08

Table 2.6: Experimental results in worst case of three approaches in Rinkeby (sec).

From the execution graph of the first approach, we can see that the insertion of delegation in Rinkeby takes on an average constant amount of time and is independent of the number of delegations present in the system. It is understandable since the execution time of the actual operations involved in the insertion of the delegation is much less compared to the time required to reach the consensus and acceptance of the transaction. In case of checking the access of a user, we can see a linear trend in the observed execution time. In this case, we do not wait for the confirmation of the execution to improve the response time of the operation. In case of deletion as well, we can see a constant trend on the average, owing to substantial transaction confirmation time.

The system suffers from some limitation because of the underlying blockchain platform. The Rinkeby network has a gas limit of 7.7 million at the time of writing this thesis. The concept of gas gives Ethereum the Turing completeness feature. This limits the depth of the

Operation	No. of Delegations	First Approach	Second Approach	Third Approach
Insertion	1	19.01±1.35	14.59±0.24	15±0.64
	10	21.5±1.5	14.8±0.33	14.78±0.55
	20	20.48±1.47	14.92±0.44	15.26±0.55
	30	21.98±1.63	14.89±0.52	14.86±0.36
	40	21.1±1.47	15.05±0.41	15.15±0.46
	50	17.98±1.21	15.6±0.69	15.13±0.37
Deletion	1	15.29±1.38	11.44±0.95	22.02±2.25
	10	15.03±1.46	10.87±0.88	24.53±1.91
	20	15.48±1.81	11.31±1.38	23.99±2.25
	30	16.47±1.88	11.28±0.9	24.76±3.01
	40	14.9±1.43	11.85±1.53	24.88±1.81
	50	13.88±1.5	10.99±1.05	24.01±2.51
Check	1	0.09±0.05	0.12±0.04	0.06±0.02
	10	0.1±0.05	0.08±0.05	0.19±0.07
	20	0.1±0.05	0.07±0.03	0.34±0.08
	30	0.07±0.04	0.14±0.05	0.45±0.04
	40	0.13±0.06	0.07±0.03	0.46±0.03
	50	0.14±0.04	0.3±0.07	0.45±0.04

Table 2.7: Experimental results in average case of three approaches in Rinkeby (sec).

delegation operation as the number of individual delete operations will be dependent on the gas limit. If the gas requirement exceeds the gas limit of the network, the delegation deletion won't be successful. The same is true for the Ropsten test network for which the gas limit is 4.7 million at the time of writing this thesis. The limitation we describe here has more to do with the underlying network and less with the implementation. But in case we need to use Ropsten or Rinkeby in any case, we can use the concept of *pagination* where you divide the list of delegations to be deleted into small batches and delete them one batch at a time.

2.7 Conclusions

In this chapter, we proposed an enhancement to an IoT-Cloud solution in the form of a decentralized design for resource access authorization and delegation duties. The proposal featured blockchain as a technological building block and smart contracts as the main engine for trustless decentralization and independent audit of operations. We discussed the use cases presenting the real world problems having different characteristics. We described the design and implementation details of the contracts customized to the presented use cases. We also performed a theoretical analysis of the proposed algorithms. We performed multiple experiments to observe the time and space complexity of the different approaches. We found that the insertion and deletion of the delegation are dependent on the block confirmation time, while checking user access confirms theoretical analysis. Finally, we discussed the limitations related to the underlying platform and a possible methodology to overcome them. We conclude this chapter by establishing blockchain as a viable solution to address the access control and delegation needs of an IoT framework.

Chapter 3

Access control and delegation for resource constrained devices

In the previous chapter, we described the proposal to address access control and delegation in resource sufficient IoT devices. Powerful devices make it possible to run Blockchain-based computations to authorize access to the devices. In this chapter, we address the problem of access control in resource-constrained devices. Due to limited capability of the devices, a Blockchain-based solution becomes unfeasible. Also, a highly distributed smart environment requires scalable architectures to support a large number of stakeholders that share Internet of Things (IoT) resources and services. We focus on authorization solutions that regulate access of users to smart objects and consider scenarios where a large number of smart objects owners want to share the resources of their devices in a secure way. A popular solution is to delegate third parties, such as public Cloud services, to mediate authorization procedures among users and smart objects. This approach has the disadvantage of assuming third parties as trusted proxies that guarantee correctness of all authorization procedures. In this section, we propose a system that allows to audit authorizations managed by third parties, to detect and expose their misbehaviors to users, smart objects owners and, possibly, to the public. The proposed system is inspired by the transparency projects used to monitor Web Certification Authorities, but improves over existing proposals through a twofold contribution. First, it is specifically designed for IoT devices, provided with little resources and distributed in constrained environments. Second, it complies to current standard authorization protocols and available open-source software, making it ready to be deployed. Finally, we apply the proposed solution to industrial IoT use-case to evaluate the efficiency of the proposal.

3.1 System and threat models

We outline the scope and guarantees of our proposal. First, we describe a reference IoT scenario that motivates the design of the system (Section 3.1.1). Second, we model the scenario with regard to established authorization frameworks (Section 3.1.2). Finally, we discuss the security threats and guarantees of the proposed system (Section 3.1.3).

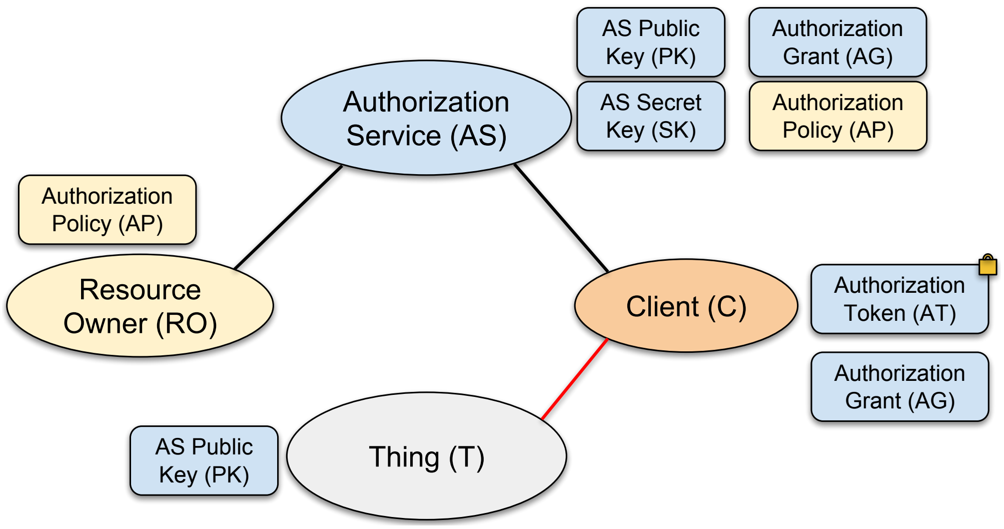


Figure 3.1: Reference architecture for delegated authorizations.

3.1.1 Reference scenario

We consider a real-world example represented by an online rental service that acts as a broker between room owners and potential guests. To improve the flexibility of the service, each room is equipped with a smart lock that is compliant with standard WiFi technologies available on any smartphone. Room owners subscribe to the service and delegate it with the capability of allowing guests to open the room door by interacting with the associated smart lock through their smartphones. Once an agreement is in place (e.g., upon successful payment), the guest should be authorized to open the smart lock in an unsupervised way during the agreed time frame.

Let us assume that the guest tries to open the smart lock during her renting period. The interaction between the guest, the service, and the lock is composed by three phases. First, the guest interacts with the service, issuing a request to open the lock. Second, the service validates the request and decides whether the guest should be allowed to open the lock. The third operation depends on the outcome of the second one: if the service decides that the guest is authorized, it grants her access to the room by releasing the due authorization material that allows her to open the smart lock; if the guest is not authorized, the service denies the guest access by not releasing the authorization material.

The aim of the proposed system is to allow both the room owners and the guests to detect whether the rental service is behaving correctly, allowing (1) the owner to detect illegitimate authorizations, and (2) both the owner and the guests to detect illegitimate denied authorizations.

3.1.2 Reference architecture and operations framework

We describe the architecture for delegated authorizations by referring to Figure 3.1. We consider four roles: *Things* (T), *Resource Owners* (RO), *Clients* (C) and *Authorization Services*

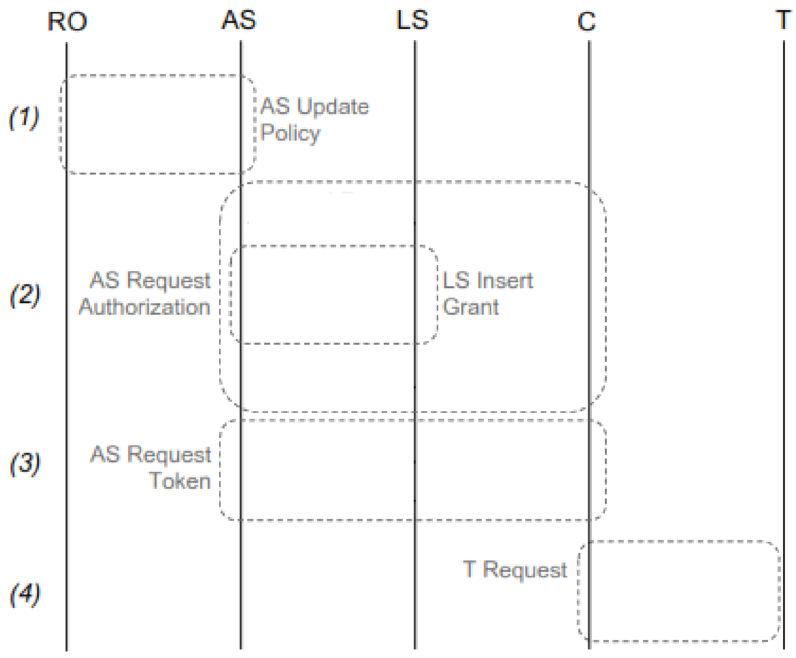


Figure 3.2: Operation workflow of reference architecture.

(*AS*). *Things* represent cyber-physical IoT devices provided with sensors and/or actuators that can communicate locally (e.g., via wireless local and/or personal area networks), but might not have reliable Internet connectivity (e.g., very slow, discontinuous, expensive, or no connectivity at all). *Resource owners* represent entities (e.g., a person, a company) that regulate accesses to things. They have access to devices with moderate capabilities (e.g., modern smart-phones, personal computers) and to reliable Internet connectivity, but do not own large infrastructures to deploy highly available and scalable services. *Authorization services* represent highly available and scalable services (e.g., Web services deployed on a public Cloud infrastructure) that act on behalf of resource owners to control access to things. *Clients* represent entities that need to get access to things (e.g., information provided by thing-hosted sensors, access to a facility controlled by thing-hosted actuators). They are authorized by resource owners to access things and must interact with authorization services to obtain the authorization material required by things to validate their requests.

The architecture includes five types of data: *Authorization Policies* (*AP*), *Authorization Grants* (*AG*), *Authorization Tokens* (*AT*), and *Authorization Service Secret* and *Public Keys* (*SK*, *PK*). *Authorization policies* are rules that regulate access to things by clients. The semantic of the acceptable rules depend on the adopted access control model and are orthogonal to the authorization protocol. *Authorization grants* are authentication tokens used by authorization services to validate clients requests. They are reference tokens, that is, they are opaque values with no implicit information that refers to authoritative information stored within the database of the authorization service (e.g., they could be implemented as long unique identifiers chosen at random). Authorization grants usually have long expiry times and can be easily revoked before-hand. *Access tokens* are authentication tokens used

Server	**Client**	**Operation interface**	**Description**
AS	RO	$\{\text{'}accept\text{'}, \text{'}reject\text{'}\} \leftarrow UpdatePolicy\,(AP)$	Update authorization policy
	C	$\{AG, \text{'}reject\text{'}\} \leftarrow Authorization\,(AP)$	Request an authorization grant AG for the authorization policy AP
		$\{AT, \text{'}reject\text{'}\} \leftarrow Token\,(AG)$	Request an access token AT with regard to the authorization grant AG
T	C	$result \leftarrow Request\,(data, AT)$	Access the thing by using the access token AT and send payload *data*.

Table 3.1: Plaintext operations framework of the reference architecture.

by things to validate clients requests. They are self-contained cryptographic tokens that include all the due information to validate a request without accessing any database (e.g., a JSON Web Token). As they cannot be easily revoked before-hand, access tokens usually have short expiry times. *Authorization service secret and public keys* are cryptographic keys used to sign and verify access tokens. While the secret key is only known by the authorization service, the public key is public information that is known by all parties.

We consider that the parties deploy network services as described in Table 3.1, where: the first and second columns (*server* and *client*) show the parties that offer and that use the service, respectively; the third column (*operation interface*) shows the interface of the service, including input and output data; the fourth column (*description*) describes what is the purpose of the operation.

The workflow of the operations that allow a client to access a thing is as following (Figure 3.2). *(1)* The resource owner inserts a new authorization policy AP_{RO} at the authorization service by using the *UpdatePolicy* service, that answers with the due return value (*'accept'*) to confirm the acceptance of the policy. *(2)* A client requests an authorization grant to access a thing by sending an authorization policy AP_C to the authorization service by using the *Authorization* routine. If the requested authorization AP_C complies to the authorization policies AP_{RO} released by the resource owner, the authorization service answers with an authorization grant AG. *(3)* The client uses the authorization grant AG to request an access token AT to the authorization service by using the *Token* routine. The authorization service will accept the request only if the authorization grant has not expired and has not been revoked. *(4)* The client issues a request to the thing by sending the (request) *data* and the access token AT with the *Request* operation. The thing approves the request only if the access token is valid.

We note that the proposed model represents an instance of the OAuth2 standard for delegated authorizations (85) and related extensions for constrained environments (48). As a minor variant, we denote as *things* the *resource servers* of the OAuth2 framework to highlight that they are devices characterized by limited resources and connectivity. As a major difference, while in the standard frameworks authorization and resource servers are controlled by the same authority, in the proposed model things are controlled by resource owners and are independent of authorization services. As a result, our proposal shares many design choices with the existing frameworks, but also provides additional guarantees that are driven by the additional system requirements and security threats.

3.1.3 Threat model and security guarantees

We assume that resource owners, authorization services and clients have known identities, and can establish secure and mutually authenticated connections by using any standard Web protocols for secure communications and credentials systems. Our proposal does not limit the adoption of any of these solutions. We do not assume that things and clients are able to establish secure channels in local networks. In case of non-secure channels, they must use the due cryptographic protocols to validate access tokens and, optionally, to protect the security of the application data. These are design choices that are orthogonal to our proposal. Finally, we assume that things know the public keys of the authorization services. As an example, public keys could be flashed by an OEM at the factory or configured by an admin in the local network.

The reference architecture guarantees security of the system against all known attacks operated by clients (e.g., a client that might try to access a thing illegitimately) and by external attackers (e.g., impersonation attacks) in the considered security model. For a comprehensive discussion about these attacks and about additional technicalities for their correct implementation the reader can refer to (48) and (86).

We distinguish from any standard authorization framework by considering weaker security assumptions, where parties may behave dishonestly by not operating all protocols correctly. First, we assume that authorization services could issue authorizations that do not comply with the policies defined by the resource owners, that is, they may illegitimately deny or release an authorization to a client that do not comply with the authorization policies defined by the resource owners. The proposed system guarantees that a client (or a resource owner) is able to detect these misbehavior and publicly accuse the authorization service by showing the due cryptographic proofs. A second security threat rises as a consequence of this security guarantee, that is, the possibility for the authorization service of being accused illegitimately by other parties (82; 87). Our proposal considers this threat and propose mechanisms that allow authorization services to defend themselves from false accusations.

3.2 Proposed system for authorizations transparency in IoT environments

We describe the proposed system for auditable authorizations in IoT environments. First, we describe the adopted access control model (Section 3.2.1). Second, we outline the architecture and operations framework of the system (Section 3.2.2). Third, we give details on the main auditable authorization protocol (Section 3.2.3) and for verifying the correct behavior of the authorization service (Section 3.2.4). We conclude by comparing the proposal to other approaches (Section 3.3.1).

3.2.1 Access control model and notation

We describe the details of the types of data adopted by the reference architecture proposed in Section 3.1.2 for a discretionary access control model. For ease of presentation, we consider a simple model that does not distinguish access privileges (e.g., read, write) and multiple resources available on a thing. That is, a client authorized to access a thing can operate any type of operation on all its resources. This model does not pose limitations to many IoT scenarios, such as the smart-lock motivating scenario described in Section 3.1.1. Note that more complex access control models, such as fine-grained attribute-based models, can be used as-well without limitations.

We represent an access policy as a tuple $AP = \langle AR, tn, V \rangle$, where AR denotes an access rule, tn denotes the issue time of the authorization policy and V denotes the validity period of the rule. Each access rule identifies the clients that can access to a thing as the tuple $AR = \langle C, T \rangle$, where client C can access thing T. The validity period defines when the access rule is to be considered valid and is represented as $V = \langle nb, na \rangle$, where nb and na denote the *not before* and *not after* time instants that define the time range, similarly to x509 Web certificates.

An authorization grant, that is a reference token, is implemented as a unique string identifier $AG = id$. An access token is a cryptographic bearer token defined as the tuple $AT = \langle AP, V, \sigma_{AT} \rangle$, where AP is a due authorization policy that authorizes the client to access a thing, V is the token validity period (with the same semantics described in the case of an access rule), and $\sigma_{AT} = sign_{SK_{AS}}(AP, V)$ is a proper signature of the authorization policy with the secret key of the authorization service.

3.2.2 Architecture overview

We describe the proposed architecture by referring to Figure 3.3. It extends the reference architecture for delegated authorizations (see Section 3.1, Figure 3.1) by introducing an additional role, the *Authenticated Log Service* (*LS*), that represents a service that stores information about authorizations granted by the authorization service. This service is hosted by an additional authority, external to resource owners, authorization services and clients, at an highly available Web service. The service allows the authorization service to insert information about authorizations, while other parties (and possibly the public) can query it. All information stored within the service is authenticated via *authenticated data structures*, allowing to detect any modifications on the data by a malicious service.

The architecture includes all the types of data managed in the reference architecture: *authorization policies*, *authorization grants* and *access tokens*. Moreover, resource owners, authorization services, clients and authenticated log services maintain public and secret keys, that we denote as $\langle PK_{RO}, SK_{RO} \rangle$, $\langle PK_{AS}, SK_{AS} \rangle$, $\langle PK_C, SK_C \rangle$ and $\langle PK_{LS}, SK_{LS} \rangle$, respectively. We assume that each secret key is only known by its owner, while public keys are known by all parties. Finally, the architecture includes *cryptographic attestations* that authenticate the information exchanged by the parties.

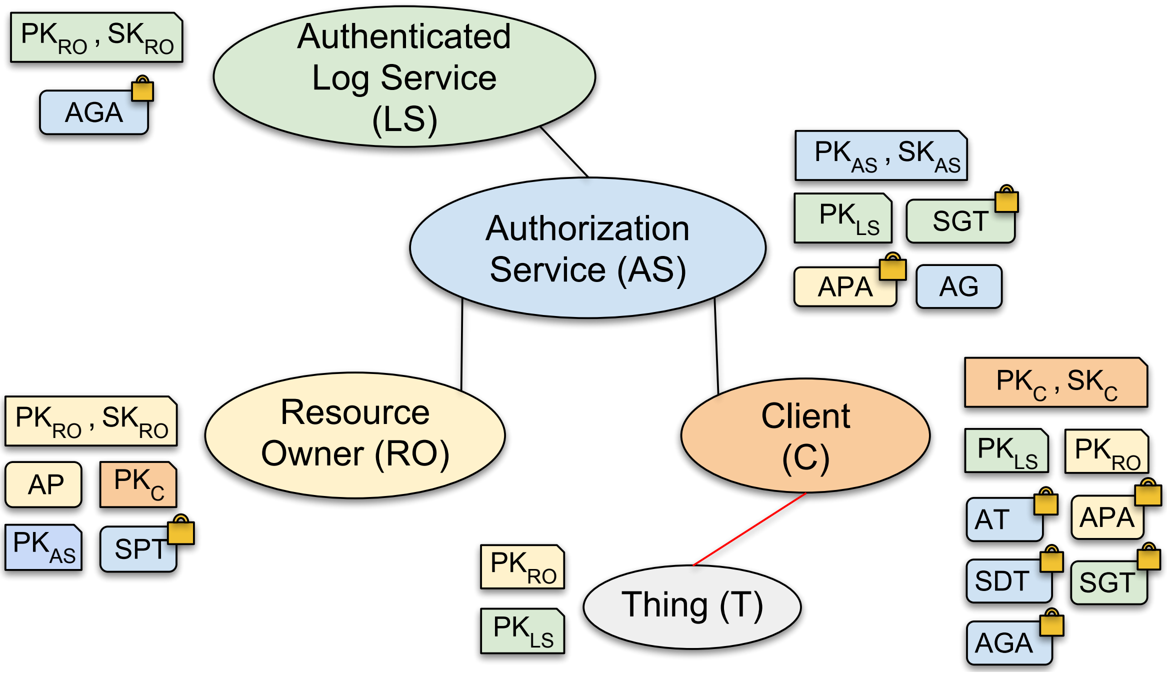


Figure 3.3: IoT transparency architecture.

The security of the proposed approach is based on three main design choices. *(a)* The system requires each party to operate requests by using *request attestations*, which are cryptographic data structures that allow to prove their (mis)behavior to third parties. *(b)* The authorization service, that is the root of trust for all issued authorizations, is forced to log all its operations to the external authenticated log service, thus allowing other parties and potentially public third parties to monitor its behavior. *(c)* By using authenticated data structures, the only security assumption on the authenticated log service is that it is always available, i.e., that it answers to all operations requests. If the service tries to misbehave by returning incorrect answers, the other parties are able to detect them.

For ease of presentation, we distinguish attestations in two categories: *request attestations* and *signed response timestamps*[1]. The *request attestations* authenticate the data sent by a party in a request. In this category we identify *authorization policy attestations* (*APA*) and *authorization grant attestations* (*AGA*). For ease of presentation, we pospone the details of their design to Section 3.2.3, that discusses the protocols in which they are used. The *Signed Response Timestamps* assess the acceptance of a request that applies modifications on the state of the service (e.g., insert or update data). In this category we identify the *signed policy timestamp* (*SPT*), *signed grant timestamp* (*SGT*) and *signed denial timestamp* (*SDT*). All signed response timestamps are computed with regard to a request as the following function $\mathcal{SRT}(\cdot)$:

$$\mathcal{SRT}(req) := \langle \mathcal{H}(req), nb, \sigma \rangle , \tag{3.1}$$

[1]We adopt the notation *signed timestamp* from the *General Transparency* project (88).

Server	**Client**	**Operation interface**	**Description**
AS	RO	$\{SPT, \text{'}\mathit{reject}\text{'}\} \leftarrow \mathit{UpdatePolicy}\,(APA)$	Update of an authorization policy
	C	$\{\langle AGA, SGT\rangle\,, SDT\} \leftarrow \mathit{Authorization}\,(AP)$	Request an authorization grant
		$\{AT, \text{'}\mathit{reject}\text{'}\} \leftarrow \mathit{Token}\,(AG)$	Request an access token
		$APA \leftarrow \mathit{AccuseDenial}\,(SDT, SPT)$	Accuse of illegitimate authorization denial
T	C	$\mathit{result} \leftarrow \mathit{Request}\,(\mathit{data}, AT, SGT)$	Request access to the thing
LS	AS	$SGT \leftarrow \mathit{InsertGrant}\,(AGA)$	Insert an authorization grant
	RO	$\pi \leftarrow \mathit{QueryGrant}\,(AGA)$	Request proof for an authorization grant
RO	C	$\{SPT, \emptyset\} \leftarrow \mathit{VerifyDenial}\,(SDT)$	Request verification of authorization denial

Table 3.2: Operations framework of the IoT transparency architecture.

where $\mathcal{H}(\cdot)$ is a deterministic collision-resistant hash function (e.g., SHA256), nb is the *"not before"* timestamp that defines at which time-instant the modifications required by the request are to be considered applied at the service, and σ is the cryptographic signature (e.g., ECDSA) of the service applied on the tuple $\langle \mathcal{H}(\mathit{request}), nb\rangle$.

In Table 3.2 we show the operations framework of the architecture. The operations *UpdatePolicy*, *Authorization*, *Token* and *Request* are variants of those proposed in the reference architecture (see Section 3.1.2, Table 3.1). *AccuseDenial* and *VerifyDenial* are additional operations offered by the authorization service and by the resource owner that allow parties to operate audit operations. Finally, operations *InsertGrant*, *QueryGrant* are services offered by the authenticated log service to store and query information by other parties. To focus on a solution that is ready to be applied in real-world applications, we consider operations that can be implemented by using the Trillian software (88) as a back-end, that is a production-ready open-source project maintained by Google as a reference implementation for Certificate Transparency log services.

In the following, we describe the details of each operation within the main protocols of the proposed system, that are *accountable authorization protocol*, *verification of denied authorizations* and *verification of issued authorizations*.

3.2.3 Accountable authorization protocol

The *accountable authorization protocol* allows clients to obtain the due authorization material to access things from the authorization service. It extends the original protocol for delegated authorizations described in Section 3.1.2 to enable accountability. In the following

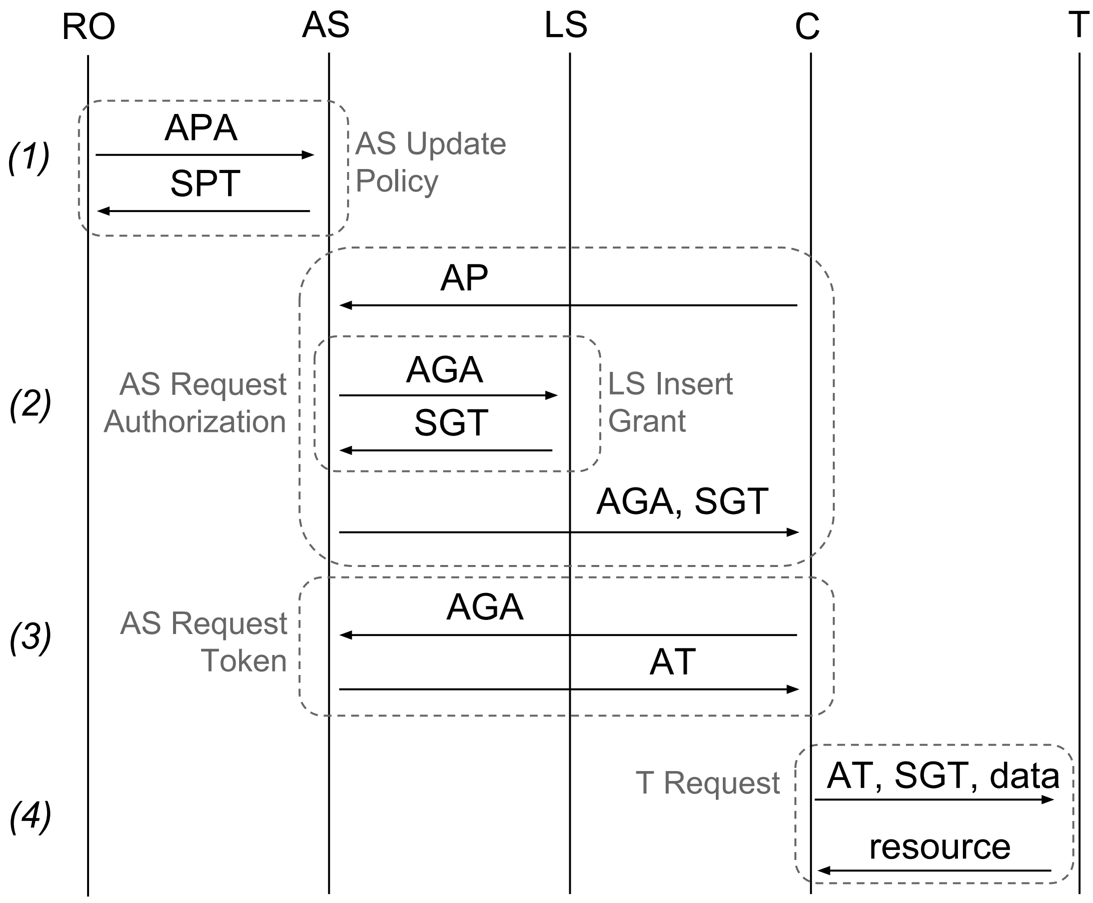


Figure 3.4: Accountable authorization protocol.

we describe the four phases of the protocol by referring to Figure 3.4.

(1) The resource owner delegates the authorization service by using the *UpdatePolicy* routine. He builds an authorization policy attestation as the tuple $APA = \langle id, AP, \sigma_{APA} \rangle$, where id denotes a unique identifier of the attestation, AP is the authorization policy (as defined in the reference protocol) and σ_{APA} is the signature of the tuple $\langle id, AP \rangle$ generated by using the secret key of the resource owner. The service answers by returning the signed policy timestamp SPT as a proof of acceptance of the policy, computed as $SPT = \langle \mathcal{H}\left(\mathrm{APA}\right), nb, \sigma_{SPT} \rangle$.

(2) The client requests an authorization grant to the authorization service to access a thing by using the *Authorization* routine. It builds a proper authorization policy, as described in Section 3.1, that is validated by the authorization service with regard to the known authorization policies. If legitimate, the authorization service builds the authorization grant attestation $AGA = \langle \mathcal{H}\left(AG\right), AP, tn, V, \sigma_{AGA} \rangle$, and inserts it in the authenticated log service by using the *InsertGrant* routine. The authenticated log service returns a signed grant timestamp $SGT = \langle \mathcal{H}\left(\mathrm{AGA}\right), nb, \sigma_{SPT} \rangle$ as a confirmation of acceptance of the grant. Note that hash function $\mathcal{H}\left(\cdot\right)$ is used to hide the content of the authorization grant from the authorization service, that is a knowledge that must be maintained secret between the service and the client. Moreover, note that the authenticated log service does not guarantee

to insert the grant immediately, but guarantees to insert it within the *nb* time of the SGT data. This is a strategy of many architectures based on authenticated data structures to guarantee scalability (59; 82; 87).

(3) The client requests an access token to the authorization service through the *Token* routine by including the authorization grant attestation AGA, that is used by the service to compute the access token $AT = \langle \mathcal{H}\left(\text{AGA}\right), AP, \sigma_{AT}\rangle$, where AP is the authorization policy included in AGA and σ_{AT} is the signature of $\langle \mathcal{H}\left(\text{AGA}\right), AP\rangle$ computed by using the secret key of the authorization service. Note that the hash function $\mathcal{H}\left(\cdot\right)$ computed over AGA must be the same used to compute SGT, to allow things to validate that AT and SGT used in the next phase refer to the same AGA.

(4) The client accesses the thing by sending a valid access token AT, the signed grant timestamp SGT and the due payload *data* of the request. The thing accepts the request if and only if the values corresponding to $\mathcal{H}\left(\text{AGA}\right)$ included in AT and SGT are the same. Then, the thing validates AT by verifying the authorization policy, as in the reference protocol (see Section 3.1.2) and the signature included in the access token by using the known public key of the authorization service. Finally, the thing also validates the signature of the signed grant timestamp SGT by using the known public key of the authenticated log.

3.2.4 Verification protocols

We propose two verification protocols: to verify that no authorization has been granted to unauthorized clients, and to verify that all authorizations have been granted to authorized clients.

The first protocol allows a resource owner to verify if the authorization service has issued authorizations to unauthorized clients. To this aim, a resource owner can act as a monitor of the authenticated log service, downloading the whole authenticated data structure periodically. This procedure is similar to that of monitors for the Certificate Transparency system and could be easily deployed by using existing software implementations. An alternative would be to use the more efficient *QueryGrant* routine offered by the authenticated log service. In this case, the authenticated log service must maintain authenticated data structures that are efficient with regard to the considered queries (62). As an example, a resource owner might want to monitor all the authorizations released within a certain time frame for a specific thing or for a specific client.

The second protocol allows clients to verify if the authorization service has illegitimately refused to issue authorizations despite existing authorization policies released by the resource owner. This verification procedure includes three phases, that we describe by referring to Figure 3.5.

(1) We assume that the authorization service denies access to a thing by a client that invoked the *Authorization* routine, and thus it answers by returning a signed denial timestamp $SDT = \langle \mathcal{H}\left(AP\right), nb, \sigma_{SDT}\rangle$. Please note that the *nb* value represents the time instant at which the authorization service refuses the authorization request.

(2) To verify that the authorization denial was legitimate, the clients uses the *VerifyDenial* routine offered by the resource owner by sending AP and SDT previously sent and obtained

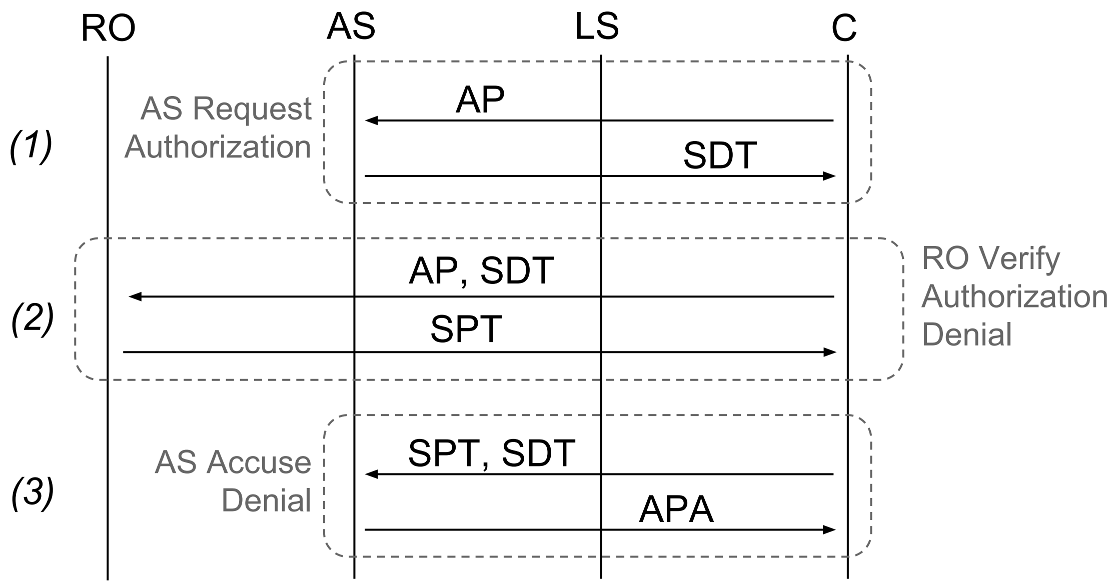


Figure 3.5: Verification of illegitimate authorization denial.

by the previous invokation of the *Authorization* routine. The resource owner verifies if the SDT is correctly bound to the AP by re-computing the hash value $\mathcal{H}(AP)$ on its side, and verifies if the AP complies to the authorization policies previously established by using the *UpdatePolicy* routine (see Section 3.2.3). If an authorization policy exists that can assess that the client had legitimate access to the thing, then the resource owner returns the signed policy timestamp SPT that the authorization service returned as an answer to the *UpdatePolicy* routine. Note that, for SPT to be considered valid, the release timestamp nb included within SPT must be greater than the one included in SDT.

(3) The client uses the SPT received by the resource owner, together with the SDT received in phase *(1)*, to send a request to the *AccuseDenial* primitive offered by the authorization service. The protocol allows the authorization service to defend itself against the accusation by returning an authorization policy attestation APA that is more recent that the SPT and that can assess that the denial was legitimate. Note that this is only possible if the resource owner actually updated its policies and misbehaved in phase *(2)* by not operating correctly. If such an APA does not exist, then the authorization service behaved incorrectly.

3.2.5 Comparison with related approaches

We present a comparison of the proposed system with regard to alternative approaches proposed in the literature by referring to Table 3.3. We consider four proposals that have strong security guarantees, based on cryptographic primitives, and consider similar threat models where the authorization service is considered malicious: *transparency* refers to the system proposed in this chapter; *Bitcoin log* refers to the Catena protocol illustrated in (56), proposed in the two architectural variants based on a device that runs a blockchain *full node* or the Simplified Payment Verification protocol (SPV); the *Ethereum smart-contract* approach refers to the systems proposed in (57; 89).

We distinguish three types of characteristics: *security* identifies the peculiar traits of each protocol in terms of security *assumptions* and *guarantees*; *thing requirements* identifies the characteristics of the IoT environments, distinguished in hardware capabilities of the IoT devices (e.g., capability of *execution* of certain algorithms, and *storage* size) and characteristics of the *architecture* where things are operated (e.g., availability of an Internet connection); *protocols timings* identify the performance of the main protocols operations, that are *grant authorization* by an authorization service (the time required by a party to release an authorization that can be considered valid by a thing), *audit delay* (the maximum delay after which a party can operate a verification at the authorization service) and *request verification* (the time required by a thing to validate a request by a client).

The *security* characteristics allow to understand if a solution is able to satisfy the requirements of the considered scenario. As an example, we highlight that approaches based on distributed ledgers do not guarantee only protection against a dishonest authorization service but, due to the native characteristics of the distributed ledgers protocols, also availability of the service. Moreover, protocols based on smart-contracts offers even stronger securities by preventing an authorization service from behaving maliciously (we identify this trait by using the *enforcement* guarantee instead of *accountability* guarantee).

The *thing requirements* allow to identify if each solution can be deployed in a certain scenario. As an example, we note that both blockchain solutions might require a lot of storage at the thing side to maintain the blockchain. This disadvantage could be reduced by using special-purpose blockchain networks of smaller sizes separated from the public ones, or by using the SPV protocol, that however introduce stronger security assumptions (that is, trust in intermediate proxy nodes). The proposed approach is very lightweight as it requires only storage of a few public keys and it does not require any Internet connectivity for the things. As disadvantages, the proposed system requires a trusted setup of these keys (as we discussed in Section 3.2.2), and the resource owner to be available to verify illegitimate authorization denials.

Finally, the *protocol timings* highlights that the proposed approach might require longer wait times to be able to operate an audit operation after an authorization grant (the *audit delay time* field), that is the Maximum Merge Delay (MMD) value used by the authenticated log service to merge the released authorization grants within its authenticated data structures. However, please note that while the wait time in our approach is deterministic and refers to the release of the authorization grant, in bitcoin-based logs it depends on when miners are able to insert the due transactions in the blockchain. Finally, note that releasing a verified authorization in the smart-contract approaches requires significant additional times because they require that the transactions are inserted in the blockchain before releasing the authorization.

3.3 Analyses and evaluations

We analyze the benefits of the proposed architecture by comparing it against related approaches (Section 3.3.1). Then, we measure its performance and we analyze its applicability

Approach	**Advantages**	**Disadvantages**
Trusted platforms	Lightweight	Security and trust issues
Blockchain-based log	Cryptographic proofs High availability Prevent misuse	High latencies for write operations Large storage requirements Internet-connected devices
Intrusion detection system	Wider features No modifications to devices	No cryptographic proofs
Proposal	Cryptographic proofs Support offline devices Confidentiality	Security trade-offs Small overhead on access requests

Table 3.3: Advantages and disadvantages of related families of approaches.

to IIoT scenarios (Section 3.3.2).

3.3.1 Comparison

For comparison purposes, we consider the *trusted platforms* described in Section 3.1, where both domain and inter-domain platforms are trusted, and two families of approches namely a *blockchain-based* approach where released authorizations are stored in a public distributed ledger maintained by all the platforms and devices through a state-replication consensus protocol; an *intrusion detection system* where authorization misuses are detected by inspecting network traffic. A summary of the comparative analysis is reported in Table 3.3, and discussed below.

The first advantage of the trusted platforms is its lightweight performance, because it only requires domain platforms to send information to the inter-domain platform, which is trusted in storing them and in not violating confidentiality of authorization information. However, this approach has severe security and lack of trust issues, and does not match the considered threat model where domain and inter-domain platforms are untrusted.

Storing all authorization information on a blockchain-based system would inherit the advantages of the blockchain including high availability of the information and the capability of enforcing verification of authorizations at insertion time by using smart contracts (11). On the other hand, this system inherits the disadvantages of the blockchain that can introduce high delays at insertion time and costs (e.g., (90)). Moreover, all devices that must verify correctness of the requests would require large storage to maintain the blockchain and Internet-connectivity to guarantee updates.

An intrusion detection system is a quite different approach, but it might guarantee similar attributes. Its main advantages include the ability to operate without introducing modifications to devices and to detect a much wider range of attacks. However, it is unable to guarantee a cryptographic proof of misbehavior.

Our proposal does not require to trust domain and inter-domain platforms thanks to cryptographic assertions exchanged by the parties. Misbehaviors by the inter-domain platform can be detected thanks to authenticated data structures. The inter-domain platform cannot access confidential information because domain platforms encrypt and obfuscate data before storing them. The cryptographic schemes on which all protocols are based, which are message authentication codes, hash functions and digital signatures, can be supported by many classes of devices. Even more important, IIoT devices are allowed to operate offline because they do not require frequent updates nor to maintain a large amount of data. As a result, the proposed approach represents the best trade-off in terms of performance, security and flexibility for securely attributing authorizations misuse in industrial scenarios.

3.3.2 Experimental evaluation

The experimental evaluation of the proposed protocol considers the performance of the authenticated log service that represents the most innovative and critical component of the architecture. The software is based on Google Trillian (88) that implements an authenticated data structure based on Merkle Hash Trees (MHT). As described in Section 3.2, the service receives new Obfuscated Authorization Grants (OAG) and returns Signed Grant Timestamps (SGT). For performance reasons, a new OAG is not inserted synchronously within the MHT, but the service inserts it periodically through batch merge operations. This service guarantees the clients that a received OAG will be merged within a specific time window through the *not before* (nb) value within SGT. The delay required to merge an operation is called *maximum merge delay* (*MMD*).

Authorization mechanisms have different requirements in terms of service operation latencies and maximum merge delays. As an example, a scenario may require a log service to grant authorizations within few milliseconds and involve audit operations every few days or weeks. Other scenarios may have opposite requirements, such as allowing long authorization delays (e.g., programmable interventions) but short audit times (e.g., the operation is critical and an invalid authorization must not be allowed or be detected within very short time intervals).

The trade-off between performance and costs should be clear. As authorizations can be audited only after merging of authorization grants, higher *MMD* values denote lower costs and lower quality of the service. On the other hand, lower *MMD* values are more expensive because MHT update operations are characterized by low degree of parallelism. As a result, merge delays must be sized appropriately depending on the throughput of insertion operations. In our evaluation, we consider different amounts of operations and we measure the service latency for each of them. To the aim of analyzing the behavior of the proposed system and of determining the best candidate values for different maximum merge delays, we evaluate the performance for increasing amounts of concurrent authorization operations.

We consider a real scenario represented by authorization systems for air-gapped environments, where operators and machines (e.g., automated trolleys) must request access to resources that cannot always verify authentication and authorization material online. In these environments, a moving device must obtain authorization grants from an authorization service, move to the air-gapped location, and present these grants to offline devices, that must decide to accept or reject the material. We estimate the number of automated devices that move in industrial plants to be highly variable, depending on the size of the environment and the types of productions that can range from few units to tens of requests. In the emulated environment, that use fine-grained and short-lived authorizations, the number of requested authorizations can be in the order of hundreds per second. We evaluate the performance of the system by measuring:

- the time required to confirm the acceptance of a new authorization from the domain platform. All operations timings are measured through the Trillian testing utility;
- the total *authorization log time*, that is, the time required by the service to merge new authorizations in the MHT and make them available for auditing. The log time is evaluated by measuring merge delays through querying the hash of the log transaction and extracting the *queue timestamps* and the *integration timestamp* that are two metadata included in the MHT. Then, we compute the log time as the difference between the two values;

For both measures, we perform multiple evaluations that are based on the following configurations:

- we consider increasing operation workloads to evaluate the scalability of the system, including 10, 100 and 200 insert operations per second. The requests are sent until 10000 logs are integrated into the log tree. In order to avoid measurement biases we repeat the evaluation 50 times and consider the average value.
- we evaluate the influence of *MMD* on the system performance by considering five *MMD* configurations: 2, 3, 4, 5 and 6 seconds;
- we measure the integration time by combining the operation latencies and the maximum merge delay.

The server machine executing the authenticated log service is configured with five virtual CPUs, 50GB of RAM and an SSD hard drive. Figure 3.6 shows the response times of the service for new authorizations. The y-axis reports the average response times in ms. The x-axis denotes the results for the three workloads and increasing *MMD* values. The average response times typically range from $22ms$ to $80ms$ as a function of increasing workloads. We can observe that low values of *MMD*, in the order of 2 seconds or less, greatly affect the performance of the system in the case of high workloads. Indeed, the system tends to saturate for configurations with $MMD = 2s$ and more than 100 operations per second. This result evidences that for a similar system, adequate performance is achieved when the service is configured with values of *MMD* greater the $3s$.

Figure 3.7 reports the *authorization log time*, where the x-axis and the y-axis denote the three workloads for increasing *MMD* values, and the log times expressed in seconds, respectively. This figure confirms that the total log time of the proposed mechanism is unaffected by the required number of operations per second, as the log time is almost equal to the choice of the *MMD* value. In practice, some milliseconds are added to the basic *MMD* value that is in the order of seconds. These results confirm the exceptions about

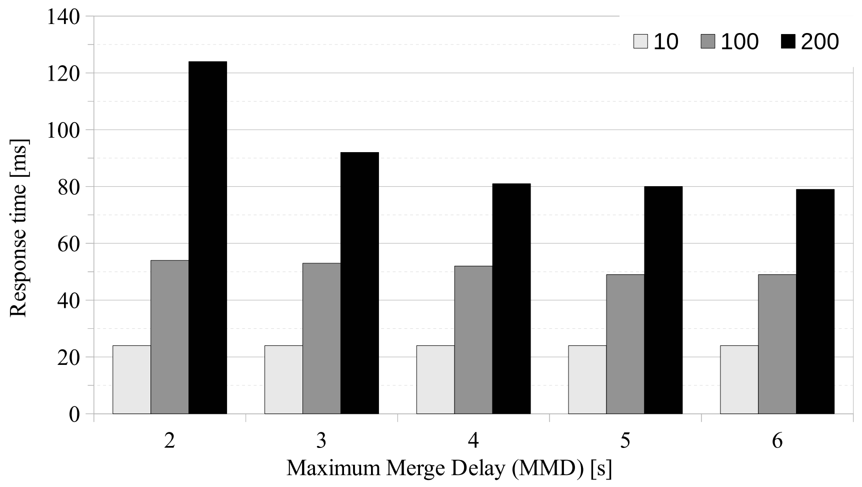


Figure 3.6: Latency of the authorization insertion operations.

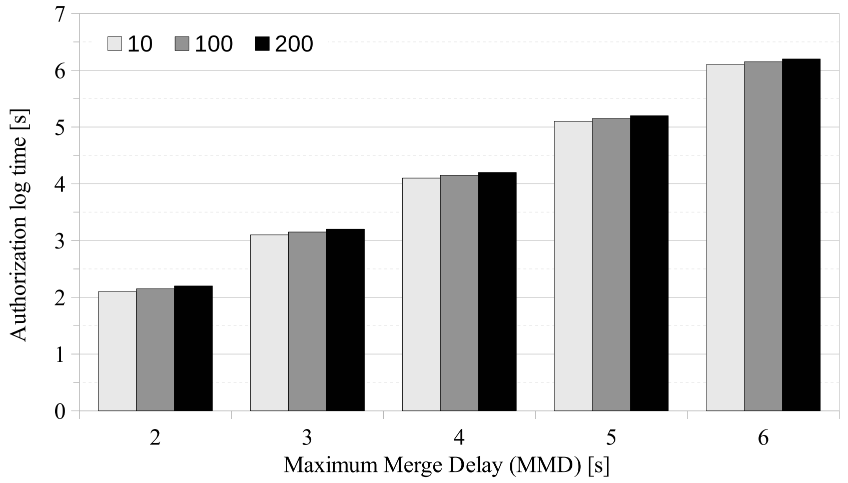


Figure 3.7: Total log time of new authorizations.

the proposed system, when *MMD* is a reliable setting to control authorizations even in an industrial setting where smart devices have low computational capacities.

Based on previous results and four main applications, we can claim that the proposed system is applicable to any realistic scenario. The performance of the system is characterized by two features: the access time that is, the amount of time required to complete the procedure related to granting access; the log time (represented as MMD) that is the time taken to provide reason to the end-user in the case of an adverse decision to an access.

For example, in the case of a Smart Car belonging to the factory fleet a typical employee can directly approach a car to borrow it for commuting. In a similar case, the time required to complete the procedure for granting access should be small. On the other hand, if the access is denied, the employee may move to the next available car (or even take his owned vehicle) and check the logs for denial later. In this case, the log time can be large.

IIoT Application	**Access Time**	**Log Time**
Smart Car	3.5-4 seconds	24 hrs
Smart Locker	3.5-4 seconds	3.5-4 seconds
Smart Lab	few weeks	3.5-4 seconds
Smart Warehouse	few days	24 hrs

Table 3.4: IIoT application latency assuming 3-seconds MMD.

The case for Smart Lockers is similar to the case for Smart Cars. The time required for access setup is very short as the employee is already at the locker requesting for access. If the access is denied, the employee needs to know the reason immediately as he may have used the locker to keep his stuff.

Now consider the case of a Smart Lab, where the (team of) employees typically begin the booking process in advance, so there is sufficient time for the completion of access related procedure. If the access is denied, the employees would like to know the reason immediately so that the problem can be solved and they can avoid delay or rescheduling issues.

For the Smart Warehouse application, let us consider the case where an online retailer is going to deliver a package at the warehouse, but no warehousemen is available for delivery processing/acknowledgment. Thus, an employee creates the access rule for the delivery person. The access time to setup the delegation is high. When the delivery person arrives, the access is denied. There is no remedy but to leave the package outside unattended or to take back. Later on, the delivery person fills a complaint report about the access denial, thus allowing some time for logs.

The Table 3.4 summarizes the four smart applications with an estimated access time and log time characteristics.

For an asymptotic analysis, we have to observe that a tree can manage a maximum of 2^{63} transactions. If n denotes the number of delegating domain platforms, k the number of devices, and x stands for the number of transactions generated by each device per day, we can easily evaluate that a saturation may occur after some thousands of years. Even in a large industry with hundreds of delegators, thousands of devices and hundreds of thousands of transactions per day, we can reach some dozen of billions of log records per day that remain largely below 2^{63}. This analysis evidences meaning that our solution never saturates in practice.

3.4 Conclusions

We proposed a system that allows to audit authorization procedures operated in industrial IoT environments, characterized by highly secure air-gapped systems and devices with low resources placed in constrained environments. The proposed design is compliant with standard authorization and network communication protocols and can leverage existing software services and libraries for a reliable deployment. Its security is based on established cryptographic protocols, such as standard digital signature schemes and hash functions, and allows each party involved in the system to prove misbehaviors publicly, incentivizing each industrial party involved in a collaboration to adopt the best security practices to avoid misbehaviors due to internal or external adversaries. The proposed experimental evaluation operated by using a prototype implementation based on the established Google Trillian project, shows the feasibility of the system even in presence of intensive operation workloads.

Part II

Data Acquisition, Storage, and Visualization

Chapter 4

Introduction

In the first part, we discussed protecting the IoT devices against unauthorized access using Blockchain-based and authenticated data structure-based approaches. The next crucial aspect is the data generated by IoT devices. Data forms the backbone of the value-added services offered to the end-user. Moreover, trust in collecting, storing, and representing data is fundamental in such services. This trust ensures the transparency of the system, the residents' participation in the process of governance, and the economy's growth. In this chapter, we focus on securing the data against tampering, right from the time it is generated at an IoT device to data visualization. The security mechanism should also ensure transparency and verifiability for the user to trust the system. We begin by describing the challenges faced by the research community in protecting data integrity and transparency while allowing auditing. Then, we present the literature survey of the existing approaches addressing the issues and their shortcomings. Next, we present a primer for the technologies used in the proposed solutions. Finally, we conclude the chapter by briefly describing the proposed approaches to address some of the challenges.

4.1 Problem Statement

Data is the most critical element in any IoT deployment. With the recent proliferation of smart devices and the Internet of Things, vast amounts of data are generated at any time (91). With the amount of data available, data analytics can be explored to get insights into various problems present in the world. The amount of data available to analytics programs has vastly increased with the rise of smartphone technology, but the Internet of Things (IoT) promises to take this data deluge to new heights. Data analytics has been implemented in the business world for some time, enriching all industries, from manufacturing to marketing. With home appliances, health monitors, transport infrastructure, and many other everyday items soon to be equipped with online connectivity, the insights available to businesses will be staggering.

With the rapid deployment of IoT and the volume of data generated, it is quickly becoming impossible to keep increasingly high volumes of data locally. The resource-constrained nature of the IoT devices also limits the processing of the data at the source. Thus, cloud

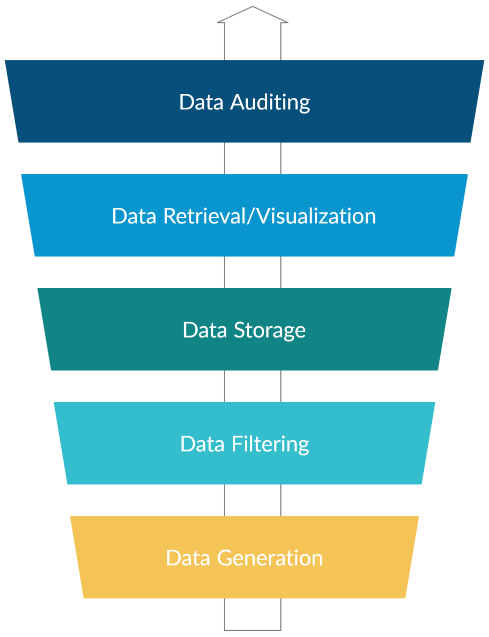


Figure 4.1: Data pipeline.

storage is gaining popularity not only among enterprises but also among small organizations and individuals. The data generated by the devices are sent to the cloud storage and later used to infer actionable information. However, remote cloud storage presents its challenges. Trust is critical for any information system generating actionable information. With multiple and varying devices and entities involved, the system needs to be trustless; the participants involved do not need to know or trust each other or a third party for the system to function. There is no single entity with authority over the system in a trustless environment, and the consensus is achieved without participants having to know or trust anything but the system itself.

For any IoT solution deployed for data collection, the first step is to set up an IoT architecture powered by low-cost micro-controller boards with sensors and actuators installed onto public structures and thus, scattered all over the urban area. Such infrastructure facilitates data and information collection to build services for users, who may participate in the network via smartphones, and other mobile devices, interacting with the (e.g., physical) objects, and may even generate data. The IoT framework can be leveraged for other services, e.g., mobility management, district area monitoring and safety, urban budget tracking, reporting of acts that may jeopardize public facilities maintenance, and public safety. In this context, with many different stakeholders and interests involved, it is essential to guarantee that (sensor) data, once saved for long-term storage, cannot be either tampered with or at risk of being withheld by any single-party.

Any remote cloud storage faces challenges like misbehaving participants to protect the information stored on the cloud. It could be server misbehavior like modifying the stored content, denying access to the content, and claiming data unavailability. The misbehavior causes the client to mistrust the server. Similarly, a client can misbehave by falsely claiming violation of user agreement. To ensure fair behavior of involved entities, provable cryptography primitives, and public auditability mechanisms need to be developed.

Further, there are several trustiness issues in any IoT framework, which may be better explored by identifying several data processing and consumption stages, as depicted in Figure 4.1. During the data generation, contributors cannot necessarily be trusted. Data can either be unreliable or can voluntarily be forged. Thus, even though the system is decentralized, it is not trustless. At the data filtering level (Figure 4.1), the initiative-hosting entity (e.g., university) holds total power in deciding if a data sample would be accepted or not. Thus, there exists a single point of trust. The same considerations apply at the data storage level where an IoT device collects the data, and again the hosting entity is in charge of secure storage. The same thing is valid for data retrieval and visualization. Finally, there is no way for consumers to audit data in the centralized approach and, thus, to be sure that everybody is behaving correctly.

4.2 Related work

Initially, we survey the literature for provable data immutability solutions. Blockchains and DLTs have been proposed several times as tools to tackle data immutability and keep it tamper-proof. (92) tried to address the problem of data provenance in cloud services using a secure provenance scheme called SECProv. The authors devised a secure provenance chain, a chain of provenance blocks linked together with the hash of the preceding block. The design, though similar to blockchain, eliminated the need for mining. The system introduced a provenance gateway which was responsible for handling security-related operations. (93) used a strategy similar to Proof-of-Possession wherein the file is represented as a collection of data identifiers organized in a Merkle tree. To prove ownership of the client, the cloud provider presents hashes of the Merkle tree as a challenge, and if the root hash matches, the client is the valid owner of the file. (94) approached the trust issues in cloud service using certification. The solution proposed a certification model consisting of test cases to be validated and probes responsible for collecting evidence. Test managers manage the certification models. Thus, the test manager and the probes responsible for evidence collection must be trusted. Also, third-party validation is not possible in this case, as the test manager handles the verification.

Next, we survey the literature for securing the entire data pipeline. In (95), blockchain technology has been leveraged as a shared public database in an IoT Smart City context, and SCRUM methodology, which entails clients playing a central role in the system, is used for verification and correctness of the developed system. More in general, blockchain-related technologies have been widely applied in the context of Smart City-related services (96) and other adjacent domains, such as Smart Tourism (97), Smart Grids(98), Smart Environments, and Smart Mobility(99) the latter with specific reference to renewable energy sources. A blockchain-based distributed framework for the automotive industry, still in the context of Smart Cities, has been proposed in(100), and a blockchain design specifically geared towards smart vehicles has been proposed in (101) to ensure privacy, integrity, resilience, and non-repudiation of data. The use of blockchain has also been investigated to incentivize citizens in sharing data (102). A broader look at enhancing IoT security through blockchain-based mechanisms has been outlined in (103). In (104), a blockchain has been leveraged to counter security vulnerabilities, and the blockchain itself has been proposed as the primary database. In (105), a light client for Ethereum has been proposed to preserve data integrity when running on IoT nodes. One of our system's goals, i.e., keeping data tamper-proof, expands the scope of usage for otherwise not fully trusted sources of data to various applications, creating a healthy environment that engages both producers and consumers. A similar idea has been extensively discussed in (106). It proposes a blockchain-based marketplace for the data generated by IoT devices.

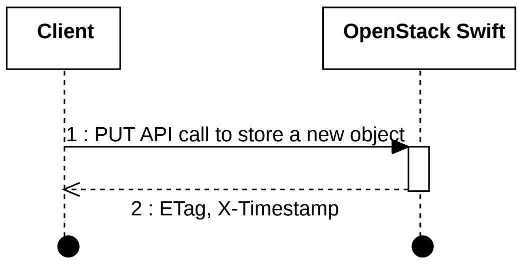
4.3 Preliminaries

This section provides some background about the leading technologies that have been exploited during the project, namely Openstack Swift, Arancino hardware platform, Tendermint software platform, and BigchainDB data store.

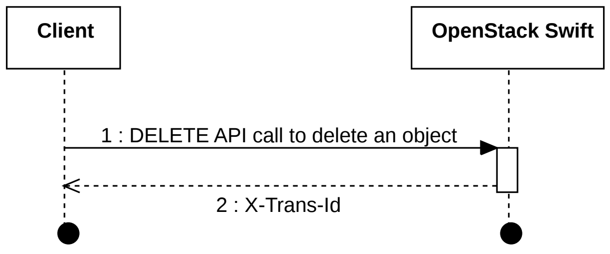
4.3.1 Openstack Swift

OpenStack Swift is a highly available, distributed, eventually consistent object storage. It can store all unstructured data such as photos, videos, container/VM images, log files, and archives. Swift comes with Proxy and Storage nodes. Proxy exposes REST service to read, write, delete objects, and their metadata into the storage. Swift organizes the collection of objects in containers. A container can be identified as a folder that holds a collection of objects. Proxy service enables users to store, retrieve, and delete objects (with their associated metadata) in the containers via a RESTful HTTP API. Swift can be accessed with HTTP requests directly to the API or by using one of the many Swift client libraries such as Java, Python, Ruby, or JavaScript.

OpenStack Swift allows users to store unstructured data objects with a canonical name containing three parts: account, container, and object. Using one or more of these parts allows the system to form a unique storage location for data. The account storage location is a uniquely named storage area that will contain the metadata (descriptive information) about the account itself and the list of containers in the account. Note that in Swift, an account is not a user identity. The container storage location is the user-defined storage area within an account where metadata about the container, and the list of objects in the container will be stored. Figure 5.1a depicts the sequence of interactions between a client and the Swift service to store a new object. OpenStack Swift allows a client to store an object using API calls. In order to deposit a file in the storage, the client needs to invoke PUT API. Once the file is stored successfully, Swift returns an MD5 hash of the content of the file and the timestamp of initial object creation in addition to other details. Similarly, in updating the file, the client needs to invoke PUT API with the modified file. The Swift service replaces the file and returns the MD5 hash of the new file and the timestamp of initial object creation, and other details. Finally, in case of an object's deletion, the client invokes



(a)



(b)

Figure 4.2: (a) Object insertion in Swift using API call, and (b) Object deletion in Swift using API call.

a DELETE API call to request the Swift service to delete the object from the storage. The Swift service queues the request and returns a transaction identifier that needs to be saved for reference in any issues (Figure 5.1b).

4.3.2 Arancino

Arancino[1] is the trademark for a family of embedded boards ("Arancinos") produced by an academic spin-off company, SmartMe.io. The prominent feature of these boards is the joint on-board availability of one (or more) microprocessors (MPUs) alongside one (or more) microcontroller (MCUs) in order to assign workloads and peripherals (e.g., sensors) according to the unique features of every module, specialized for specific duties, while making room for the two subsystems to work closely enough to cooperate. In particular, on the MCU side, Arancinos host, in their base configuration, an Atmel SAMD 32-bit part, which acts as a bridge between sensors and the MPU, where most logic will run. In terms of MPU, a socket can host, e.g., a Raspberry Pi 3 Compute Module. When populated with modules this powerful, the socketed design makes Arancinos equivalent to (and exceeding) in functionality, a Single-Board Computer, including the capability to run a minimal Linux-based environment.

Most notably, concerning this work, focusing on data generation here, Arancinos are equipped with one (or more) cryptographic co-processors (*cryptochips*), which act as tamper-proof hardware-based key storage and support crypto ops on-chip. Arancino also features

[1]http://smartme.io/it/projects/arancino-cc-2/

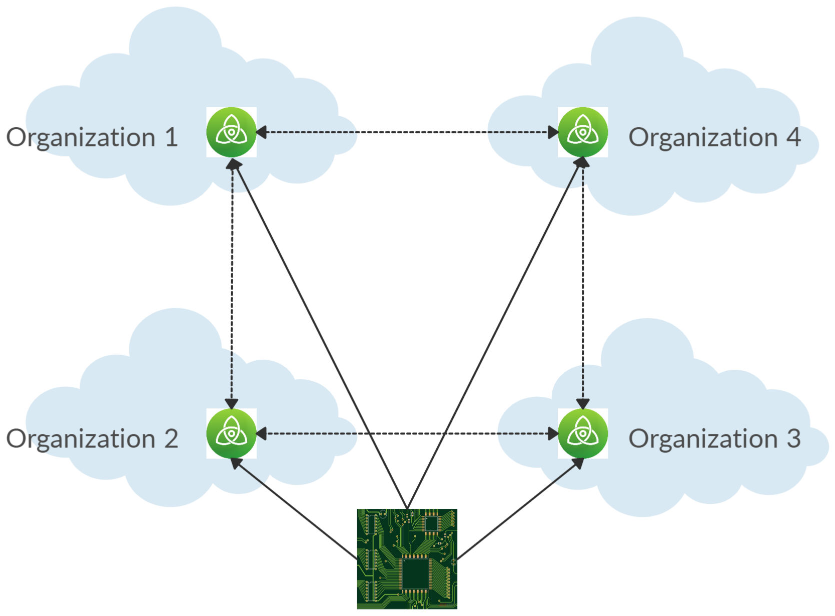


Figure 4.3: Trustless consortium-run/consensus-based application.

the usual (sizable) array of connectors (UART, SPI, I2C, GPIO, and USB) and connectivity options (GSM, WiFi, BLE, and LoRa). As energy consumption may be a stringent requirement even for this kind of board, e.g., battery-powered and equipped with pocket-size solar panels, duty-cycling techniques (107) are being evaluated for future iterations.

4.3.3 Tendermint

Tendermint (108) is a PBFT consensus engine that enables interested parties in building consortium blockchains, on top of which any application can be executed in a decentralized and fully verifiable way. This is valid even if the single parties involved in the consortium do not fully trust each other and are not fully trusted by any external entities. This is a perfect match for a Smart City. In fact, in a city, it is expected that several stakeholders that do not necessarily fully trust each other may have an interest in building a public service, representing an asset for everybody, i.e., everyday goods. At the data filtering level, we had to decentralize the decision about which data sample should be accepted in the system or not, ensuring that it is taken fairly. This means that there is no single point of failure, but it also means that there is no possibility of censorship by any central authority. In our case, we experimented with a consortium formed by four nodes hosted by different city institutions, and we assigned such a consortium the duty of checking for any data sample if it was coming from a known and authorized combination of IoT node and sensor, or not. The decision is taken accordingly, thanks to the PBFT consensus protocol, so that if a single node misbehaves, it is kicked out of the consortium, and the application stalls. In this way, nobody is interested in cheating, i.e., excluding data samples that would be in good standing and rightfully expected to be inserted in the system. The only possibility to act against

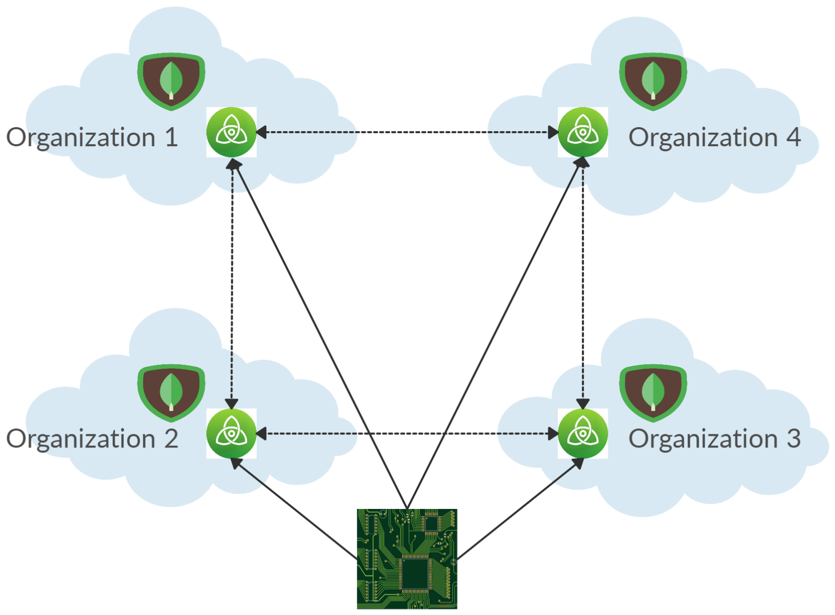


Figure 4.4: Trustless consortium-run/consensus-based data storage layer.

the rules mentioned above is that most nodes collude and start misbehaving, but this is also highly improbable in this setting.Summarizing, as exemplified in Fig. 4.3, application logic is run by several stakeholders forming a consortium, and the decentralized application, providing no single point of failure or trust, is assigned the following duties:

- Receiving data samples
- Accepting or discarding samples
- Forwarding samples to the data storage layer
- Adding or removing trusted nodes

4.3.4 BigchainDB

BigchainDB (109) is an open-source database software which, along with traditional database features like high transaction rate, indexing and querying capabilities, and low latency, also has DLT properties. It combines the two paradigms of industry-standard distributed databases and blockchains, providing benefits of both. It uses the widely accepted and highly developed MongoDB as the underlying database, and it has all the flexibility in querying, which MongoDB provides. As exemplified in Fig. 4.4, the Tendermint consensus engine is used again here to introduce the blockchain properties. The data stored in BigchainDB is immutable and cannot be tampered with once stored inside the database. This means that, once the filtering level has accepted a data sample, it is stored forever and cannot be deleted (even accidentally) without this being detected. Moreover, the database is transparent, i.e., there is no filtering, and anybody can access all the data without any possibility of censorship. The BigchainDB setup is entirely decentralized, and there is no single point of control or failure as multiple nodes are present in a single cluster voting for decisions according to a modified version of the PBFT consensus protocol. Thus, in this case, we formed a consortium among the four institutions above to guarantee decentralization, transparency, and trustless archival. The BigchainDB 2.0, which we have used for our set up, is also Byzantine fault-tolerant - up to one-third of the network nodes can fail, and the system will keep functioning. Furthermore, BigchainDB is amenable to customization: it exposes a built-in HTTP API on top of which it is possible to build a higher-level API, thus allowing complete control over custom requirements, like restricting access to a client or exposing only limited features in a production environment.

4.4 Proposal Summary

We begin by addressing the immutability and transparency challenges related to data storage in the cloud. Cloud storage adoption, due to the growing popularity of IoT solutions, is steadily on the rise and ever more critical to services and businesses. In light of this trend, customers of cloud-based services are increasingly reliant, and their interests correspondingly at stake, on the good faith and appropriate conduct of providers at all times, which can be misplaced considering that data is the "new gold," and malicious interests on the provider side may conjure to misappropriate, alter, hide data, or deny access. A key to this problem lies in identifying and designing protocols to produce a trail of all interactions between customers and providers, at the very least, and make it widely available, auditable, and its contents therefore provable. Considering the above scenario, we explore blockchain as a possible solution to address the above limitations. We present a blockchain-based solution to solve the problem of trust in cloud storage. We describe the use case scenarios, threat models, architecture, interaction protocols, and security guarantees and discuss how the proposal addresses the literature's challenges.

Further extending the experience of securing the stored data, we focus on securing the entire supply chain of data production and storage. We propose building an IoT data collection framework that is decentralized (without any single point of failure), transparent (so that everything could be easily verifiable by everybody), and trustless (no requirement for any involved entity to be fully trusted, for the system to work). Decentralization, transparency, and trustlessness should be built-in at all steps so that there is no possibility of data forging, alteration, and/or censorship. We show how the IoT Data collection platform of the #SmartME project has been revised and extended by including a trustless system engaging each stakeholder (the University of Messina, the Messina Municipality) in the data storage and protection duties. This is achieved by introducing security features at different levels and enabling multiple entities and groups to participate at all levels of the data processing and consumption pipeline, as represented by the icons on the right side of Fig. 4.1. At the data origin, the Arancino platform is introduced that is equipped with a cryptochip partially able to act akin to a TPM (7) chip, e.g., as a secure enclave for keys and on-chip (i.e., host-invisible) crypto operations (signing, authenticating, encrypting, etc.), thus ensuring that known and authorized sources produce each piece of data. At the data storage level, an adapted version of the BigchainDB platform is proposed. Based on blockchain technology, BigchainDB guarantees decentralization (no single entity controls the data), scalability (multiple endpoints to which data can be delivered), and, last but not least, data immutability. In particular, concerning the latter, in the context of Open Data collection for Smart City services, it is of utmost importance that data may be relied upon as pristine along with all phases of the acquisition and storage process. Indeed, immutability lends Open Data-based services suitable reliability guarantees, possibly extending data usage scope to legally binding processes as well, where any lesser standing in terms of genuine origin for data would be an unacceptable compromise.

Chapter 5

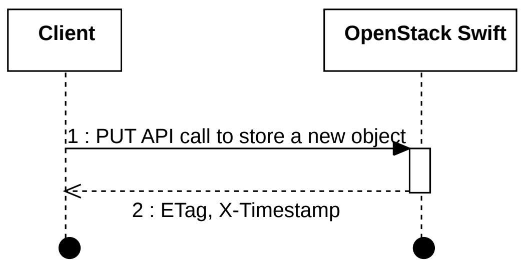
Provable Data-Storage Security

This chapter begins by addressing the immutability and transparency challenges related to data storage in the cloud. Cloud storage adoption, due to the growing popularity of IoT solutions, is steadily on the rise and ever more critical to services and businesses. A protocol enabling the auditing of interactions between customers and providers, at the very least, and making it widely available and its contents therefore provable, is desired. Considering the above scenario, we explore blockchain as a possible solution to address the above requirements. We present a blockchain-based solution (14) to solve the problem of trust. For reference, we consider Openstack Swift cloud storage, but the proposal can be applied to any cloud storage solution. We begin by analyzing the standard behavior of the Swift service and the limitations of the existing design. Then, we present a reference architecture to overcome the shortcomings mentioned above. Finally, we present a performance analysis of the proposed solution and compare it to the existing methodology.

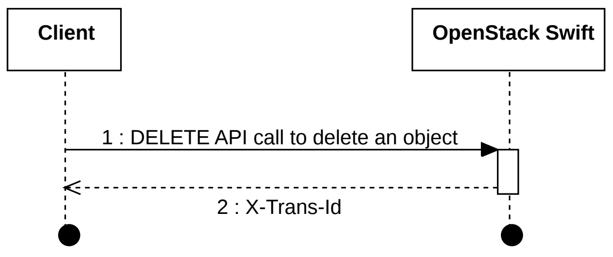
5.1 Existing storage solution

Figure 5.1a depicts the sequence of interactions between a client and the Swift service to store a new object. OpenStack Swift allows a client to store an object using API calls. In order to deposit a file in the storage, the client needs to invoke PUT API. Once the file is stored successfully, Swift returns an MD5 hash of the content of the file and the timestamp of initial object creation in addition to other details. Similarly in case of updating the file, the client needs to invoke PUT API with the modified file. The Swift service replaces the file and returns the MD5 hash of the new file and the timestamp of initial object creation in addition to other details. Finally, in case of deletion of an object, the client invokes a DELETE API call to request the Swift service to delete the object from the storage. The Swift service queues the request and returns a transaction identifier which needs to be saved for reference in case of any issues (Figure 5.1b).

We can observe various security loopholes in the existing interaction pattern. In particular, with our proposed systems we intend to address the following limitations: *(i)* A malicious user may upload a file to the cloud and later on claim that the file present in the cloud is not the same as the one uploaded by the client. Our system should support non-repudiation by



(a)



(b)

Figure 5.1: (a) Object insertion in Swift using API call, and (b) Object deletion in Swift using API call.

the client. *(ii)* A malicious cloud service may deny that the user has uploaded a file while the user has actually uploaded the file. Our system should also support non-repudiation by the cloud vendor. *(iii)* A malicious cloud service may modify the object stored by the client by adding, modifying or deleting data from some blocks stored in the cloud. The service may then claim that the data has not been altered by anyone. Our system should support publicly provable data integrity for dynamic data.

5.2 Assumptions and Security Guarantees

The reference architecture guarantees security of the system against attacks operated by the client (e.g., repudiation by client) and by the SCM (e.g., repudiation by SCM). We assume that the public keys of the entities are known to all. We assume that the involved entities may misbehave by partially executing the operations of the protocol. We assume that the SCM can falsely deny the existence of files stored by the client or return a modified file. The proposed solution publicly logs any actions taken on the file and, thus, any misbehaviour can be detected. An SCM which refuses to log is also considered as misbehaviour. We also

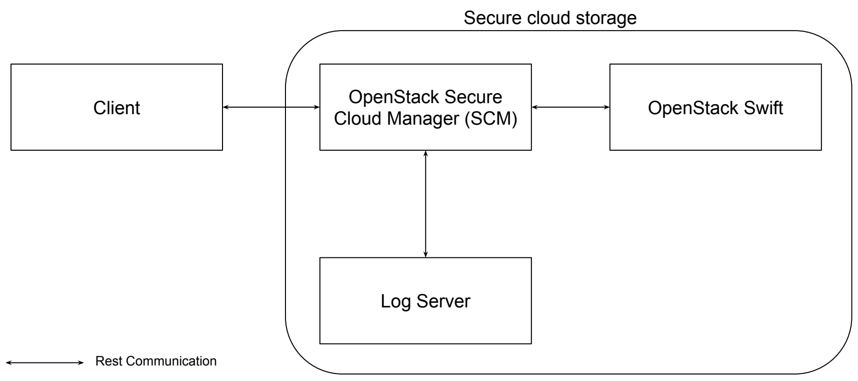


Figure 5.2: Proposed solution architecture.

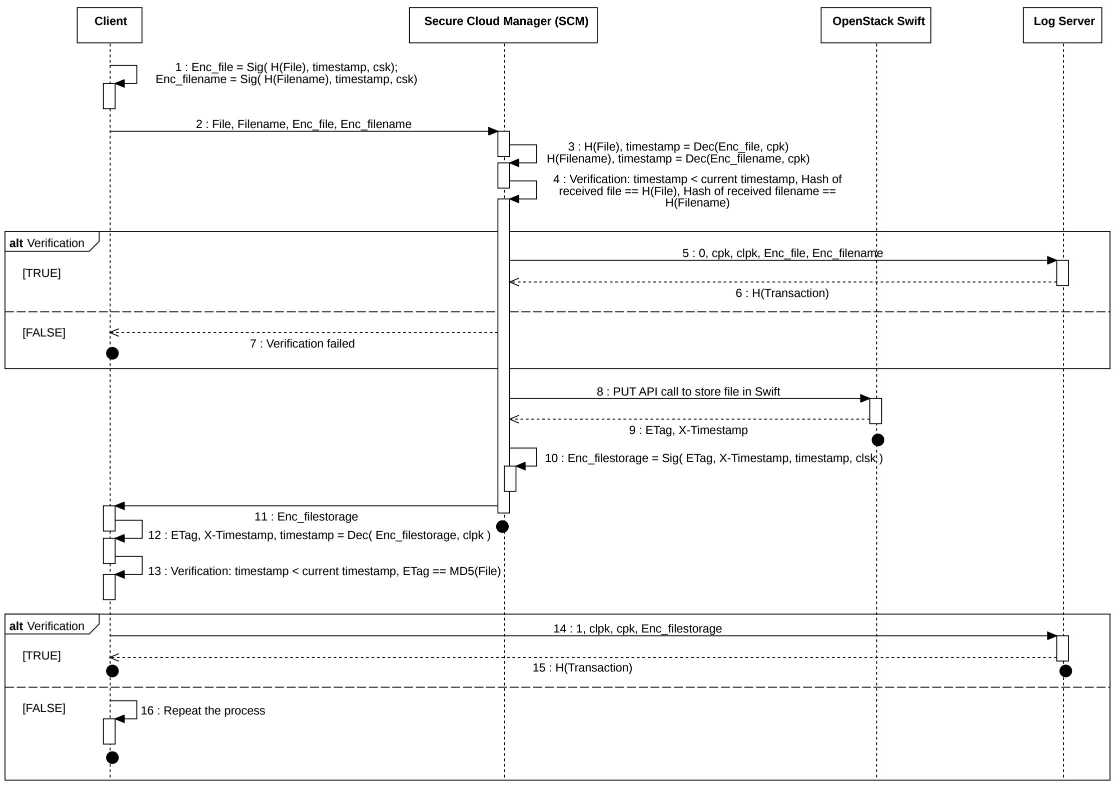

Client
Secure Cloud Manager (SCM)
OpenStack Swift
Log Server
1 : Enc_file = Sig(H(File), timestamp, csk);
Enc_filename = Sig(H(Filename), timestamp, csk)
2 : File, Filename, Enc_file, Enc_filename
3 : H(File), timestamp = Dec(Enc_file, cpk)
H(Filename), timestamp = Dec(Enc_filename, cpk)
4 : Verification: timestamp < current timestamp, Hash of received file == H(File), Hash of received filename == H(Filename)
alt Verification
[TRUE]
5 : 0, cpk, clpk, Enc_file, Enc_filename
6 : H(Transaction)
[FALSE]
7 : Verification failed
8 : PUT API call to store file in Swift
9 : ETag, X-Timestamp
10 : Enc_filestorage = Sig(ETag, X-Timestamp, timestamp, clsk)
11 : Enc_filestorage
12 : ETag, X-Timestamp, timestamp = Dec(Enc_filestorage, clpk)
13 : Verification: timestamp < current timestamp, ETag == MD5(File)
alt Verification
[TRUE]
14 : 1, clpk, cpk, Enc_filestorage
15 : H(Transaction)
[FALSE]
16 : Repeat the process

Figure 5.3: Object insertion in proposed solution.

assume that the client can misbehave by falsely accusing an SCM. The proposed solution, with the support given by logs, enables an SCM to claim no misbehaviour on its part. We present a secure cloud storage system as an extension to OpenStack. First, we describe the proposed model interfacing OpenStack to ensure security guarantees. Second, we outline the primary operations with SCM. Third, we present the misbehaviour situations handled by the system.

5.3 Architecture

In this section, we present our proposed solution (Figure 5.2). The SCM is responsible for handling the cryptographic primitives to ensure the integrity of data as well as create an audit trail together with the client. After ensuring valid sequence of operations, SCM calls the Swift API to perform CRUD operations on the file in the cloud. The log server is an Ethereum based blockchain which maintains the transactions in a public ledger verifiable by anyone. Figure 5.4 shows the format of the log present in the blockchain. The transaction id is an indicator of the type of the transaction. We have four types of transactions, namely metadata, insert, update, and delete. The next field is the public key of the entity sending the message. Third field is the public key of the entity receiving the message and the last field is the data. The data field holds the authenticated data structure for provable guarantee.

The first three fields of the log are indexed. Any third-party can access the logs and verify the legitimacy of any operation using the data structures present in the log.

5.3.1 Auditable Secure Storage

The workflow of operations that the entities should follow to store a file, depicted in Figure 5.3, include the following: The client hashes the file and filename and together with the current timestamp signs them individually using his/her private key. The client then sends the file together with the two signed hashes to the SCM. SCM, after receiving the file and the signed hashes, verifies the filename and the file content using the public key of the client. If the verification is successful, SCM inserts a metadata transaction into the log server. This enables any third-party auditor to verify the interactions, thereby ensuring non-repudiation of the client. SCM receives the transaction hash from the Ethereum-based Log Server to enable the SCM to verify if the transaction has been added to the blockchain. SCM then calls Swift API to store the file in the object storage. On successful storage, the Swift service returns MD5 hash of the file content named ETag and initial object creation time. SCM then signs the ETag, X-Timestamp and current timestamp with its private key and sends it to the client. On receiving the signed message, the client verifies the content of the stored file by comparing the latter with the actual file, to ensure data integrity. On successful verification, the client sends an insert transaction to the log server. This ensures non-repudiation guarantee of the SCM.

For modifying an existing file, the sequence of operations is presented as follows. The client, trying to modify a file existing in the cloud storage, first the hash of the new file content. The hash of the filename remains the same. The client then sends the modified file together with the signed file and filename hash to the SCM. The SCM verifies the file and the content of the received file by using the public key of the client. If the verification is successful, the SCM makes an entry into the log server with transaction id as zero, the sender key as the client's public key, the receiver key as his/her own public key and the signed hashes as the data. The log server will return a transaction hash to enable the SCM to verify the status of the transaction. The SCM then calls the Swift API to update the file and in return receives the modified E-Tag and the X-Timestamp from the Swift service. The X-Timestamp remains the same, i.e. the timestamp the object was created for the first time. The SCM then signs the new ETag, X-Timestamp and the current timestamp with his/her own private key and sends it back to the client. The client verifies the content of the message by comparing the new ETag with the MD5 hash of the modified file. On successful verification, the client sends a transaction with id as two, sender's key as SCM's public key, receiver's key as his/her own public key and data as the signed message. The log server

Transaction Id	Sender Public Key	Receiver Public Key	Data

Figure 5.4: Format of log stored in the log server.

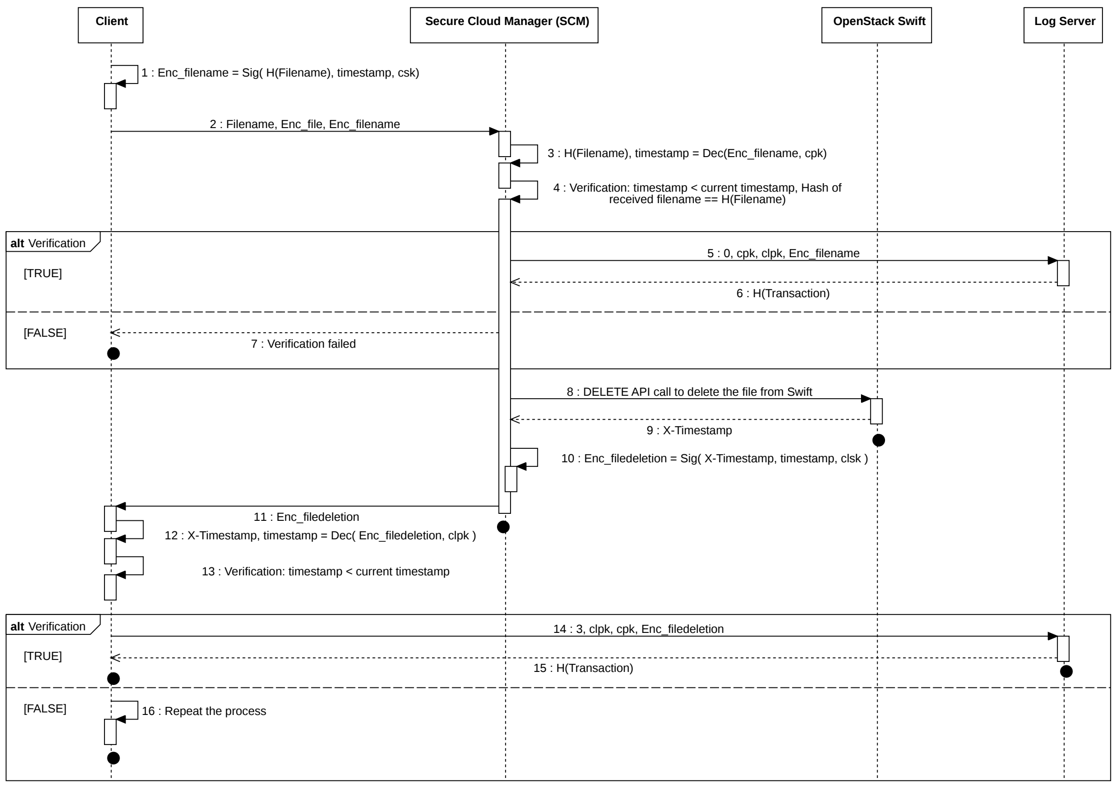

Client
Secure Cloud Manager (SCM)
OpenStack Swift
Log Server
1 : Enc_filename = Sig(H(Filename), timestamp, csk)
2 : Filename, Enc_file, Enc_filename
3 : H(Filename), timestamp = Dec(Enc_filename, cpk)
4 : Verification: timestamp < current timestamp, Hash of received filename == H(Filename)
alt Verification
[TRUE]
5 : 0, cpk, clpk, Enc_filename
6 : H(Transaction)
[FALSE]
7 : Verification failed
8 : DELETE API call to delete the file from Swift
9 : X-Timestamp
10 : Enc_filedeletion = Sig(X-Timestamp, timestamp, clsk)
11 : Enc_filedeletion
12 : X-Timestamp, timestamp = Dec(Enc_filedeletion, clpk)
13 : Verification: timestamp < current timestamp
alt Verification
[TRUE]
14 : 3, clpk, cpk, Enc_filedeletion
15 : H(Transaction)
[FALSE]
16 : Repeat the process

Figure 5.5: Object deletion in proposed solution.

returns the transaction hash to keep track of the log. Similarly Figure 5.5 depicts the sequence of operations for deleting a file from our proposed solution. The client is responsible for signing the hash of the filename with his/her private key and send it together with the request to the SCM. The SCM verifies the filename with the signed hash by using the public key of the client. On successful verification, the SCM inserts a metadata transaction into the log server and receives a transaction hash for verification. The SCM then calls the DELETE API with the name of the file and the Swift service returns X-Timestamp and X-Trans-Id in addition to other things. The X-Timestamp is the time of creation of the file and the X-Trans-Id is a reference number provided by the Swift service. The SCM then signs the X-Timestamp and the current timestamp with his/her private key and sends it to the client. The client verifies the deletion using the public key of the SCM and if successful, store a delete transaction (id as 3 and signed message as data) in the log server. The server returns a hash for transaction verification.

5.3.2 Addressing misbehaviour

Any behaviour deviating from the above description is considered misbehaviour and this section outlines the countermeasures in our proposed system against misbehaviour.

Figure 5.6 represents the series of operations to accuse a misbehaving SCM or a Cloud service. The misbehaviour is in terms of client requesting a file and the SCM replying with

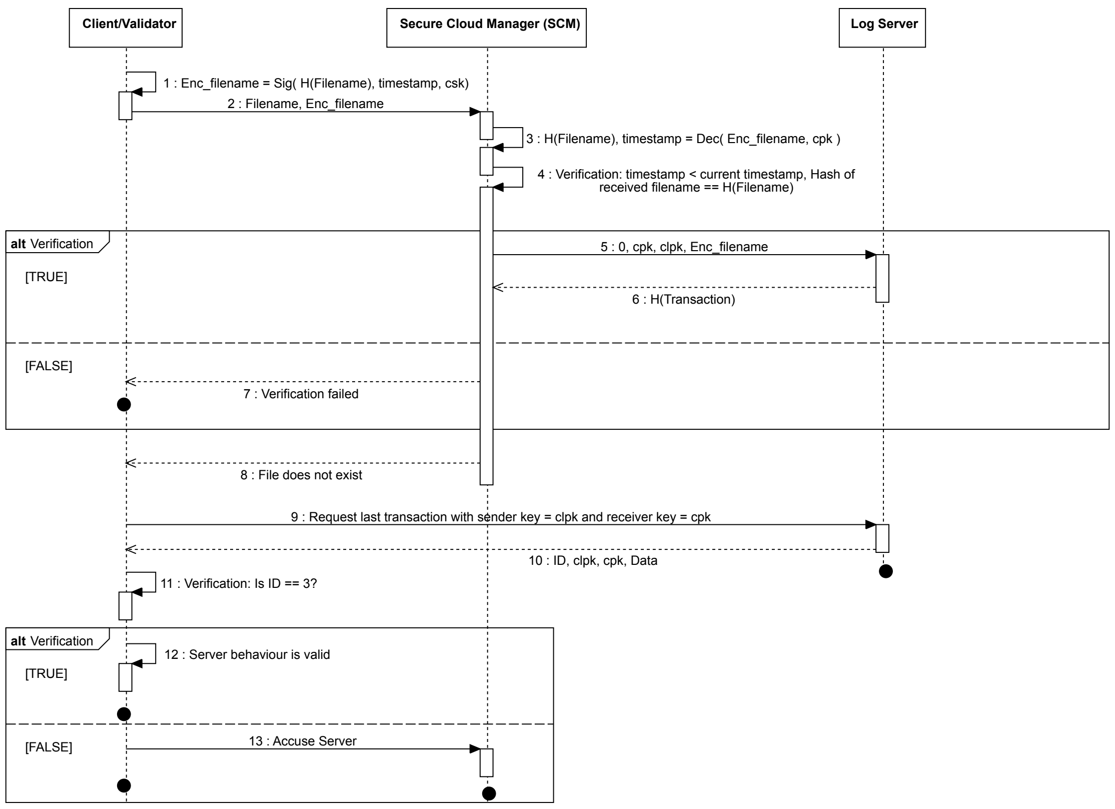

Client/Validator
Secure Cloud Manager (SCM)
Log Server
1 : Enc_filename = Sig(H(Filename), timestamp, csk)
2 : Filename, Enc_filename
3 : H(Filename), timestamp = Dec(Enc_filename, cpk)
4 : Verification: timestamp < current timestamp, Hash of received filename == H(Filename)
alt Verification
[TRUE]
5 : 0, cpk, clpk, Enc_filename
6 : H(Transaction)
[FALSE]
7 : Verification failed
8 : File does not exist
9 : Request last transaction with sender key = clpk and receiver key = cpk
10 : ID, clpk, cpk, Data
11 : Verification: Is ID == 3?
alt Verification
[TRUE]
12 : Server behaviour is valid
[FALSE]
13 : Accuse Server

Figure 5.6: Protection against secure server misbehaving.

a file that does not exist, even though the file exist. The initial operations involve the client signing the hash of filename and timestamp with his/her private key and sending the retrieval request to the SCM. The SCM validates the received message with the client's public key and stores a metadata transaction into the log server. The log server returns the hash of the transaction to validate the addition of the transaction into the blockchain. After the validation, the SCM returns a file not found message to the client. On receiving the message, the client queries the log server with the sender key as the public key of the SCM and the receiver key as his/her own key and finds the last transaction. The client verifies the data field of the transaction with the public key of the SCM and checks if the last transaction id is 3. Last transaction being 3 signifies the file is deleted and the SCM is behaving fairly. Otherwise the client can accuse the SCM.

Another scenario is about a client falsely accusing the server of misbehaviour. Figure 5.7 represents the series of operations to resist such behaviour. The initial interactions remain the same till the SCM sends the metadata transaction to the log server and receives a transaction hash in return to validate the transaction. The SCM then returns the file requested by the client. The client then accuses the SCM of misbehaviour. The SCM retrieves the last transaction from the log server with the sender key as his/her own public key and the receiver key as the public key of the client. The SCM also calls the HEAD API of the Swift service to receive the latest ETag of the file in the storage. The SCM then

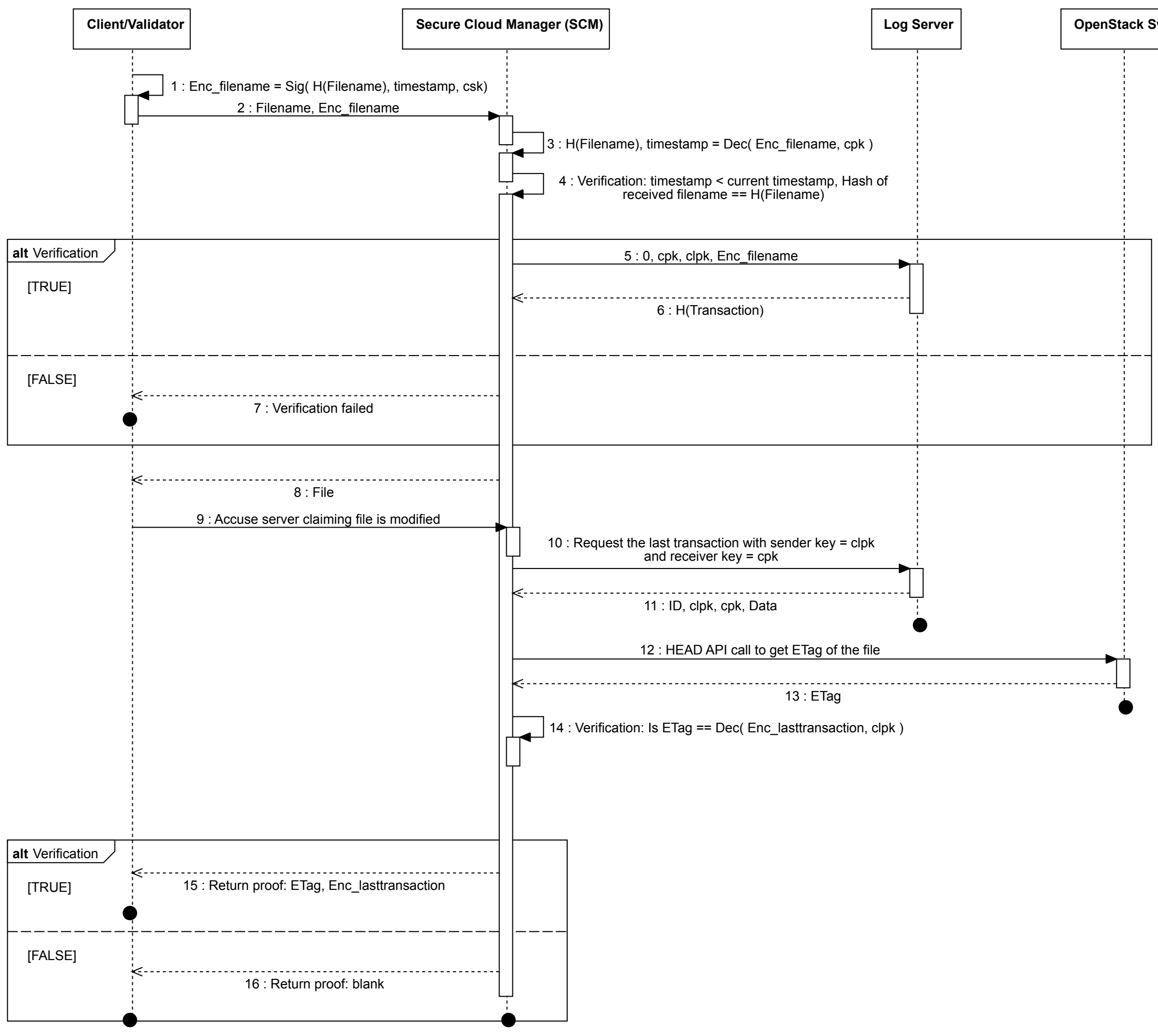

Client/Validator
Secure Cloud Manager (SCM)
Log Server
OpenStack Swift
1 : Enc_filename = Sig(H(Filename), timestamp, csk)
2 : Filename, Enc_filename
3 : H(Filename), timestamp = Dec(Enc_filename, cpk)
4 : Verification: timestamp < current timestamp, Hash of received filename == H(Filename)
alt Verification
[TRUE]
5 : 0, cpk, clpk, Enc_filename
6 : H(Transaction)
[FALSE]
7 : Verification failed
8 : File
9 : Accuse server claiming file is modified
10 : Request the last transaction with sender key = clpk and receiver key = cpk
11 : ID, clpk, cpk, Data
12 : HEAD API call to get ETag of the file
13 : ETag
14 : Verification: Is ETag == Dec(Enc_lasttransaction, clpk)
alt Verification
[TRUE]
15 : Return proof: ETag, Enc_lasttransaction
[FALSE]
16 : Return proof: blank

Figure 5.7: Protection against client misbehaving.

tries to verify if the ETag is the same as the signed data of the last transaction using the client's public key. If successful, the SCM returns the ETag and the signed data as a proof of innocence. If the SCM does not return anything, this proves the misbehaviour of the client.

5.4 Results

In this section we discuss the proof-of-concept system we developed and its performance comparison to standard Swift API call. We selected execution time as the measure for comparison. We developed the proposed system using Javascript client and Express server. For log server, we used Ethereum based blockchain. For OpenStack Swift, we used a VM with OpenStack installation on our local machine. We used files ranging from 1MB to 100 MB to estimate the execution time of the proposed system. We present initial results in Table

File Size	**API (msec)**	**Proposed Solution (msec)**
1 MB	138	2367
10 MB	509	4837
25 MB	510	9958
50 MB	652	19276
100 MB	1449	54192

Table 5.1: Execution time of standard and proposed solution with varying file size.

5.1. The proposed system has polynomial time complexity with respect to the increasing file size. Thus, we can conclude that with the overhead of reasonable computation, the proposed solution can guarantee the security requirements.

5.5 Conclusion

In this chapter, we laid out the proposal to include security countermeasures against the customer and the provider misconducts, in a popular opensource instance of cloud (object) storage service, OpenStack Swift. In particular, we devised threat models, interaction protocols and security guarantees of a solution which extends Swift by leveraging distributed ledger technologies (DLT) to achieve auditability and non-repudiation. We concluded by presenting a performance analysis of the execution time required to store the data. With reasonable overhead, the result shows the viability of the proposed solution.

Chapter 6

Provable Data Supply Chain Security

In the previous chapter, we addressed the immutability and transparency challenges of data storage in the cloud. The proposal enables auditing of the malicious operations at the data plane. This chapter extends the security of data across data and network plane by focusing on securing the entire supply chain of data from production to storage. We propose building an IoT data collection framework (16) that is decentralized (without any single point of failure), transparent (so that everything could be easily verifiable by everybody), and trustless (no requirement for any involved entity to be fully trusted, for the system to work). Decentralization, transparency, and trustlessness should be built-in at all steps so that there is no possibility of data forging, alteration, and/or censorship. We show how the IoT Data collection platform can be revised and extended by including a trustless system engaging each stakeholder (the University of Messina, the Messina Municipality, etc.) in the data storage and protection duties. This is achieved by introducing security features at different levels and enabling multiple entities and groups to participate at all data processing and consumption pipeline levels. At the data origin, the Arancino platform is introduced that is equipped with a cryptochip partially able to act akin to a TPM (7) chip, e.g., as a secure enclave for keys and on-chip (i.e., host-invisible) crypto operations (signing, authenticating, encrypting, etc.), thus ensuring that known and authorized sources produce each piece of data. At the data storage level, an adapted version of the BigchainDB platform is proposed. Based on blockchain technology, BigchainDB guarantees decentralization (no single entity controls the data), scalability (multiple endpoints to which data can be delivered), and, last but not least, data immutability.

6.1 Overview

In this section, we present the overview of the components involved and the reasoning behind the design choices. We being by describing the BigchainDB storage setup. BigchainDB runs on a cluster, wherein a minimum of four nodes are required. In our case, we formed a four node cluster, with each node being hosted independently by an institution in the territory of Messina, i.e., the Computer Center of the University of Messina (CIAM)[1], the Department

[1]https://www.unime.it/it/centri/ciam

of Engineering at the University of Messina[2], the Messina Municipality[3], and the "Horcynus Orca" Community Foundation of Messina[4]. Obviously, apart from neighboring locations, all within the city of Messina, the aforementioned institutions do not have otherwise anything in common, i.e., non-overlapping governance, mission, etc. This is to say that it is highly unlikely for any of the four institutions under consideration to collude with any other, including CIAM and the Department, which are both under the University rule, but run by separate governance bodies. In our use case, along with immutability of data stored in the database, it is of equal importance that data originates from a verifiable and identifiable valid source (an authorized Arancino-powered IoT node). There should be no possibility for a rogue or fake node to pose as a real node, obfuscating the stored data. The default HTTP API exposed by BigchainDB is minimal and not meant for source identity verification or to restrict access in the networking layer. Rather, it is designed to be extended with a higher-level layer which fulfills the custom requirements for the use case at hand. We designed our own custom API using the Flask framework in Python, and we used the NGINX reverse proxy server along with the Consul DNS resolving system to accomplish identity and data verification, and meet the networking requirements of our system.

It is important to note that we aim for *dual* verification - one in the data plane, and the other one in the networking plane - of domain identity, as a requirement. In the data plane, for verification of the origin of data, the Ed25519(110) digital signature scheme is used. During the initial setup, each IoT node is assigned a permanent private key, stored on the cryptochip and thus, unreadable to prying eyes of any malicious actor able to access physically the node and its flash storage. On the other hand, the corresponding public key is known to all the stakeholders in the system. Any data stored inside BigchainDB is signed with the private key of the data source, and thus, can be verified using the corresponding public key. Our custom Flask API leverages a Python implementation of Ed25519 which provides functions to verify data against public keys and signatures. From a security perspective, it is desirable to restrict access to our custom API, and subsequently to BigchainDB, to authorized devices only. This is to thwart any potential network security attacks, like denial of service or unauthorized write access to data. In the networking plane, we achieve this objective by suitably configuring the NGINX reverse proxy server. In particular, NGINX is configured to enforce TLS validation of client certificates: only IoT nodes equipped with valid certificates issued by a trusted CA are able to interact with the APIs exposed by the BigchainDB nodes.

To make the system truly decentralized, it is necessary to ensure that there is no single point of failure, even from a networking perspective. Thus, instead of hard-coding the addresses of the BigchainDB nodes in the IoT nodes, it is desirable to keep the endpoints dynamic. This is meant to boost security, as well as for load balancing and in general achieving higher availability. As a matter of fact, any BigchainDB node could have a downtime, due to, i.e., being compromised, physically shut down, or damaged. Therefore, as an essential system requirement there is need for a discovery service which, upon request, can

[2]https://www.unime.it/it/dipartimenti/ingegneria
[3]http://www.comunemessina.gov.it
[4]http://www.fdcmessina.org

provide the IP addresses of BigchainDb nodes, which are available and working at service querying time, while constantly keeping tabs on the health of nodes. Hashicorp's Consul is a distributed, highly available system which provides a fully featured control plane with provisions for service discovery and health checks. Clients - BigchainDB nodes in our case - where a consul agent is deployed, can register services and then, using Consul DNS and HTTP APIs, it is possible to discover the providers of any such service. Consul features various health check capabilities and returns only available, healthy nodes. Like BigchainDB, Consul also operates in a distributed fashion and supports a cluster of servers.

As each node which provides services to be exposed by Consul runs a Consul agent, all the four BigchainDB nodes in our system run this agent. Though Consul itself may work off a single server, it is recommended to run it in a cluster configuration, to avoid data loss in case of node failure - hence we opted to run Consul servers on each of the BigchainDB nodes as well. The servers are responsible for replication of data, and to actually discover services, while the agent are responsible for reporting the health status of the local node on which they are deployed. Agents can function by interacting with a single server or a cluster of servers as per requirement.

6.2 System Architecture

The resulting deployment, after accounting for all the requirements of the system and the tools required for fulfilling them, involves a four node cluster of BigchainDB nodes each running on independent machines along with several instances of networking and data tools, which are detailed in the following section. Fig. 6.1 depicts the deployment and interactions of the developed system. In our setup, the #SmartME testbed comprising of various IoT devices with sensors and actuators, represent 'Smart City' of the figure context. Each node is running Ubuntu 17.04, a server-class operating system. Only listening ports for select services are open, to reduce the attack surface and the risk of any unauthorized access. The IP address of each node is known to all the other nodes as it is required to form a cluster. BigchainDB again has three independent components which must be able to communicate with each other for it to function - the MongoDB server, the Tendermint instance, and the BigchainDB server instance itself, which communicates with Tendermint and MongoDB and exposes the BigchainDB HTTP API. Thus each of these three separate instances is running inside any single node at all times.

As the default HTTP API of BigchainDB has limited capabilities, and by design it is intended for programmers to develop their applications on top of that API, our custom API - implemented as a Flask-based application server wrapping the HTTP API of BigchainDB, is responsible for the processing of data, as well as registration of IoT nodes. It has end-points meant for retrieval of data from BigchainDB in a meaningful manner i.e. returning data for specific time intervals, sensors or a other complex database queries. The default HTTP API of BigchainDB is limited also in its querying capabilities hence the custom API allows interaction directly with MongoDB instance on the node for read-only queries. In the original #SmartME infrastructure, an interactive map visualization layer is available, which displays markers for all the deployed IoT nodes, according to their location, as shown in

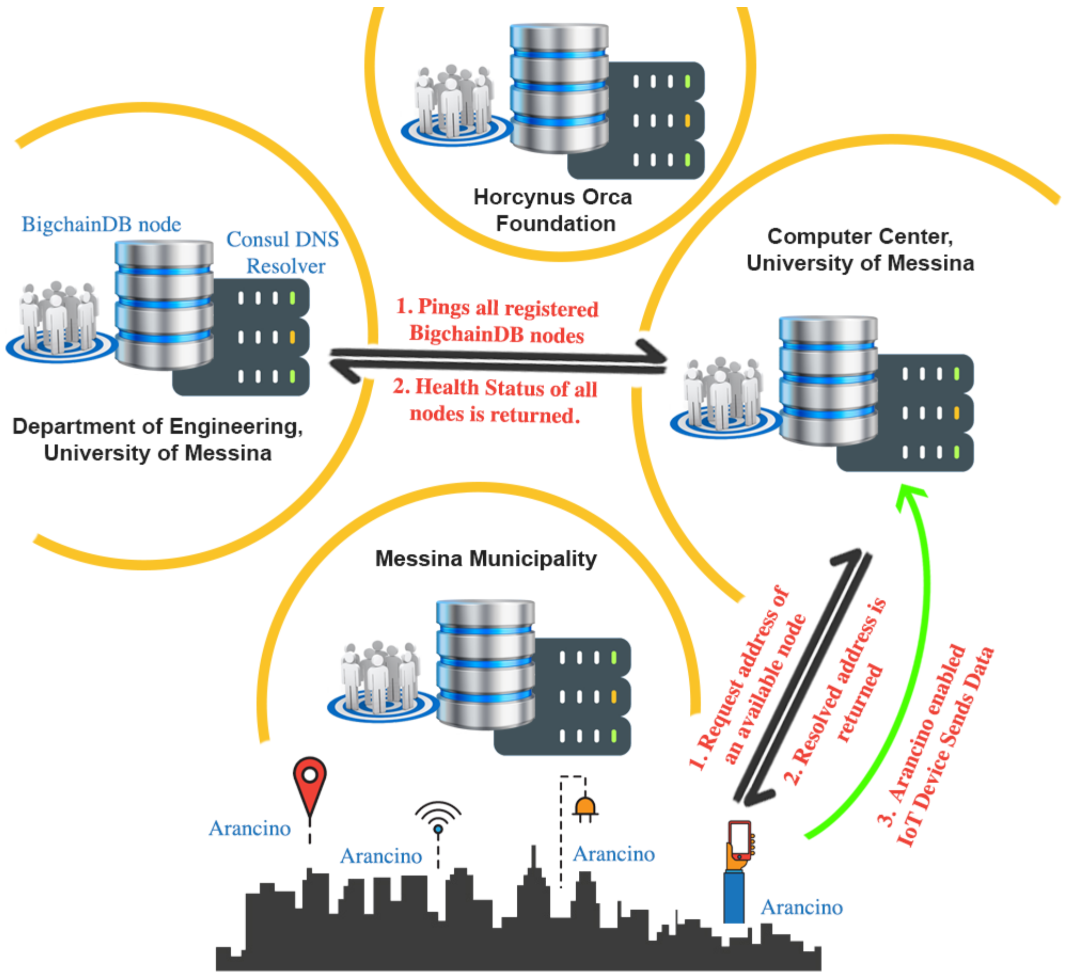

Horcynus Orca Foundation
BigchainDB node
Consul DNS Resolver
Department of Engineering, University of Messina
Computer Center, University of Messina
1. Pings all registered BigchainDB nodes
2. Health Status of all nodes is returned.
Messina Municipality
1. Request address of an available node
2. Resolved address is returned
3. Arancino enabled IoT Device Sends Data
Arancino
Arancino
Arancino
Arancino

Figure 6.1: System deployment and interactions.

Fig. 6.2, and it is possible for users to click on the markers to get current and past readings from node-hosted sensors in a user-friendly (e.g., scrollable, zoomable) fashion. The original #SmartME architecture employs a SQL database for data storage, whereas MongoDB is a NoSQL database, hence the code and queries were rewritten to work with MongoDB. To summarize, at the data visualization and auditing levels we were able to reuse this visualization portal just by connecting it to BigChainDB instead of the CKAN instance. The portal is written in JS as a fully client-side application so there is nothing in the middle between the browser and the BigChainDB consortium, and no censorship is possible.

While retaining all the previous features of the interactive map, an added functionality has been added: while viewing the scrollable graphs out of sensor timeseries, it is now possible to click on any of the datapoints to open up a data verification wizard which guides the user in the verification process for that particular datapoint, as shown in the Fig. 6.3. The map

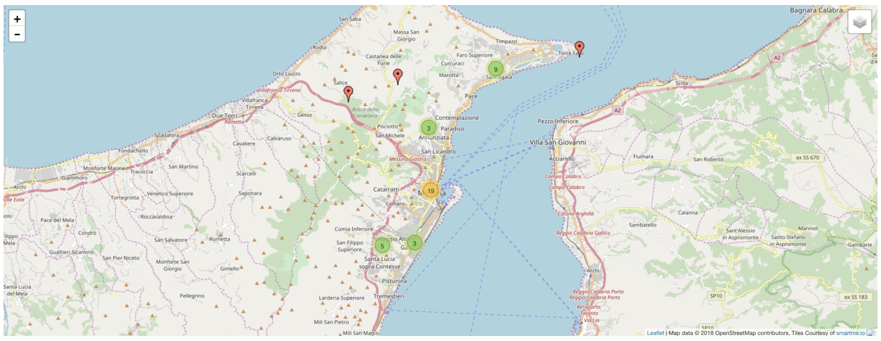


Figure 6.2: Map with sensor markers.

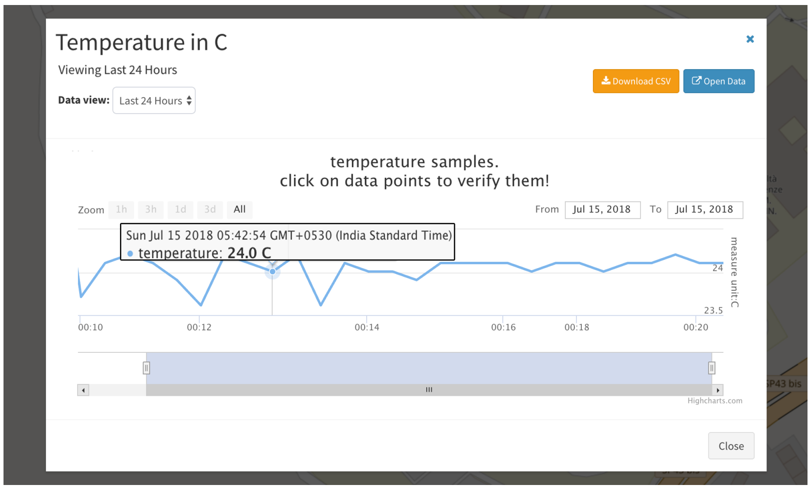
is served through a PHP file which makes AJAX requests to the custom API to fetch all the required data.

In the original #SmartME deployment, a plugin had been injected on each board. As evolution from the original (111) S4T data collection approach, the plugin carried the logic to push any newly collected samples from sensors to a developer-defined endpoint. The endpoint of choice in this case was an instance of the CKAN Content Management System (CMS) for Open Data.

All of the components discussed in this section run simultaneously on the nodes on various default ports. It is of absolute importance that these components should not be accessible to anything outside the node environment. For instance, MongoDB runs on default port 27017, and if the port is open outside interference is possible and can have catastrophic results for the system's security and functioning, while also at the same time, a heterogeneous variety of services are running on the nodes, all of which require access through logically named sub routes (e.g., /map), without concerning the user with the actual ports on which the applications are running. Thus, NGINX is used as a reverse proxy. It serves two primary purposes - restricting access to unwanted devices and ports, and exposing the desired heterogeneous components to the public and the IoT nodes. For instance, all the API endpoints are available at / (root) whereas going to /map redirects the user to standalone map running on the apache server inside the node. Each node thus serves content via an NGINX instance running on it.

6.2.1 Data Flow

The data in its raw form is generated as readings by the sensors of the IoT node. In BigchainDB, the data is structured as assets. An asset can represent any physical or digital value. Thus, raw readings are an asset in our use case. To register an asset in BigchainDB, the client (IoT Node) needs to make a CREATE transaction in the database. Each CREATE transaction has an input and an output. As mentioned previously each IoT node has a private key, and a public key, with the former used to sign data (i.e., the reading), and the latter serving as attestation of the (unique) identity of the node. The input to the CREATE transaction contains the asset (readings of a sensor) and other (meta)data, such as a signature, which serves as proof that data is really being generated by the device which


Temperature in C

Viewing Last 24 Hours

Data view: Last 24 Hours

temperature samples.
click on data points to verify them!

Sun Jul 15 2018 05:42:54 GMT+0530 (India Standard Time)
● temperature: 24.0 C

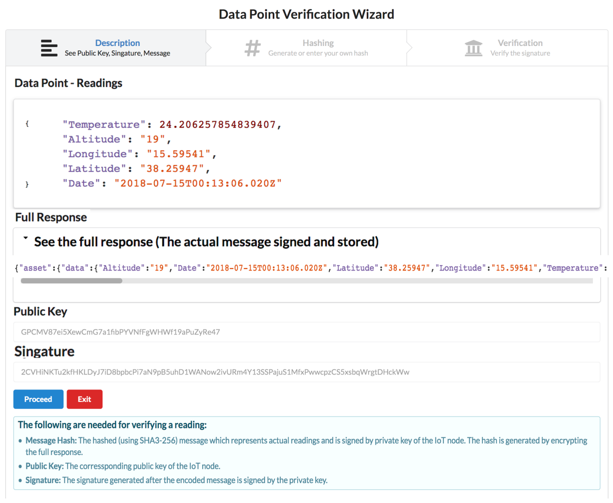
Figure 6.3: Interactive modal showcasing verifiable data points.

is claiming to be the source, and a Transaction ID, which is a hash of the entire transaction input and is used to unambiguously identify a transaction.

The S4T Cloud has been leveraged, in this work, to inject an additional plugin we developed into the IoT node, alongside the one originally deployed for data push to CKAN. This plugin carries the logic to generate a transaction record, following the BigchainDB transaction spec, for each individual reading, and push the record to the BigchainDB node(s). Internally, the entire transaction record before being signed is converted into binary and then is one way encrypted using SHA-256. The generated hash gets signed by the private key which produces a signature. This hashing process will have to be repeated in order to verify the readings at a later stage. Next step involves the IoT node querying the Consul DNS servers, to get a healthy BighchainDB node, and selects one randomly from the results. A valid CREATE transaction is then registered on BigchainDB node cluster. The output of the transaction specifies that the public key mentioned in the transaction indeed belongs to the IoT node which generated the data, and provides a fulfillment object which is composed of the signature and the public key.

6.3 Data Verification Wizard

For verification of a given data point, three entities are essential - (i) SHA-256 hash of the data point, (ii) public key of the IoT Node which generated the data point, and (iii) signature generated when the hash was signed by the corresponding private key of the IoT Node. Hence we designed an intuitive, user-friendly datapoint verification wizard. It has been devised to aid the end-user, even if totally unaware of crypto or validation processes, to audit data independently, through a graphical interface, with an easy point-and-click experience. The wizard operates in three steps: it is launched for a particular data point as soon as it is clicked on, as shown in Fig. 6.3. It is essential to note that a stakeholder in the system need not use this particular verification wizard to verify the data, and can use his own, or widely


Data Point Verification Wizard
Description
See Public Key, Singature, Message
Hashing
Generate or enter your own hash
Verification
Verify the signature
Proceed
Exit

Data Point - Readings

```
{       "Temperature": 24.206257854839407,
        "Altitude": "19",
        "Longitude": "15.59541",
        "Latitude": "38.25947",
}       "Date": "2018-07-15T00:13:06.020Z"
```

Full Response

See the full response (The actual message signed and stored)

```
{"asset":{"data":{"Altitude":"19","Date":"2018-07-15T00:13:06.020Z","Latitude":"38.25947","Longitude":"15.59541","Temperature":
```

Public Key

GPCMV87ei5XewCmG7a1fibPYVNfFgWHWf19aPuZyRe47

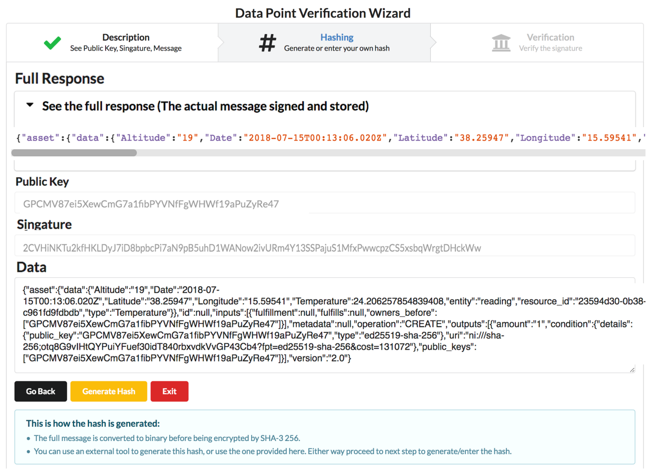
Singature

2CVHiNKTu2kfHKLDyJ7iD8bpbcPi7aN9pB5uhD1WANow2ivURm4Y13SSPajuS1MfxPwwcpzCS5xsbqWrgtDHckWw

The following are needed for verifying a reading:

- **Message Hash:** The hashed (using SHA3-256) message which represents actual readings and is signed by private key of the IoT node. The hash is generated by encrypting the full response.
- **Public Key:** The corressponding public key of the IoT node.
- **Signature:** The signature generated after the encoded message is signed by the private key.

(a)


Data Point Verification Wizard
Description
See Public Key, Singature, Message
Hashing
Generate or enter your own hash
Verification
Verify the signature
Go Back
Generate Hash
Exit

Full Response

See the full response (The actual message signed and stored)

```
{"asset":{"data":{"Altitude":"19","Date":"2018-07-15T00:13:06.020Z","Latitude":"38.25947","Longitude":"15.59541",'
```

Public Key

GPCMV87ei5XewCmG7a1fibPYVNfFgWHWf19aPuZyRe47

Singature

2CVHiNKTu2kfHKLDyJ7iD8bpbcPi7aN9pB5uhD1WANow2ivURm4Y13SSPajuS1MfxPwwcpzCS5xsbqWrgtDHckWw

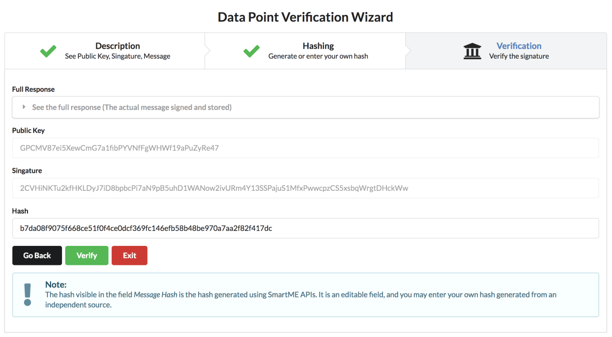
Data

{"asset":{"data":{"Altitude":"19","Date":"2018-07-15T00:13:06.020Z","Latitude":"38.25947","Longitude":"15.59541","Temperature":24.206257854839408,"entity":"reading","resource_id":"23594d30-0b38-c961fd9fdbdb","type":"Temperature"}},"id":null,"inputs":[{"fulfillment":null,"fulfills":null,"owners_before":["GPCMV87ei5XewCmG7a1fibPYVNfFgWHWf19aPuZyRe47"]}],"metadata":null,"operation":"CREATE","outputs":[{"amount":"1","condition":{"details":{"public_key":"GPCMV87ei5XewCmG7a1fibPYVNfFgWHWf19aPuZyRe47","type":"ed25519-sha-256"},"uri":"ni:///sha-256;otq8G9vIHtQYPuiYFuef30idT840rbxvdkVvGP43Cb4?fpt=ed25519-sha-256&cost=131072"},"public_keys":["GPCMV87ei5XewCmG7a1fibPYVNfFgWHWf19aPuZyRe47"]}],"version":"2.0"}

This is how the hash is generated:

- The full message is converted to binary before being encrypted by SHA-3 256.
- You can use an external tool to generate this hash, or use the one provided here. Either way proceed to next step to generate/enter the hash.

(b)


Data Point Verification Wizard
Description
See Public Key, Singature, Message
Hashing
Generate or enter your own hash
Verification
Verify the signature
Go Back
Verify
Exit

Full Response

See the full response (The actual message signed and stored)

Public Key

GPCMV87ei5XewCmG7a1fibPYVNfFgWHWf19aPuZyRe47

Singature

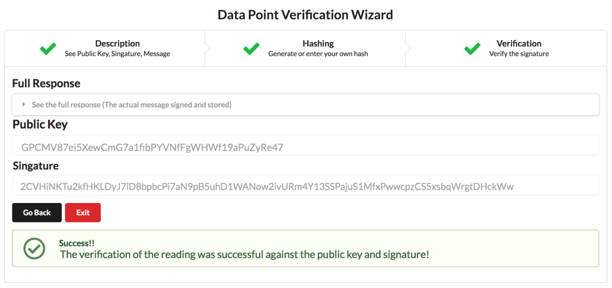
2CVHiNKTu2kfHKLDyJ7iD8bpbcPi7aN9pB5uhD1WANow2ivURm4Y13SSPajuS1MfxPwwcpzCS5xsbqWrgtDHckWw

Hash

b7da08f9075f668ce51f0f4ce0dcf369fc146efb58b48be970a7aa2f82f417dc

Note:
The hash visible in the field *Message Hash* is the hash generated using SmartME APIs. It is an editable field, and you may enter your own hash generated from an independent source.

(c)


Data Point Verification Wizard
Description
See Public Key, Singature, Message
Hashing
Generate or enter your own hash
Verification
Verify the signature
Go Back
Exit

Full Response

See the full response (The actual message signed and stored)

Public Key

GPCMV87ei5XewCmG7a1fibPYVNfFgWHWf19aPuZyRe47

Singature

2CVHiNKTu2kfHKLDyJ7iD8bpbcPi7aN9pB5uhD1WANow2ivURm4Y13SSPajuS1MfxPwwcpzCS5xsbqWrgtDHckWw

Success!!
The verification of the reading was successful against the public key and signature!

(d)

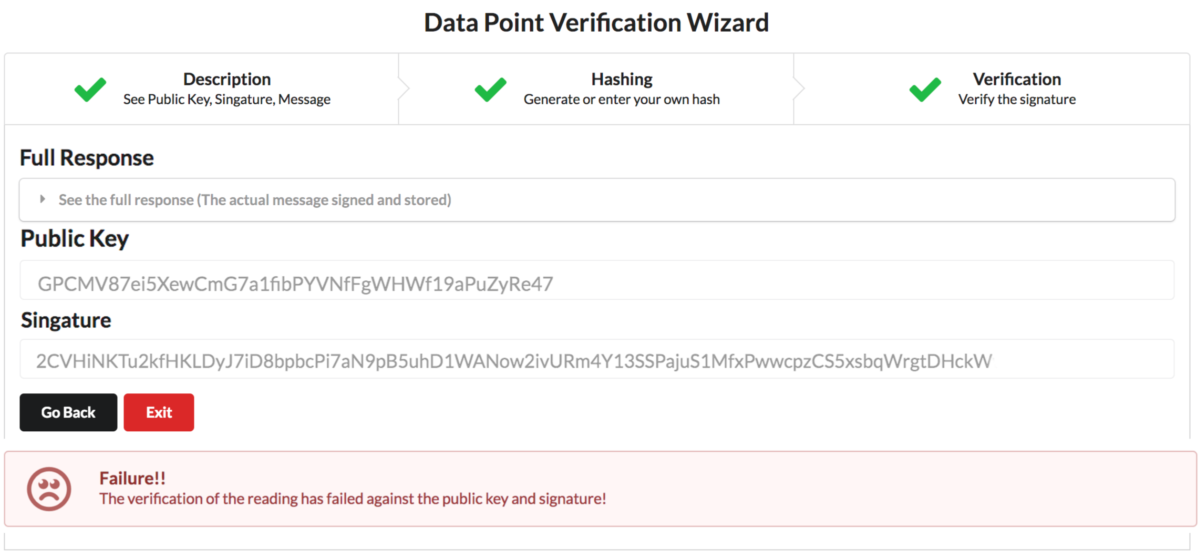
Figure 6.4: (a) Data Verification Wizard Step 1, (b) Data Verification Wizard Step 2, (c) Data Verification Wizard Step 3, and (d) Successful data verification.

available implementations of Ed25519 to verify it independently.

The wizard step 1 as shown in Fig. 6.4a is an informative step. It shows the raw readings, with an option to see the entire response - the actual message which was encrypted as mentioned previously. It also makes it convenient for the user by extracting the signature from the fulfillment object, the public key is displayed as well. These three pieces of information are self-sufficient to verify the validity of the data.

The step 2 of the wizard which is shown in Fig. 6.4b presents an option to the user to generate the SHA-256 hash of the full response and shows the data to be encrypted in a field which is intentionally editable, to allow the user to tamper with the data, and see the outcome of verification.

After generating the hash using the wizard, or entering the SHA3-256 hash generated from an external source (the hash field is also editable), the user lands on step 3, the final step of the verification wizard as shown in Fig. 6.4c. On pressing the verify button the hash is verified against the public key and the signature if the data has not been tampered with, and the public key is indeed of the IoT node which signed the data hash, the verification succeeds, else it fails and the wizard shows the appropriate message as shown in Fig. 6.4d and Fig. 6.5 respectively. This last step again can be carried out independently and there is no compulsion to use this wizard for verifying the hash against the signature.


Data Point Verification Wizard

✓ Description — See Public Key, Singature, Message

✓ Hashing — Generate or enter your own hash

✓ Verification — Verify the signature

Full Response

See the full response (The actual message signed and stored)

Public Key

GPCMV87ei5XewCmG7a1fibPYVNfFgWHWf19aPuZyRe47

Singature

2CVHiNKTu2kfHKLDyJ7iD8bpbcPi7aN9pB5uhD1WANow2ivURm4Y13SSPajuS1MfxPwwcpzCS5xsbqWrgtDHckW

Failure!!
The verification of the reading has failed against the public key and signature!

Figure 6.5: Failure of data verification.

6.4 Performance evaluation

We carry out a number of experiments to assess the scalability of our system. A 4-node BigchainDB cluster is set up to carry out these experiments. IoT nodes could interact with the BigchainDB cluster at an average bandwidth and RTT of 25 Mbps and 125 ms respectively. We consider two experiments to observe the trends in read and write speeds when certain parameters of the system are varied. The frequency at which IoT nodes generate and send readings can vary in a real-world scenario. The first experiment involves varying the sample frequency, and observing corresponding changes in response times to read requests. The plots in Fig. 6.6 show little variation in terms of response times across a range of frequencies, for a given sample size. This is consistent with the fact that the our custom-designed API queries MongoDB directly for read operations on stored data, hence response times are basically independent of the load introduced on the BigchainDB server by increasing data transmission frequency at a given instant. Also, as expected, increasing the size of the query increases the total response time for a read request. In the second experiment we increase the number of IoT nodes, from 1 to 200, and measure the time for the write operations to complete. The plots in Fig. 6.7 show a linear increase in response time to write requests with increasing number of IoT nodes simultaneously transferring data. The sample frequency also impacts write speed. A higher data transmission frequency further stresses

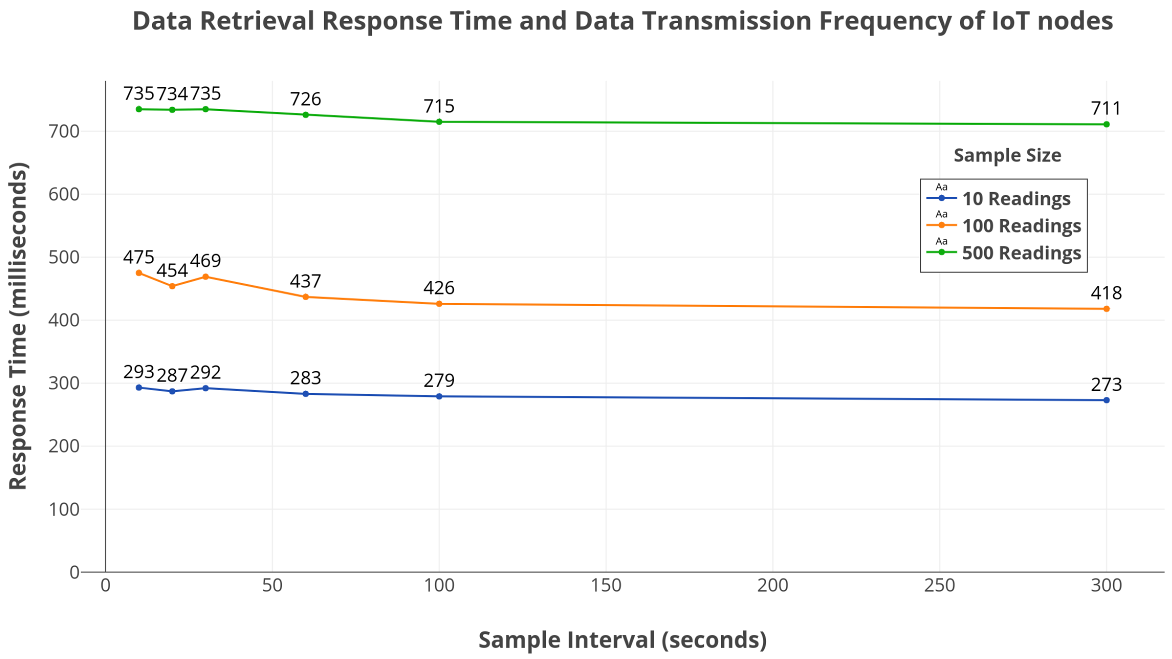


Figure 6.6: Read response times wrt sample interval.

BigchainDB servers, and results in an increased write response time. On the other hand, the hike in response times is not exponential, hence the cluster is scalable to accomodate even more nodes.

While the above observations demonstrated the feasibility of our current set up, we decided to further extend the system in a more controlled environment: while our original testbed consisted of a 4-node BigchainDB cluster, we introduced another variable to the experiments. In addition to varying sample size, and sample frequency, we varied the size of the BigchainDB cluster itself. Starting from 3 BigchainDB nodes, we recorded and read and write response times all the way up to a cluster of 8 BigchainDB nodes. The plot in Fig. 6.8a shows the read response times for different sizes of BigchainDB cluster when the sample size is fixed at 100. Measurements are carried out for IoT nodes sending sample write requests at intervals of 10, 60 and 300 seconds, respectively. If a single cluster size is considered, it can be seen there is no significant fluctuation in the read speeds when varying sample interval at which IoT nodes send the data. This is consistent with the observations made for the plots in Fig. 6.6 and follows similar reasoning. Furthermore, this plot shows that there is no considerable impact from increasing or decreasing the size of the BigchainDB cluster, which is consistent with the fact that a particular read operation is carried out by interacting with MongoDB instance of a single node at a given time, whose IP address was returned by the Consul DNS service. The plot in Fig. 6.8b shows the read response times for different sizes of BigchainDB cluster when the interval at which IoT nodes send data is fixed at 10 seconds. Measurements are carried out for IoT nodes requesting samples carrying 10, 100 and 500 readings, respectively. If a single cluster size is considered, there is a linear increase in the response time with increase in the size of the sample requested. This is consistent with the

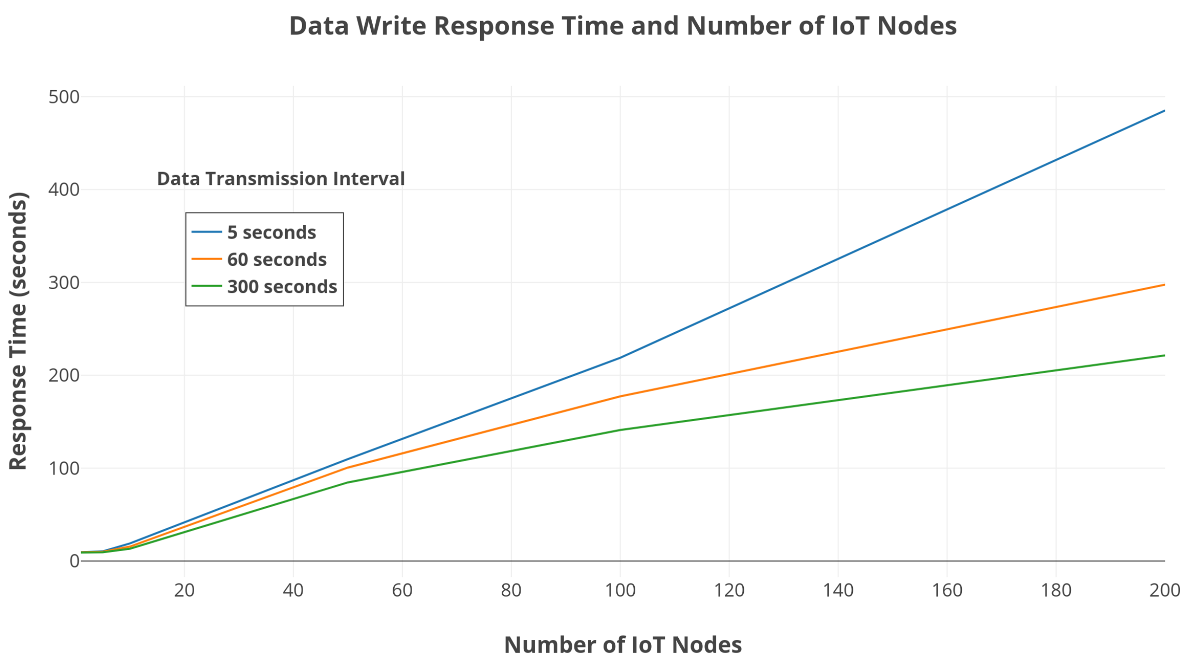
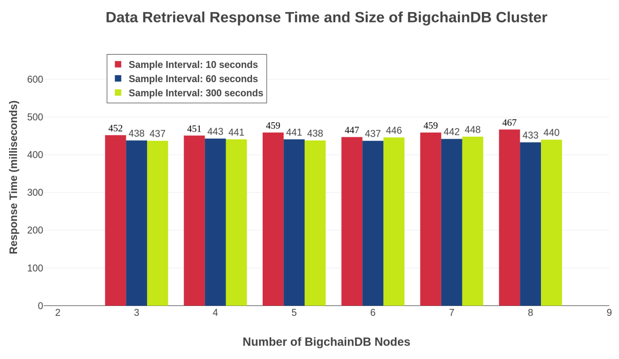
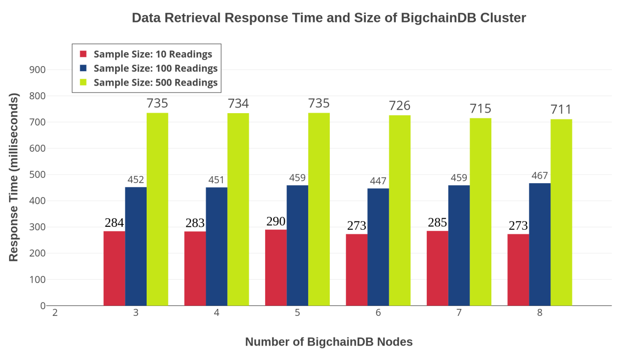


Figure 6.7: Write response times wrt number of active IoT nodes.

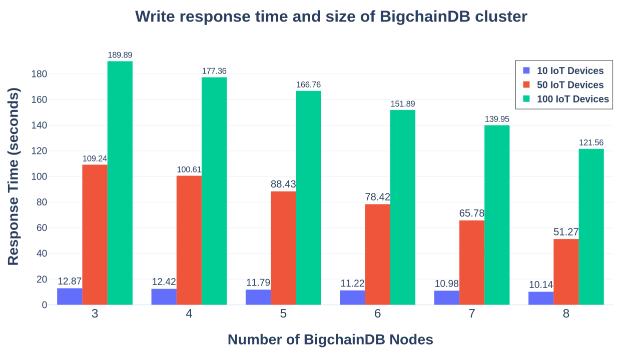
observations made for the plots in Fig. 6.6 and follows similar reasoning about the increase in response time for a bigger chunk of request data whilst available network bandwidth is left unchanged. Furthermore, this plot shows that there is no considerable impact from increasing or decreasing the size of the BigchainDB cluster, which is consistent with the results in Fig. 6.8a due to the fact that a particular read operation is carried out by interacting with MongoDB instance of a single node at a given time. The plot in Fig. 6.8c shows the write response times for different sizes of BigchainDB cluster when the interval at which IoT nodes send data is fixed at 30 seconds, while the sample size is of one reading. Measurements are carried out for 10, 50, and 100 IoT devices. If a single cluster size is considered, there is an increase in the response time with higher numbers of IoT devices which is consistent with the observations made in Fig. 6.7 and follows a similar reasoning. Furthermore, this plot shows that there is an (inverse) linear relationship between the response time of write requests and the number of BigchainDB nodes in the cluster. As the number of nodes in the cluster increases, the response time decreases linearly, which can be also interpreted as an increase in throughput. This linear improvement in performance following an increase in the number of nodes is consistent with the fact that signature verification is a CPU intensive task and, as the number of nodes in the cluster increases, while keeping the rest of the inputs constant, including number of write requests being received in unit time, the verification computation is distributed among a larger number of nodes. Consul, by randomly returning the IP address of any one node in the cluster to handle the write request, enables effective usage of this increased (i.e., pooled) processing power. Now, the same number of write requests are being split up among a larger number of nodes. The plot in Fig. 6.8d shows the write response times for different sizes of BigchainDB cluster when the interval when the number of IoT



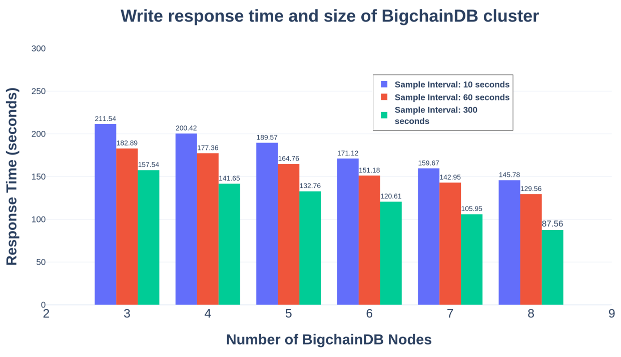
(a)



(b)



(c)



(d)

Figure 6.8: (a) Read response times wrt number of active BigchainDB nodes, (b) Read response times wrt number of active BigchainDB nodes, (c) Write response times wrt number of active BigchainDB nodes, (d) Write response times wrt number of active BigchainDB nodes.

devices is fixed at 100, while the sample size is of one reading. Measurements are carried out for 100 IoT devices sending data at 10, 60, and 300 second intervals. If a single cluster size is considered, there is a decrease in the response time with increase in the interval at which the IoT devices send data which is consistent with the observations made in Fig. 6.7 and follows a similar reasoning. Furthermore, this plot follows the same trend of having an inverse linear relationship between the response time of write requests and the number of BigchainDB nodes in the cluster. As the number of nodes in the cluster increases, the response time decreases linearly following the same reasoning which explains this trend in Fig. 6.8c.

6.5 Conclusions

In this chapter, we presented a trustless approach to acquire, store and consume sensor data, possibly originating from a number of independent stakeholders in the context of a general purpose IoT infrastructure, in order to address issues related to the the integrity, availability and immutability of datasets generated from such sensing activities. In such a context, we demonstrated that DLT-based technologies such as BigchainDB can be suitable to meet these kind of requirements, by designing and implementing a trustless Smart City data acquisition, storage and visualization system layer on top of the #SmartME stack, and testing it in a real-world Smart City scenario, by running it on the #SmartME deployment available in the city of Messina. Future works will be devoted to thoroughly assessing scalability, at the level of a city (or even a federation of cities), the experimental comparison with, other trustless solutions in the Smart City domain, both in terms of research and industrial products, and the integration with the Cosmos network for monetization and mechanisms for rewarding data contribution.

Part III

Security Patch Update

Chapter 7

Introduction

In the first part, we discussed protecting the IoT devices against unauthorized access using Blockchain-based and authenticated data structure-based approaches. Next, we discussed protecting the data right from their generation by the IoT devices to their storage. Finally, we discuss securing IoT devices over a period of time. Patches are part of essential preventative maintenance necessary to keep devices up-to-date, stable, and safe from malware and other threats. The vast majority of cyberattacks take advantage of known software and hardware vulnerabilities. The 2015 edition of the Verizon Data Breach Investigations Report (112) revealed 70% of successful cyber attacks exploited known vulnerabilities with available patches. This means that many victims could have prevented a data breach if they had only updated their OSes and apps. In this chapter, we focus on patching the IoT devices to protect them against external attacks over a period of time by using blockchain technologies. We begin by describing the challenges faced by the research community in patching the IoT devices. Then, we present the literature survey of the existing approaches addressing the issues and their shortcomings. Next, we present a primer for the technologies used in the proposed solution. Finally, we conclude the chapter by briefly describing the proposed solution.

7.1 Problem Statement

The last decade has seen tremendous growth in the Internet of Things (IoT) services and devices owing to rapid advancements in networking technologies. Gartner, Inc. predicts a 21% growth to 5.8 billion IoT endpoints by 2020 compared to 2019 (113). The growth is expected to reach 64 billion devices worldwide by 2025 (114), with the predicted market size to reach $520 billion by 2021 (115). With recent advances in next-generation mobile connection technology 5G, mobile subscriptions are predicted to reach 1.3 billion by 2023 (116). The ubiquitous nature of IoT devices makes them a primary suspect for attackers. Recent exploits like DDoS attack (117) or ZigBee chain reaction (118) demonstrated in practice point towards weak security posture of many popular IoT devices. Other attacks include break into homes (119; 120), compromise local networks (121; 122) and smart devices (123). However, with an increase in focus on IoT devices' security (124; 119; 125; 126; 127), the practice of patching the IoT devices with security updates is a simple solution to protect them from cyber-attacks. Despite being a basic solution, patching is often ignored or scarcely performed, as observed by the users and manufacturers alike (128). Narrow profit margins and operational difficulties limit the large-scale patching of IoT devices by the manufacturers. The de-facto client-server based centralized distribution mechanism, specifically for IoT patch updates, is another cause of concern. The volume of IoT devices and the data generated and consumed by them stresses the ISPs, inter-ISP business relationships, and the Internet backbone. Thus, researchers focus on edge computing solutions to limit the information exchange to a single ISP (129; 130). Also, centralized distribution depends heavily on centrally controlled and widely spread cloud service. The centralized control makes the system vulnerable to local outages or natural calamities and exposes it to significant central points of failure (131). Even spreading the cloud servers to multiple regions still makes the system vulnerable to organizational faults and human errors. Considering the limitations of existing systems, a distributed P2P content delivery network holds promise. In particular, they can be explored to optimize patch delivery to IoT devices. For example, consider the file-sharing networks like Gnutella (132), IPFS (133), and BitTorrent (134), which became popular in the last decade. As a case in point, large organizations and enterprises like Microsoft (for Windows 10 updates (135)), Twitter (to speed up servers deployment (136)), Spotify (to reduced its hosting costs (137)), or Amazon (138), are exploring P2P distribution. Unfortunately, such peer-to-peer distribution attempts are limited to a few organizations acting as peers on the Internet. To truly reach its potential, the distribution needs to be more inclusive and, thus, incentivized. Unfortunately, such attempts are severely limited to independent Internet peers.

Such systems also suffer from a fundamental problem: limited availability in case of unpopular files like patch update. For example, in the IoT patch update distribution, the IoT devices are the only parties interested in the update uploaded by a vendor. Thus, other network peers will not be interested in downloading and sharing it in the absence of any incentive. Even the IoT devices will not be able to share the files due to limited resources available. Authors in (139; 140) propose a blockchain-based IoT patch distribution to improve accountability and availability. However, in the absence of incentives, the network did not scale beyond the manufacturers' nodes. Lee *et al.* (141) proposes a cryptocurrency incentive mechanism for encouraging a network of distributor networks to deliver patches to destination IoT devices. Leiba *et al.* (142) propose a similar approach to (141), but with an efficient distribution mechanism. Both proposals enable a fair exchange of authenticated software updates and cryptocurrency payments. However, an on-chain payment solution suffers from several problems. *(i) Costs:* Each transaction on blockchain costs transaction fees in addition to incentives being transferred. For example, the bitcoin transaction fee is reaching around 60 cents.[1] IoTPatchPool (142) analyzed per-device fees to be around 10 cents. *(ii) Latency:* The delay caused due to the required number of block confirmation prevents the solution from scaling. For example, an average block creation time in Bitcoin is ten minutes, and it needs at least six blocks to confirm a transaction. Thus, a single update may take around one hour. *(iii) Throughput:* Due to the latency delay, the device

[1]https://bitinfocharts.com/comparison/bitcoin-transactionfees.html

update is limited by an upper bound within a given time frame. *(iv) Privacy:* Being a public ledger that can be audited by anyone, blockchain lack privacy. An attacker can learn critical information like the number of devices handled by the vendor, how many devices got patched, the cost of patching the devices, etc.

7.2 Related work

In this section, we present the literature from the decentralized storage networks and IoT devices patching domain. The current work spans across these domains.

7.2.1 General Architectures for IoT Patching

The traditional delivery network for patch updates used to be host-centric, where the vendor used to provision the updates for the clients. However, with the advent of the Internet of Things, there is a massive explosion in the number of devices, making this solution challenging to scale. The complex infrastructure requirements and the personnel to manage it increased the software providers' maintenance cost. Liu *et al.* (143) and Zhen-hai and Yong-zhi (144) are some of the studies that outlined the challenges around availability and reliability by outsourcing infrastructure maintenance to IaaS and a Maven-based solution, respectively. However, the proposed solutions were limited by the number of IoT devices. Yu *et al.* (145) tried to address the IoT devices' security via traditional solutions like antivirus and software patches. With the scale and diversity of IoT devices, there was a need for a paradigm shift in security solutions. The authors of (146) propose a solution based on device diversity for a particular scenario of vehicular networks. Similar solutions were proposed in (147; 148) with a non-trustworthy assumption. However, the problem of scalability, which is central to IoT devices, is not addressed. The authors of (139) proposed a blockchain and peer-to-peer file sharing based patch distribution system which is secure, highly available, and allows versioning of updates. A similar idea was proposed in (140) by Boudguiga *et al.* with the addition of a trusted entity responsible for verifying the update. Both the proposals acknowledge the complexity of sharing a patch update on the blockchain and, thus, propose an off-chain solution for distribution. However, in the absence of an incentive mechanism for the distributor nodes to distribute the patch updates, the solutions are limited in scale and marginally better than the centralized vendor network. Lee (141) proposed a blockchain-based solution for patch distribution with the incentive for delivery. A unique receipt is generated for each delivered patch, and only the distributor successfully providing the patch can claim the reward. This is achieved by registering the distributor's identity (encryption keys), which is then used to preserve the integrity of the patch update. However, the vendor's unique property is to distribute the package to every distributor to have fairness in the system. Thus, the system has similar bandwidth requirements as those of a centralized solution. Furthermore, creating blockchain transactions for each patch delivery by a distributor is a costly and time-consuming process. The solution also requires the IoT device to maintain a digital wallet to pay for the update, which introduces additional vulnerabilities and attack vectors. Finally, the wallets need to be provisioned, causing additional transactions on the blockchain. IoTPatchPool (142) presents another blockchain-based patch delivery network with an incentive for every successful delivery. The patch distribution is done via a distributed storage network like BitTorrent, while the blockchain network handles the security guarantees and incentives. The security guarantees are encoded in the smart contract and are stored on the blockchain for public verifiability. The distributors claim the reward from the smart contract by presenting a proof-of-distribution for each successful delivery. Though promising, the solution does not scale well as the contract creation, committing the proof-of-distribution, and revealing the secret for claiming the reward create transactions on the blockchain, creating roughly one hour of latency per transaction. IOTA (149) is another approach supporting micro-transactions, making it promising for patching IoT devices. However, initially designed for service exchange between IoT devices, IOTA's incentive mechanism requires initiation from IoT devices, making it unsuitable for security patching use cases.

7.2.2 Decentralized Storage Networks

Decentralized storage networks (DSN) distribute digital content among the participants via peer-to-peer networking. Also known as P2P file-sharing systems, the underlying architecture resembles blockchain, making it a suitable candidate for a blockchain-based distribution scheme. Incentivization is necessary for P2P, as peers tend to free-ride, affecting the network's performance and availability (150; 151). Thus, incentivization must be a part of any solution based on DSN. BitTorrent's choking algorithm (134) is an incentivized "quid pro quo" exchange scheme in which an uploader can eliminate a non-participating peer. The scheme is promising but is limited in case of unpopular content like patch update. This affects the long term availability of the content (152). Incentivization is crucial in IoT use-case where the 3rd party distributors are expected to share unpopular content necessary to secure the IoT devices. It is reasonable to assume that the IoT devices, being resource constraint, will not share the update among themselves. Given the untrusted and incentivization oriented situation, blockchain seems to be the solution. It allows a trustless exchange of service and cryptocurrency among its users. Siacoin (153), Storj (154), Swarm (155) and Filecoin (156) are some examples of blockchain-based DSN solutions. All the above solutions assume trustless exchange between participating entities and that payment channel supports it.

7.3 Preliminaries

In this section, we briefly describe the technologies enabling the system. We begin by briefly describing the decentralized storage network and the bitcoin lightning network. Next, we present the requirements related to the entities that participate in a generic patch delivery system.

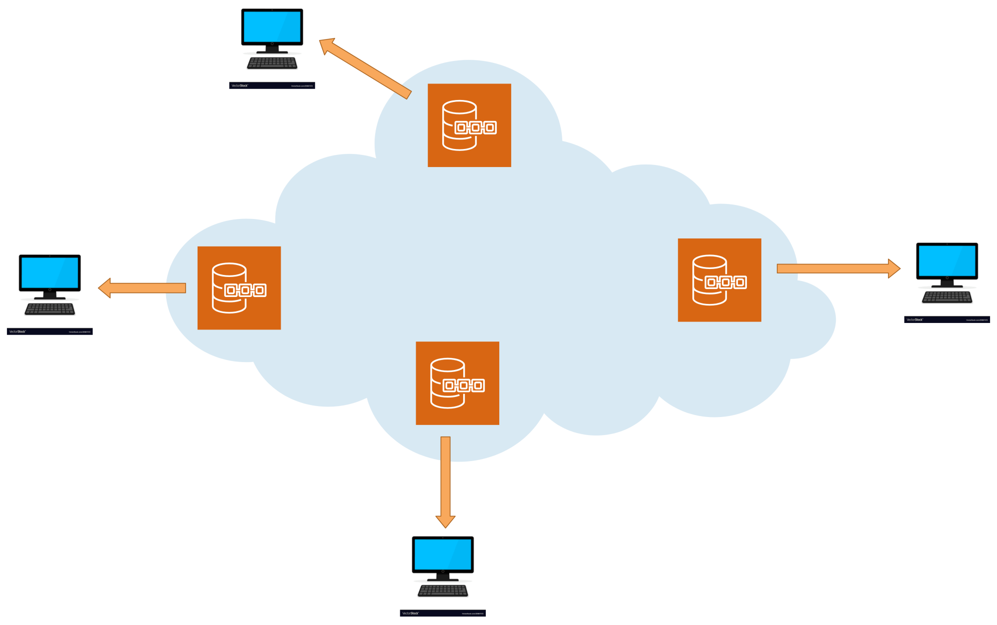


Figure 7.1: Decentralized storage network.

7.3.1 Decentralized Storage Network (DSN)

Decentralized storage (Figure 7.1) means the files are stored on multiple computers (called nodes) on a decentralized network. Like with conventional cloud storage, you can request it and receive the file when you need the file. Requesting your file works similarly to BitTorrent and other P2P clients where you download fragments of that file from participants in the network until you have the full file. However, that does not mean those holding your files can read them. Instead, decentralized storage automatically encrypts files, and only you hold the encryption key, guaranteeing you can only read your files. Furthermore, through a process of sharding, no single person holding your files has the entirety of it, thus adding an extra layer of security and protection. Unlike centralized cloud storage, which keeps data in a central point based on a location that might not be near you (resulting in users competing for bandwidth), the nature of decentralized storage means data distribution and retrieval is handled by nearby peers regardless of physical location. This results in higher transfer speeds due to utilizing local network bandwidth. The peer-to-peer or decentralized storage network should be available to all the participating entities. The peer discovery is based on the distributed hash table (DHT), allowing access to all peers without imposing any substantial requirements. Based on DHT, each peer is aware of all the peers holding a part of the data.

7.3.2 Lightning Network

The Lightning Network (Figure 7.2) scales blockchains and enables trustless instant payments by keeping most transactions off-chain and leveraging the security of the underly-

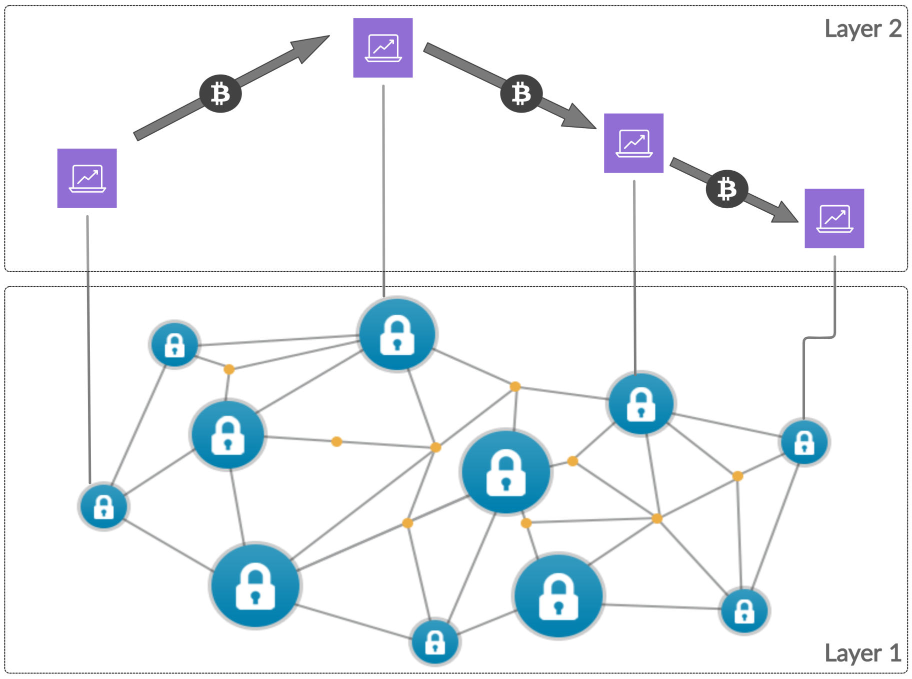


Figure 7.2: Bitcoin lightning network.

ing blockchain as an arbitration layer. This is accomplished primarily through ”payment-channels,” wherein two parties commit funds and pay each other by updating the balance redeemable by either party in the channel. This process is instant and saves users from waiting for block confirmations before they can render goods or services. Payment channels are trustless since any attempt to defraud the current agreed-upon balance in the channel results in the liable party’s complete forfeiture of funds. By moving payments off-chain, the cost of opening and closing channels (in the form of on-chain transaction fees) is amortized over the volume of payments in that channel, enabling micropayments and small-value transactions for which the on-chain transaction fees would otherwise be too expensive to justify. Furthermore, the Lightning Network scales not with the transaction throughput of the underlying blockchain but with modern data processing and latency limits - payments can be made nearly as quickly as packets can be sent. Hash Time-Locked Contracts (HTLCs) allow transactions to be sent between parties who do not have a direct channel by routing them through multiple hops, so anyone connected to the Lightning Network is part of a single, interconnected global financial system. In short, the Lightning Network enables scalable blockchains through a high-volume of instant transactions, not requiring custodial delegation.

7.4 Proposal Summary

As a solution to a scalable patch distribution network, we present P^{4}UIoT, a pay-per-piece patch update for IoT software updates using a gradual release, an incentivized distributed delivery network based on Bitcoin's Lightning Network (8). The proposal reduces the transaction fee, as a single transaction with a commitment is mined on the blockchain. This commitment is known as a channel. Once confirmed, further transactions happen purely peer-to-peer thus, improving the throughput of the system. Also, latency is reduced to network latency. Privacy is preserved thanks to an underlying onion-routing scheme and having just the payments' involved parties aware of a payment made and its terms. The gradual release concept refers to dividing and exchanging the commodity between the sender and receiver in rounds wherein the trust between participants grows with each round. We formally analyze the proposed approach using TLA+ (9) formal modeling language and evaluate the formal specification's correctness. Also, we implement a prototype of the proposed framework on the Bitcoin's lightning network to evaluate its performance. We perform experiments to evaluate the distribution latency and cost metrics in various test scenarios. Finally, we present the proposed framework's advantage by comparing it with existing work in the literature.

Chapter 8

Securing IoT Devices using Patch Update

In this chapter, we describe a solution to a scalable patch distribution network. We present P^{4}UIoT (17), a pay-per-piece patch update for IoT software updates using a gradual release, an incentivized distributed delivery network based on Bitcoin's Lightning Network (8). The proposal reduces the transaction fee, as a single transaction with a commitment is mined on the blockchain. This commitment is known as a channel. Once confirmed, further transactions happen purely peer-to-peer thus, improving the throughput of the system. Also, latency is reduced to network latency. Privacy is preserved thanks to an underlying onion-routing scheme and having just the payments' involved parties aware of a payment made and its terms. The gradual release concept refers to dividing and exchanging the commodity between the sender and receiver in rounds wherein the trust between participants grows with each round. We formally analyze the proposed approach using TLA+ (9) formal modeling language and evaluate the formal specification's correctness. Also, we implement a prototype of the proposed framework on the Bitcoin's lightning network to evaluate its performance. We perform experiments to evaluate the distribution latency and cost metrics in various test scenarios. Finally, we present the proposed framework's advantage by comparing it with existing work in the literature.

8.1 Introduction

In this section, we describe P^{4}UIoT: pay-per-piece patch update framework, its building blocks, and the design of the pay-per-piece digital exchange protocol, which enables it to achieve an open incentivized system. Pay-per-piece is a general exchange protocol that includes two or more parties that are interacting in a trust-less exchange of goods. The core concept is the fact that each party will reduce its risk exposure by dividing the patch by a particular factor. The trust is built over time with multiple cycles of exchange of the patch piece and incentive. The proposal will try to address a few of the shortcomings of the former frameworks, which are as follows:

1. **Patch size limitations**: the IoTPatchPool (142) framework relies on the ZKCP protocol, which is limited in its current implementation for 55 kB.

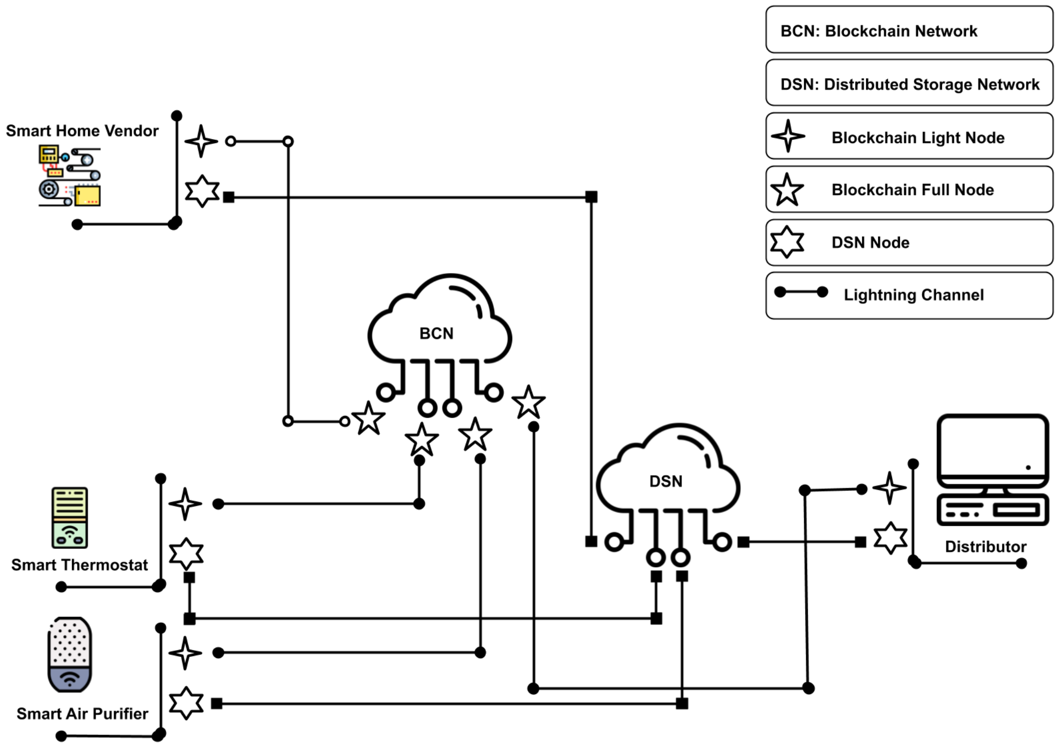


Figure 8.1: P^{4}UIoT High level architecture overview.[1]

2. **Network overhead**: in addition to the patch update itself, the former framework requires the sending of additional data in the form of ZKsnarks proving and verification keys; in some cases, it is a hundred folds bigger than the patch file itself.

3. **Demands for IoT devices**: in the ZCKP protocol, the IoT devices need to verify the ZKsnarks statement, and also, there is a need to decrypt the patch binaries once the decryption key is revealed. These two operations are having additional demands on the device storage memory and processing power.

4. **Costs**: the former network is a first layer solution on the blockchain network. To reduce costs, new methods of offering payment to the distributors should be used.

The proposed framework consists of two networks, a lightning network over a blockchain network used to enable a transparent exchange of patch update and cryptocurrency incentive in the form of binding bid, and a decentralized storage network (DSN) which allows a peer-to-peer sharing of a patch update. The participating entities of the network, namely Vendors, Distributors, and IoT devices, use the two networks to follow the protocol proposed in the framework. Figure 8.1 represents an overall architecture of the proposed solution, focused for the sake of clarity on a smart home use case. The delivery life-cycle is initiated by a vendor that needs to distribute a new patch update to the deployed IoT nodes. The

[1]Icons made by https://www.flaticon.com/authors/freepik

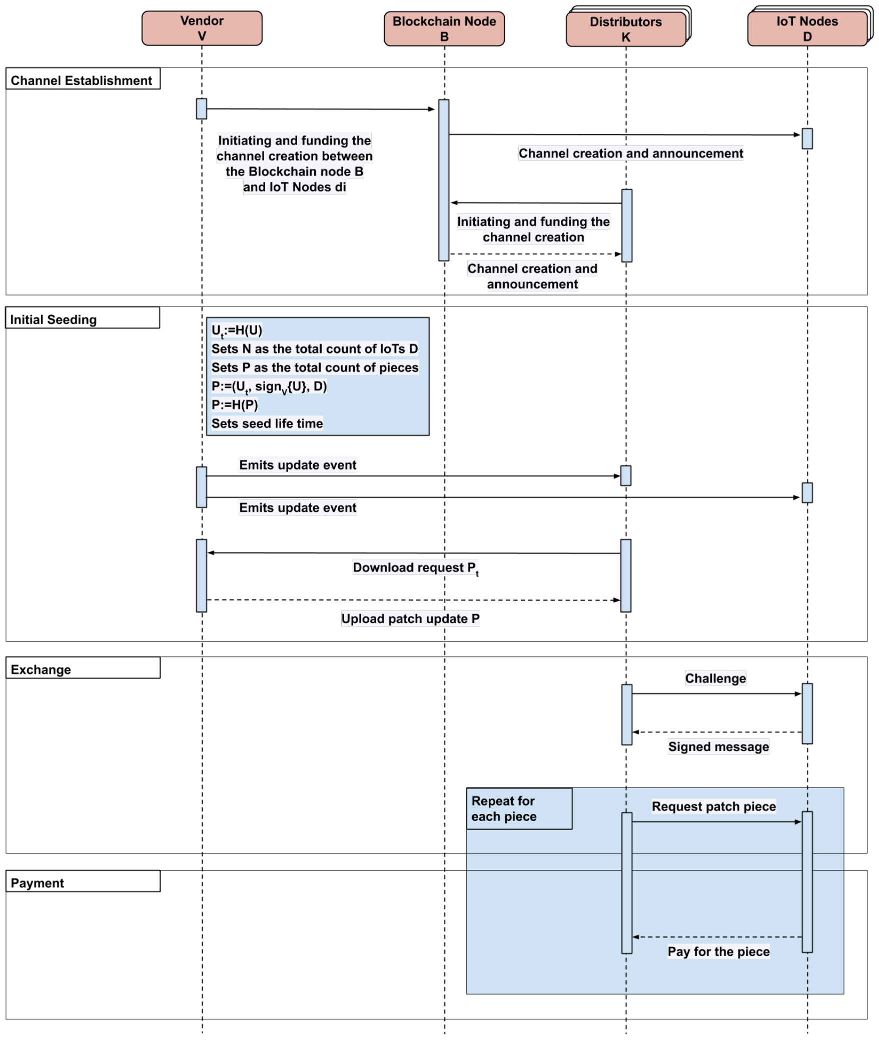


Figure 8.2: P^4UIoT sequence diagram.

vendor invites the distributors interested in the task of providing the update to the IoT nodes by committing a bid on the blockchain, which will be gradually released with the proof-of-distribution. The IoT nodes will release the proof-of-distribution to the distributor upon receiving a piece of patch update, who can then exchange it for the cryptocurrency incentive. Also, for a limited duration, the vendor acts as a source of distribution of patch update for the distributors and IoT nodes alike. Once done, the distributors take over the process, seeding the pieces downloaded from the vendor via the DSN.The IoT nodes, upon notification of a patch release, download the pieces of update from distributors via the DSN and release the corresponding proof-of-distribution.

Leading from the previous work on the IoTPatchPool (142), the proposal adds additional methods to achieve a trustless protocol for IoT patch distribution. The desired framework should accompany all the former framework properties and also address a few of the shortcomings listed earlier. The pay-per-piece protocol requires an exchange for signature between

the distributor and the IoT during the file exchange. As a security parameter for the system, the smaller pieces and hence larger count of pieces is preferable since it minimizes the initial risk in the fair exchange. On the other hand, a considerable number of signatures means higher fees paid in the form of the transaction cost. One method to achieve a lower number of signatures sent to the smart contract is signature aggregation. The ability to compact multiple signatures into one signature will lead to smaller fees paid to the miners.
The following are the two approaches integrated into the bitcoin protocol to reduce the signature size.

1. Schnorr Signatures were introduced to the bitcoin network in (157). Given n signatures on n distinct messages from n distinct users, it is possible to aggregate all these signatures into a single short signature. The tests conducted show a significant speedup. The ratio between the time it takes to verify n signatures individually and to verify a batch of n signatures goes up logarithmically with the number of signatures or, in other words: the total time to verify n signatures grows with $O(n/log(n))$.

2. ECDSA Elliptic Curve Digital Signature Algorithm (158) is one of the main building blocks of the Bitcoin network, and it is used for verifying all the transactions. Reviewing all the possibilities, the ECDSA alternative was chosen for its low costs and lack of demand for preliminary online setup between signers. It's important to notice that choosing ECDSA will allow this framework to be scalable as the blockchain ledger itself is making additional improvements in scalability to be applied to this framework.

The Schnorr Signatures and ECDSA together can improve the verification time with signature aggregation and key size reduction (159). This is particularly advantageous in the case of resource-constrained devices.

8.2 Protocol

In this section we outline the proposed protocol for sending patch update to a set of IoT devices $d_1, ..., d_N$ manufactured by a vendor V. Let us denote the patch update by U. The proposal consists of five stages: initial setup, bid commitment, initiating patch update seeding, exchanging patch piece for a pay-per-piece confirmation, and a reward claim. Detailed sequence diagram presented in Figure 8.2.

Initial Setup
The proposed framework uses connection establishment as an initial step for any patch update procedure. The connection links the vendor to the community of distributors who would like to participate in future IoT patch update procedure. The distributor would listen to an event emitted by the vendor to receive the newly published patch and initiate the process of a patch update.

Bid Commitment
The vendor V sends a transaction to the blockchain committing the incentive I_i for each of

the IoT devices d_i. Once committed on the blockchain, a peer-to-peer payment channel is established between each distributor k_i and each of the IoT nodes d_i.

Initiating Patch Update Seeding
The vendor V performs the following steps on receiving an update U:

1. Computes the hash of patch file $U_t := H(U)$
2. Sets N as the total count of IoTs d_i
3. Sets M as the total count of pieces
4. Prepares the payload as: $P := (U, sign_{v_i}\{U_t\}, d_i)$, where d_i is represented as an ordered list of the IoT devices public keys
5. Computes the hash of the payload $P_t := H(P)$
6. Sets $\Delta_{LifeTime}$ as sufficient time for vendor V to seed the patch
7. Emits an event announcing the availability of patch update.

Both the distributor nodes k_i's and IoT devices d_i's are listening to the events emitted by the vendor V known by his public key pk_i. Once the event is received, the following set of operations are performed:

1. Corresponding to the hash P_t, the distributor requests vendor V for the payload via the DSN network.
2. The vendor V begins the seeding of the payload P using the DSN network. The vendor seeds the payload P for a reasonable time within which some distributor k_i can receive the update.
3. After completely downloading the package P, k_i verifies that:
   - the patch is published by the vendor.
   - the integrity of the patch is maintained throughout the communication.
4. k_i then announces via DHT to its peers as a possible source of the patch update.

Exchange
In this step, the IoT device, and distributor communicate to exchange the patch update and proof-of-distribution. Initially, a verification of the IoT device is conducted to understand if it is a member of the vendor-approved list d_i. The detailed exchange is listed below:

1. The patch update available event is received by the IoT device d_i, and the device verifies the sender of the event.

2. The device d_i searches for distributor k_i possessing of the patch update via the DHT peer discovery. Once found, the device sends the download request to the distributor k_i.
3. k_i sends a random nonce c, a challenge, to d_i to sign on.
4. d_i:
   - Computes $sign^{d_i}\{c\} := Sign(sk^{d_i}, c)$.
   - Sends the tuple $(pk^{d_i}, sign^{d_i}\{c\})$ to k.
5. In this stage, the IoT device can start downloading pieces from any free providing distributors (i.e., the vendor node).
6. d_i:
   - Verifies that $pk^{d_i} \in D_i$
   - Verifies that $VerifySig(pk^{d_i}, sign^{d_i}\{c\}, c) = 1$
7. Piece by piece exchange:
   - d_i sends the requested j part of the patch file
   - k_i concats M_j=$concact(U_t, pk^{d_i}, j)$ and sends to d_i
   - k_i sends part to d_i
   - k_i chokes further transfer till the payment for the part is received.

Reward Claiming
The reward is claimed by the distributor of the piece sent as follows:

1. d_i:
   - once the piece is received, d_i sends the payment of the piece to k_i.
   - once k_i received the payment, it sends the next piece and waits for payment.

8.3 Implementation

The P^{4}UIoT implementation proves the capability of such a framework to function in the multi-party environment. Figure 8.3 refers to the technology stack used for the implementation. Raspberry pies are used as a fleet of IoT devices that request patch from the distributors, which in return communicate with the lightning network to reclaim their reward. The implementation of the proposal needs an evaluation of the new features of the framework. One key question is *the latency of the blockchain.* Since the proposal is based on the lightning network, once the vendor sends the commitment to the blockchain, the payment is processed in real-time between the distributor and IoT device. Another key question is *the costs.* The commitment is the only transaction sent by the vendor on the blockchain. Assuming multiple usages, the transaction cost is negligible. To answer these questions, the implementation is based on the lightning network and deployed over the Bitcoin's regtest. The bitcoin and lightning network protocols are used as is, and no changes has been made to the original protocols. The transaction prices are aligned with the mainnet, and the block-time is similar to using PoW as the consensus algorithm. Finally, all attacker scenarios were tested, among reward interception, malformed channel, Denial-of-service (DoS) redeem with false message, exhausting resources, blockchain-related attacks, compromising a firmware's integrity, software downgrade attack, greedy distributor, and greedy IoT node.

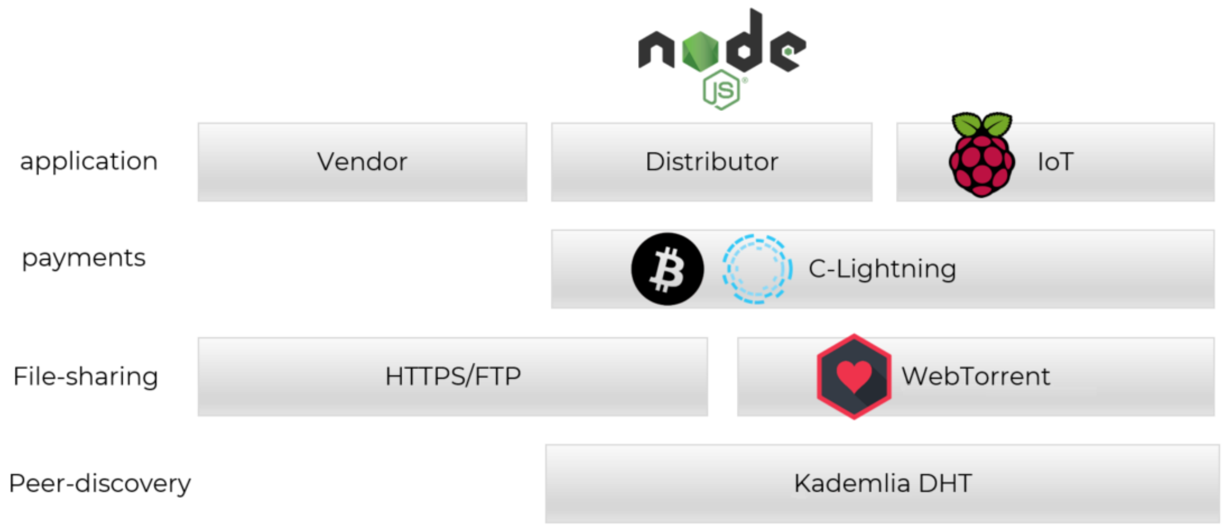


Figure 8.3: The technology stack used in P^{4}UIoT implementation.

8.4 Security Model

In this section, we present a threat model and the assumptions related to the participants. Based on the model, we outline the security goals of the system design.

8.4.1 Threat Model

The proposed architecture consists of three kinds of participants: a *vendor* who wants to distribute the update, a *distributor* who shares a part or whole of an update in exchange for an incentive, and an *IoT device* that needs and pays for the patch. We assume that the vendor, the distributor, and the IoT device can be uniquely identified and can securely communicate with each other via standard protocols and credentials like https. The IoT device is identified by its public key, which the vendor pushes into the device during the manufacturing process. A more robust mechanism for device identity like SSI (160) can be used to secure the system further and is beyond the scope of this work. Also, we assume that the distributor/s and the IoT devices are aware of the public key of the vendor/s, and the identity of IoT devices is available with the vendor. These assumptions can be achieved during the manufacturing of an IoT device or configured by an admin. We also assume that the vendor can be trusted to work in favor of the network and does not accept the hash of a malicious file, and IoT devices trust the patch update sent by the vendor.

The IoT devices and the distributors can assume malicious intent towards each other and can try to claim updates/incentives.

8.4.2 Adversarial Model

We assume that an adversary has full control over the communication medium and can alter any communication with the blockchain or any participating entity (161). We also assume that the adversary can take a passive as well as an active role and perform actions varying from listening on the network packets to injecting, replaying, or filtering any message.

8.4.3 Security Goals

Based on the adversarial and trust profile, we outline the security goals that our proposal should satisfy: *(i)* an IoT device pays for a patch before receiving it; at the same time, a distributor delivering a patch is paid. *(ii)* the IoT device can verify the integrity and the origin of the patch in the presence of malicious actors. *(iii)* within a given time frame allocated by a vendor, an IoT device will be able to patch itself.

8.5 Formal Specification

In this section, we present a formal specification of the proposed approach using TLA+ abstract language (9) for model verification. Using TLA+ definitions, we define the system interaction of P^{4}UIoT and verify the correct operation of the system. Finally, using the formal specification, we check for the correctness of the proposed solution using TLC model checking tool in the next section.

8.5.1 State Transitions

There are three participants in the proposed approach, the vendor, the distributor, and the IoT. The payment (incentive) system is based on the lightning network running on top of bitcoin network. The bitcoin network and the vendor is an offline entity, while the rest of the entities are online. The state transition diagram in Figure 8.4 presents the system behaviour. The vendor generates the package to be distributed, as well as establishes a payment channel with the IoT devices. The package preparation and channel establishment are done offline. The distributors and the IoT nodes have their own states to process the patch update. The communication and payment between the distributors and the IoT nodes happen through channels.

The TLA+ action formula is used to represent individual states of the distributors and IoT nodes, and their transitions. We represent the unchanged variables as primed variables (indicated by the UNCHANGED keyword in the action formula). Each step consists of the following: *(i)* the entry condition, and *(ii)* the variable changes. Detailed specifications on TLA+ can be found in (9; 162).

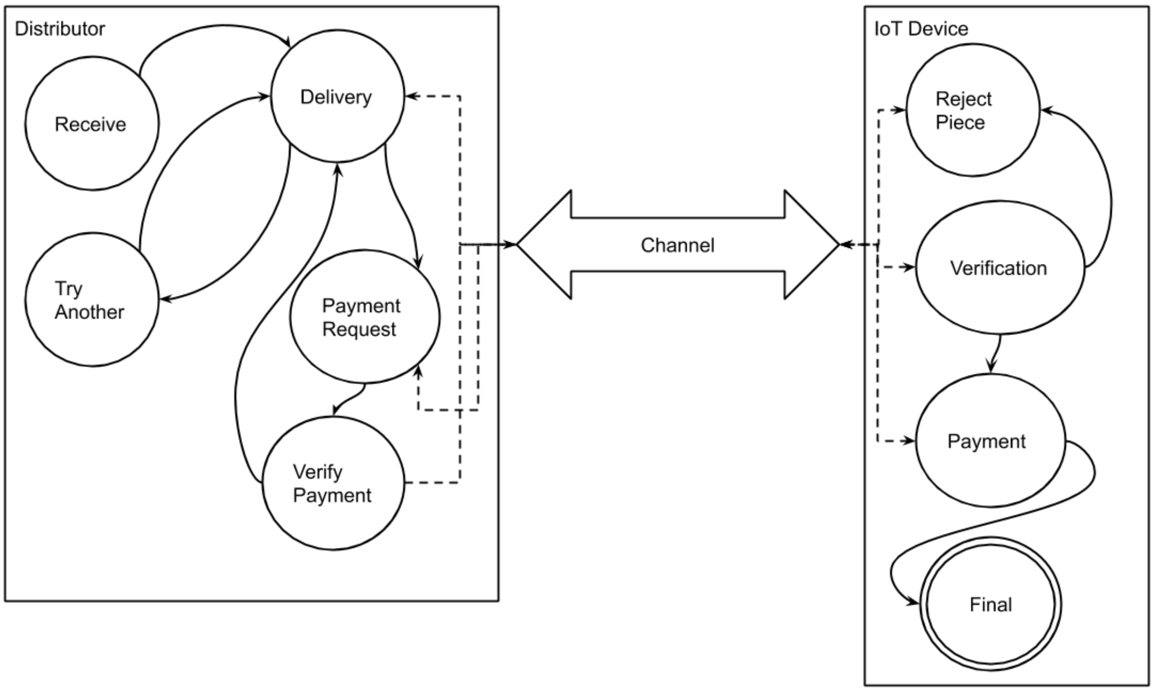


Figure 8.4: P^{4}UIoT state transition diagram.

We begin by initializing each variable with a default value. The variables InitVendor, InitDistributor, and InitIoT are initial values of the variables for Vendor, Distributors, and IoTs, respectively. The channel and payment_channel are used for message and payment exchange between distributors and IoTs. Init is defined as:

$$\begin{aligned}
&Init \triangleq \\
&\quad \land\ InitVendor \\
&\quad \land\ InitDistributor \\
&\quad \land\ InitIoT \\
&\quad \land\ balance = [c \in Distributor \cup IoT \mapsto InitBalance] \\
&\quad \land\ channel = \{\}
\end{aligned}$$

The state change for the proposed model is defined as $Init \land *[\ Next]\ _{vars} \land WF\_vars(Next)$ in TLA+ specifications where the WF_vars is for fairness. The possible transitions from the initial state are described in the following equation:

$$Next \triangleq$$

```
        $\lor$ $Seeding$
        $\lor$ $\exists\, k \in Distributors$ :
            $\lor$ $Receive(k) \lor Delivery(k) \lor PaymentRequest(k)$
            $\lor$ $VerifyPayment(k) \lor TryAnother(k)$
        $\lor$ $\exists\, d \in IoT$ :
            $\lor$ $Verification(d) \lor Payment(d) \lor RejectPiece(d)$
        $\lor$ (* Disjunct to prevent deadlock on termination *)
            $((\forall\, q \in IoT : dState[q] = I\_Final)$
                $\land\, Terminating)$
$Spec \triangleq$
        $Init \land *[\; Next]_{\,vars} \land WF\_vars(Next)$
```

We consider the case of patch delivery by Vendor to multiple IoT nodes with the help of multiple Distributors. In the initial step, the vendor is responsible for preparing the package to be distributed and meta-data related to the distribution, such as the distributors and IoT nodes involved in the process (identified by their public keys), mechanism to ensure integrity of the package, etc. The blockchain is not actively involved in the system interactions as the scope is limited to creation of lightning channels for the payment. The channel creation is considered as part of pre-process and is thus not part of the state transition. Termination describes the conditions in which P^{4}UIoT is considered to be stable i.e., the patch has been distributed to the IoT devices successfully and the Distributors have been paid.

8.5.2 Vendors

The Vendor is responsible for preparation of the package to be distributed to the IoT devices. We define the Seeding as a combination of TLA+ actions. For the sake of simplicity, we use external oracles to define core cryptographic primitives and provide these values externally during the verification process. As per the proposal, the keys of the communicating entities are registered with each other. For example, the Vendor's public key is known to the Distributors and the IoT devices. EPAYLOAD is the data structure used to provide payload. regKey holds the public keys of the target nodes which act as their identities on the network. torrentinfohash and filehash are used to verify the integrity of the torrent and file content. Finally, the patch update availability is posted publicly to all entities and the patch update is seeded by the torrent network.

```
$Seeding \triangleq$
        $\exists\, pu \in regKey$ :
        $LET$
            $payload \triangleq EPAYLOAD[1]$
            $torrentinfohash \triangleq EPAYLOAD[2]$
            $filehash \triangleq EPAYLOAD[3]$
```

$$
\begin{aligned}
&\quad signature \triangleq EPAYLOAD[4] \\
&\quad target\_node \triangleq NODE\_TYPE \\
&IN \\
&\quad \wedge regKey' = regKey \setminus pu \\
&\quad \wedge Post(\langle pu, payload, target\_node, torrentinfohash, \\
&\quad\ \ filehash, signature\rangle) \\
&\quad \wedge pTime' = pTime + 1 \\
&\quad \wedge UNCHANGED\langle distributorVars, iotVars, commVars\rangle
\end{aligned}
$$

8.5.3 Distributors

A Distributor transits through different states based on the enabling conditions. The Distributor begins with a $D\_Waiting$ and then transits through $D\_Delivering$, $D\_VerificationWaiting$, $D\_ReceiptWaiting$, and D_Final. The actions associated with a Distributor on the above defined states.

Receive: Initially, a Distributor is waiting for a patch update from the Vendor. The Distributor is said to be in $D\_Waiting$ state. Once the Distributor receives the patch, it performs verification on the integrity of the received file and existence of duplicate package. Once the integrity and uniqueness is verified, the package is stored in a buffer and the Distributor makes a transition to $D\_Delivering$.

$$
\begin{aligned}
&Receive(k) \triangleq \\
&\qquad \wedge kState[k] = D\_Waiting \\
&\qquad \wedge Len(patchPool) > 0 \\
&\qquad \wedge LET \\
&\qquad\quad patch == Head(patchPool) \\
&\qquad\ \ IN \\
&\qquad\quad VerifyPatch(patch[2], patch[4], patch[5], patch[6]) \\
&\qquad \wedge kBuffer' = [kBuffer\ \ EXCEPT\ \ ![k] = Head(patchPool)] \\
&\qquad \wedge kState' = [kState\ \ EXCEPT\ \ ![k] = D\_Delivering] \\
&\qquad \wedge patchPool' = Tail(patchPool) \\
&\qquad \wedge UNCHANGED\ \langle regKey, pTime, kIndex, kReceipt, \\
&\qquad\ \ iotVars, commVars\rangle
\end{aligned}
$$

The first few lines are entry level conditions namely the current state is $D\_Waiting$; the patchPool contains a package to be distributed, and the package integrity is verified. Once satisfied, the package is stored in buffer and the state is changed to $D\_Delivering$. The variables related to IOT and lightning channel remain unchanged.

Delivery: The Distributor in $D\_Delivery$ state uses the IoT device identities to find the target devices and delivers the patch to the IoT devices piece-wise. The Distributor uses the

Send message to send a piece of the package with source, piece index, price of a piece, and lightning receipt. After the transmission, the Distributor chokes the next piece and waits for the receipt of payment for the piece.

```
$Delivery(k) \triangleq$
        $\wedge\ kState[k] = D\_Delivering$
        $\wedge\ \exists\ d \in IoT :$
            $\wedge\ dState[d] = I\_Waiting$
            $\wedge\ dIndex[d] = kIndex[k]$
            $\wedge\ LET$
                $m \triangleq kBuffer[k]$
                $pieceindex \triangleq kIndex[k]$
            $IN$
                $Send([src \mapsto k, dst \mapsto d, data \mapsto \langle m[2][pieceindex] \rangle])$
        $\wedge\ kState' = [kState\ EXCEPT\ ![k] = D\_VerificationWaiting]$
        $\wedge\ UNCHANGED\ \langle vendorVars, kBuffer, kReceipt,$
          $kIndex, iotVars, balance \rangle$
```

The Distributor transits to $D\_VerificationWaiting$ once the piece id delivered to the IoT device. It then waits for the response from the IoT device.

PaymentRequest: Once the Distributor receives a successful verification message from the IoT device (M_OK), the Distributor transits to Payment_Waiting state. The Distributor waits for the IoT device to make payment for the piece sent.

```
$PaymentRequest(k) \triangleq$
        $\exists\ c \in channel :$
            $\wedge\ c.dst = k$
            $\wedge\ c.type = M\_OK$
            $\wedge\ kState' = [kState\ EXCEPT\ ![k] =$
              $D\_PaymentWaiting]$
            $\wedge\ LET$
                $index \triangleq kIndex[k]$
                $price \triangleq PIECEPRICE$
                $receipt \triangleq kReceipt[k]$
            $IN$
                $RecvInadditionSend(c, [src \mapsto k, dst \mapsto c.src,$
                  $type \mapsto M\_PaymentRequest,$
                  $data \mapsto \langle k, index, price, receipt \rangle])$
            $\wedge\ UNCHANGED\ \langle kBuffer, kIndex, kReceipt,$
```

$vendorVars, iotVars, balance\rangle$

TryAnother: In case the received piece fails the verification at the IoT node, the node returns a M_NO message to the Distributor. In this phase, the Distributor goes back to delivering another patch piece.

```
$TryAnother(k) \triangleq$
        $\exists\, c \in channel :$
            $\land\, c.dst = k$
            $\land\, c.type = M\_NO$
            $\land\, (\lor\, kState[k] = D\_VerificationWaiting$
              $\lor\, kState[k] = D\_PaymentWaiting)$
            $\land\, kState' = [kState\ EXCEPT\ ![k] = D\_Delivering]$
            $\land\, channel' = channel \setminus \{c\}$
            $\land\, UNCHANGED\ \langle kBuffer, vendorVars, iotVars\rangle$
```

VerifyPayment: On successful verification of the piece, the IoT device pays for the piece via lightning channel. The Distributor verifies the status of the payment and on successful verification, sends the next piece. If the Distributor has successfully delivered all the pieces, the Distributor transits to D_Delivering state to service the next IoT device.

```
$VerifyPayment(k) \triangleq$
        $\exists\, c \in channel :$
            $\land\, c.dst = k$
            $\land\, c.type = M\_Paid$
            $\land\, balance' = [balance\ EXCEPT\ ![c.dst] =$
                $balance[c.dst] + Fee,$
                $![c.src] = balance[c.src] - Fee]$
            $\land\, kState[k] = D\_PaymentWaiting$
            $\land\, c.data[1] = STATUS$
            $\land\, channel' = channel \setminus c$
            $\land\, IF\ kIndex[k] = NUMPIECES$
                $THEN$
                 $\land\, kIndex' = [kIndex\ EXCEPT\ ![k] = 1]$
                $ELSE$
                 $\land\, kIndex' = [kIndex\ EXCEPT\ ![k] = kIndex[k] + 1]$
            $\land\, kState' = [kState\ EXCEPT\ ![k] = D\_Delivering]$
            $\land\, UNCHANGED\langle kBuffer, kReceipt,$
             $vendorVars, iotVars\rangle$
```

8.5.4 IoT Devices

An IoT node has transition states namely I_Waiting, I_Verifying, I_RequestWaiting, I_Paying, and I_Final. I_Waiting is the initial state and the IoT node waits for the patch piece from the Distributor.

Verification: The Verification process in IoT device is initiated by a Distributor. When the Distributor sends a piece to the IoT device, the IoT device validates the integrity of the piece. We defined the function VerifyMeta to validate the received parameters and return TRUE if the validation succeeds. We will be providing the validation values externally to simulate the cryptographic primitives. In addition, the IoT device validates the identity of the Distributor. In case of successful validation, the IoT device sends M_OK to the Distributor and transits to I_RequestWaiting state. In case the validation fails, the IoT device sends M_NO back to the Distributor and remains in the present state.

```
$Verification(d) \triangleq$
                $\exists\, c \in channel :$
                    $\wedge\, c.dst = d$
                    $\wedge\, c.type = M\_Package$
                    $\wedge\, dState[d] = I\_Waiting$
                    $\wedge\, LET$
                        $recvPayload \triangleq c.data[1]$
                    $IN$
                        $IF\ VerifyPiece(recvPayload, dBuffer[d])$
                        $THEN$
                            $\wedge\, dState' = [dStateEXCEPT\ ![d] =$
                              $I\_RequestWaiting]$
                            $\wedge\, dBuffer' = [dBuffer\ EXCEPT\ ![d] =$
                              $dBuffer[d]c.data]$
                            $\wedge\, RecvInadditionSend(c, [src \mapsto d, dst \mapsto c.src,$
                              $type \mapsto M\_OK, data \mapsto \langle\rangle])$
                            $\wedge\, UNCHANGED\ \langle vendorVars, distributorVars,$
                              $dIndex, balance\rangle$
                        $ELSE$
                            $\wedge\, RecvInadditionSend(c, [src \mapsto d, dst \mapsto c.src,$
                              $type \mapsto M\_NO, data \mapsto \langle\rangle])$
                            $\wedge\, UNCHANGED\ \langle vendorVars, distributorVars,$
                              $iotVars, dIndex, balance\rangle$
```

RejectPiece: The IoT device rejects the piece already available with it and sends a M_NO

message back to the Distributor.

```
$RejectPiece(d) \triangleq$
        $\exists\, c \in channel :$
            $\land\ c.dst = d$
            $\land\ c.type = M\_Piece$
            $\land\ dState[d] \# I\_Waiting$
            $\land\ RecvInadditionSend(c, [src \mapsto d, dst \mapsto c.src,$
                $type \mapsto M\_NO, data \mapsto \langle\rangle])$
            $\land\ UNCHANGED\ \langle vendorVars, distributorVars,$
              $iotVars, balance\rangle$
```

Payment: In this phase, the IoT device pays for the piece the Distributor sent to it. The lightning receipt is sent with the piece. The IoT devices decodes the receipt and sends the payment via the payment channel. The state of the IoT device depends on the state of the patch. If the IoT device has received the complete patch, it transits to I_Final state else it transits to I_Waiting.

```
$Payment(d) \triangleq$
        $\exists\, c \in channel :$
            $\land\ c.dst = d$
            $\land\ c.type = M\_PaymentRequest$
            $\land\ dState[d] = I\_RequestWaiting$
            $\land\ IF\ dIndex[d] = NUMPIECES$
                $THEN$
                 $\land\ dState' = [dState\ EXCEPT\ ![d] = I\_Final]$
                $ELSE$
                 $\land\ dState' = [dState\ EXCEPT\ ![d] = I\_Waiting]$
            $\land\ IF\ c.data[2] = dIndex[d]$
                $THEN$
                 $\land\ dIndex' = [dIndexEXCEPT![d] = dIndex[d] + 1]$
                 $\land\ RecvInadditionSend(c, [src \mapsto d, dst \mapsto c.src,$
                  $type \mapsto M\_Paid, data \mapsto \langle STATUS\rangle])$
                 $\land\ UNCHANGED\langle dBuffer, vendorVars,$
                  $distributorVars, balance\rangle$
                $ELSE$
                 $\land\ RecvInadditionSend(c, [src \mapsto d, dst \mapsto c.src,$
                  $type \mapsto M\_NO, data \mapsto \langle\rangle])$
                 $\land\ UNCHANGED\langle dBuffer, dIndex, vendorVars,$
```

$$distributorVars, balance\rangle$$

8.5.5 Terminating

The system terminates once all the pieces of the patch are delivered to the IoT devices and the Distributors are paid for the piece they delivered. The conditions like FaithfulDelivery, AuthenticatedOrigin, and PackageUniqueness formally evaluate the completion of the patch delivery and are presented in detail in Section 8.6.

$$\begin{aligned} Terminating \triangleq &\\ &\land SuccessfulDelivery\\ &\land PatchIntegrity\\ &\land UNCHANGED\ vars \end{aligned}$$

8.6 Evaluation

In this section we evaluate the correctness of the proposed model based on the formal specifications presented in Section 8.5. Also, based on the security framework, assumptions presented in Section 8.4, and the formal specification we evaluate the proposed framework against the threats and its conformance to the stated security goals.

8.6.1 Model Checking with Correctness Properties

The TLC model checking tool spans the state space to find critical events, e.g., violation of always true conditions called invariant, and deadlocks. The formal specification presented in Section 8.5 is evaluated by the tool to expose such conditions. The tool evaluates the deadlock conditions when the system terminates and checks the invariant on each state. *Successful Delivery and Patch Integrity:* The *Successful Delivery* verifies that the patch pieces delivered by the Distributor are correct and received by the IoT devices. It can be verified by checking the state of the IoT device. As per the specification, the IoT device successfully receiving the patch ends up in I_Final state. Thus the following specification checks for successful delivery.

$$\begin{aligned} SuccessfulDelivery \triangleq &\\ &\lor Cardinality(\{d \in IoT : dState[d] =\\ &\quad I\_Final\}) = Cardinality(NODE\_TYPE) \end{aligned}$$

$$PatchIntegrity \triangleq$$

$$\forall\, q \in \{d \in IoT : dState[d] = I\_Final\} : \\ dBuffer[q] = EPAYLOAD[1]$$

In patch integrity, we validate the content delivered by the Distributor. For all the IoT devices in the final state, we check if the patch delivered by the Distributor is the same as the patch available with the Vendor. Both these conditions are tested on termination.

Incentive Fairness: In order to evaluate the fairness of the distribution of incentive among the Distributors, we check the following conditions.

$$TotalBalanceInvariance \triangleq \\ Sum(balance,\ DOMAIN\ balance) = \\ InitBalance\ *\ Cardinality(Distributor \cup IoT)$$

$$NoUnpaidFee \triangleq \\ \forall\, q \in \{d \in IoT : dState[d] = I\_Final\} : \\ \land\ balance[q] = InitBalance - \\ NUMPIECES\ *\ PIECEPRICE$$

The TotalBalanceInvariance and NoUnpaidFee are the invariant which is true for all the states satisfying the condition. The first condition ensures that the balance of the system is consistent. If the IoT device has paid for a piece of a patch, the balance of the delivering Distributor increases by price of the piece. The second condition ensures that the Distributor who has delivered a piece of the patch, is paid for the delivery.

Model Simulation and Validation: We performed simulation on TLA+ model verification tool to check the correctness of the proposed system and the critical conditions. We ran the experiment on a Windows 10 based Intel i7 quad-core machine with 32 GB RAM. Table 8.1 shows the model checking results with the properties for three distributors and three iot devices with four piece patch update. The simulation process took 240s and visited 1,174,249 distinct states. The simulation completed without encountering any error except for violation of NoUnpaidFee. We discuss this attack in the next section.

Table 8.1: Model checking results with three distributors and three iot devices with four pieces of patch file.

Time	Depth	States Found	Distinct States	Errors
780’	71	17,315,287	3,302,038	0

8.6.2 Threat Analysis

Reward interception: Since an attacker has complete control over the communication channel, he/she can spoof the redeem transaction, establish a channel with the distributor, and

try to withdraw the reward to himself. To prevent this, the invoice generated for the payment contains the public key of the receiver (distributor). The IoT device verifies the public key before making the payment.
Malformed Channel: An attacker can try to change the channel established between the distributor and the IoT devices, thereby, disrupting the exchange of the patch updates and incentives. The success of the attack depends on compromising the Bitcoin network as the channel becomes public once the funding transaction is confirmed on the blockchain. Thus, unless the Bitcoin network is compromised, the channel attributes cannot be altered.
Denial-of-service (DoS) Exhausting Resources: An attacker, acting as an honest distributor or as an honest IoT node, can try to waste resources of the participating entity. However, the proposed framework doesn't require high demanding tasks to be performed. The only attack it can be subjected to is to hash a message and produce a signature, which is not difficult to compute.
Denial-of-service (DoS) Blockchain-related attacks: Another way to prevent patching from being performed is to censor transactions that reach the blockchain network. Aside from the eclipse attack, which has been explained above, such an attack is typically referred to as a censorship attack. Again, such type of attacks requires the Bitcoin network to be compromised, which is difficult by design.
Compromising a firmware's integrity: A malicious actor acting as a vendor may try to announce a malicious software update to the IoT devices. In the proposed framework, it is a costly attack as the malicious vendor has to commit a bid per IoT node on the blockchain.
Software downgrade attack: An alternative to the above attack, a malicious actor may try to push a compromised version of the patch update. Again, in the proposed framework, it is a costly attack as the malicious vendor has to commit a bid per IoT node on the blockchain.
Greedy Distributor: A distributor, acting greedily, may request payment for pieces not sent to the IoT devices. The proposed framework enables the IoT device to verify the availability of the piece. The IoT device can reject the payment request for the pieces not received by it.
Greedy IoT Node: An IoT node, acting greedily, can request distinct pieces from separate distributors without releasing the payment. The attack can succeed in case the number of pieces of patch update is less than the number of distributors seeding the update. Under reasonable circumstances, we can assume that the number of distributors will be less than the number of pieces of the patch update (a 1MB file is split into 64 pieces). Also, in case the number of distributors are comparable to the number of pieces, the IoT device can cheat utmost once, after which the misbehaviour will be detected and the IoT node will be blocked.

8.6.3 Properties

In this section, we will prove the properties of the protocol concerning the security goals in Section 8.4.3:

1. **Fair exchange**: The protocol construction is based on first delivering the piece from the distributor to the IoT device and, in return, receiving the signature assuring the reward claim via the lightning network. In the protocol, the distributor can be listed on

the DSN by first downloading the patch and hashing the file into an address, meaning that the distributor holds the patch binaries ready to deliver. Each piece in the torrent protocol is hashed and can be verified by the receiver. In the case of delivering false pieces from the distributor to the IoT device, the IoT will validate the piece against the hash and confirm its validity.

The second phase is IoT signing the requested message. In this construction, the distributor can validate the signature only after receiving the signature, thus, putting himself in risk and will be compromising a fraction of the reward once per IoT lifetime. Since this phase starts only after validating the IoT public address in the handshake phase, the distributor can hold a blacklist of IoT, making it protected the next time the IoT requests a piece. In the transitional construction of the gradual release, both parties should have the same computational power to eliminate the ability of one party to abort between rounds and brute force the remaining information In this protocol, the availability of the piece is from multiple sources, eliminating the incentive of one party to redraw from the transfer.

2. **Patch integrity**: The patch update file hash is listed in the BitTorrent protocol header. This, along with the use of a collision-free hash function patch, will lose its integrity with insignificant probability.

3. **Patch availability**: In P^{4}UIoT, the initial seed of the vendor is used throughout the process. The decentralized storage and bitcoin network with the seed mentioned above ensures that at least one distributor can complete the file download, and thus, further ensures the availability of the patch update.

8.6.4 Execution Results

To evaluate the performance of the proposed framework, an experimental setup is created on the *regtest* network of Bitcoin using lxd containers. The *regtest* is configured to mimic the Bitcoin mainnet by adjusting the mining delay. Also, to simplify the scenario, a hub-spoke configuration is established, where the hub is running a full node, while all other entities run a light client connected to the hub. The containers are created on an 8 core, 30GB machine with 1tb disk space.

Table 8.2: Performance measure of distribution latency (sec).

Number of IoT Devices	**Patch Size**		
	10kb	**100kb**	**1mb**
1	0.532	13.027	128.513
10	1.41	13.56	133.75
25	2.8	18.77	177.12
50	5.81	34.88	336.94

Table 8.3: Cost measure (millisatoshi).

Number of IoT Devices	**Patch Size**		
	10kb	**100kb**	**1mb**
1	1	7	64
10	10	70	640
25	25	175	1600
50	50	350	3200

Table 8.4: Cost (in $) analysis and comparison with IoTPatchPool.

Transaction Type	**Usage Frequency**	**Proposed Solution ($)**	**IoTPatchPool ($)**
Create factory	per framework setup	–	0.36
Create contract	per update file	–	0.29
Create channel	per IoT device	0.017	–
Commit	per delivery	–	0.08
Reveal	per delivery	–	0.02
Delivery cost	per piece	0.000097 + 1% of transaction amount	–

The experiment consists of different scenarios with a varying number of IoT devices and the size of the patch update. The number of vendors and distributors is fixed to understand the scalability of the network. The number of IoT devices are varied as 1, 10, 25, and 50. The patch size is varied as 10kb, 100kb, 1mb, 10mb, and 100mb. The latency of patch delivery and the cost (transaction cost and fee) involved in delivering the patch update to every IoT device is recorded and presented in Table 8.2 and Table 8.3. The patch is split into equal size parts as per the BitTorrent protocol. Also, for the sake of simplicity, we assume the piece price to be one milli satoshi. The negotiation for the price of a piece can be performed offline and is beyond the scope of this work. Figure 8.5 presents the variation of latency against the number of IoT devices for patch size 10kb, 100kb, and 1mb. We can see from Figure 8.5 and Figure 8.6 that the distribution latency is almost linear for small size patch, while it increases exponentially for larger file size. After establishing a payment channel, the payment between two parties happens in a peer-to-peer fashion. Thus, the only latency in payment of incentive is the network delay, which in turn depends on the length of the lightning channel between the IoT device and the distributor. Also, the lightning network adds pseudo-path to the route to protect the privacy of the participants. In a real-world scenario, with more distributors participating in the process, the distribution latency will reduce even further with the distributor being near to the IoT device. Table 8.4 presents a monetary comparison between the proposed solution and IoTPatchPool, which is based on a blockchain-based patch delivery framework. We can observe from the table that the cost of

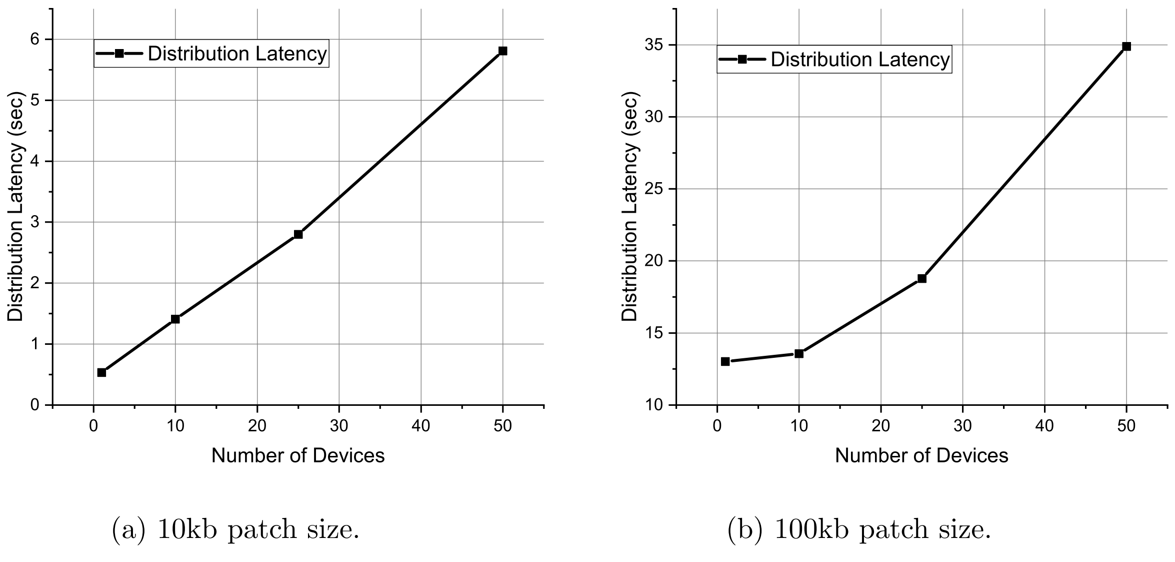


Figure 8.5: Distribution latency for patch size (a) 10kb, and (b) 100kb patch size.

channel establishment is significantly less compared to the contract deployment. Also, the contract deployment needs to be done for each patch update, while the channel establishment needs to be done only once and can be used until the commitment exhausts. Regarding the incentive for the patch delivery, the fees involved in the lightning network consists of two components, a standard base fees, which is roughly one satoshi, and liquidity fees, which is roughly 1% of the transacted amount. Considering the micropayment use case, the overall cost of the patch delivery will be significantly less compared to any pure blockchain-based solution.

8.7 Conclusion

We described and implemented P^{4}UIoT, a pay-per-piece patch update for IoT software updates using a distributed storage network. The proposal combines a distributed file-sharing network and lightning network-based payment channels to transfer patch updates to the IoT devices. The lightning channel enables quick micropayments with minimum delay and zero transaction fees. A fair exchange of patch updates and micropayments is established gradually using a pay-per-piece exchange protocol. The privacy of the framework is maintained by the onion routing of the lightning network. Thus, the information flows between the participating entities. We presented a formal specification of the proposed approach using the TLA+ formal language and checked the correctness of the model using the TLC model checker. We evaluated the resilience of the proposed framework against known threats like reward interception, malformed channel, Denial-of-service (DoS) redeem with false message, exhausting resources, Blockchain-related attacks, compromising a firmware's integrity, software downgrade attack, greedy distributor, and greedy IoT node. Also, the distribution

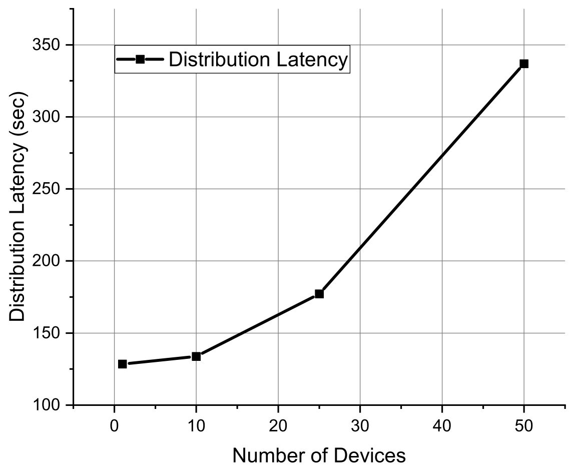


Figure 8.6: Distribution latency for 1mb patch size.

latency and financial analysis of the framework is experimented, and the scalability of the framework is evaluated. As future work, formal security analysis can be conducted using tools like Scyther. Also, the statistical and cost analysis of the framework in a real-time scenario can be conducted to evaluate production readiness.

Conclusions

In this chapter, we summarize the results presented in the thesis, and we give some ideas for the possible development of the research carried out so far.

Obtained results

The blockchain can be effectively used as a trustless mechanism to address some of the IoT ecosystem's critical issues. The trustless nature of the blockchain effectively complements the security requirements of an IoT system. Other features like immutability, transparency, distributed architecture, auditability, and consensus make it a suitable fit for the IoT use case. On the other hand, the latency associated with the mining of the blocks on the blockchain and the devices' capability to interact with the blockchain restricts the applications trying to use it. We need to introduce specific workarounds without compromising the security guarantees. Even with the limitations, the introduction of blockchain to address the IoT challenges is necessary for widening the application field, even if this implies an increase in computational complexity and latency.

In the first part of this thesis, we have investigated and showed the advantages of using blockchain and authenticated data structures for access control and delegation in IoT. In the resource sufficient device context (Chapter 2), we have focused on blockchain-based implementation, and we have proposed an access control and delegation mechanism for an IoT framework using blockchain. Our methodology can reduce the latency involved in block mining by segregating the tasks involved and then designing the mechanism such that the frequency of blockchain affecting operations is reduced. Also, we were able to maintain an audit trail of the operations for which we proposed a novel mechanism to reduce the latency. We evaluated our proposal's performance on test environments, which mimicked the real production environment and demonstrated the usability of the proposal. We have also shown that while the proposed solution guarantees a typical IoT framework's security requirements, the system's performance is highly dependent on the latency of the blockchain solution being used.

Similarly, in the context of resource-constrained devices (Chapter 3), we have implemented a lightweight access control and delegation technique using authenticated data structures. We proposed architecture for auditable authorizations with strong (cryptographic) guarantees suitable to IoT environments. The architecture allowed the support for constrained IoT devices characterized by low computing capabilities and a limited or absent

Internet connection. At the same time, the architecture ensured the capability of system auditing for demonstrating possible illegitimate accesses. We described the proposed system for auditable authorizations in IIoT environments in three steps. First, we described the adopted access control model, followed by its architecture and operations framework. Next, we described the details of the proposed authorization protocol. Finally, we presented the analysis and evaluation of the proposed mechanism. The main drawback of this kind of approach is that the proposal's accountability guarantee assumes the availability of the service. The proposed approach is very lightweight as it requires only storage of a few public keys, and it does not require any Internet connectivity for the things. However, the proposed system requires a trusted setup of these keys and the resource owner to verify illegitimate authorization denials.

In the next part of this thesis, we have investigated blockchain's use to secure data aggregation, storage, and visualization in the IoT framework. We addressed the immutability and transparency challenges related to data storage in the cloud (Chapter 5). A protocol enabling the auditing of interactions between customers and providers and making it widely available and its contents, therefore, was proposed. We presented a blockchain-based solution to solve the problem of trust detailing the reference architecture. We concluded a performance analysis of the proposed solution and compared it with the existing methodology. The proposed mechanism ensures transparency and auditability in any cloud storage system to efficiently integrate any blockchain platform. However, the latency of the underlying platform will be the deciding factor for adopting the solution.

Next, we extended the experience of securing the stored data by focusing on securing the entire supply chain of data from production to storage (Chapter 6). We proposed building an IoT data collection framework that is decentralized (without any single point of failure), transparent (so that everything could be easily verifiable by everybody), and trustless (no requirement for any involved entity to be fully trusted, for the system to work). At the data origin, the cryptochip was used to partially act as a secure enclave for keys and on-chip (i.e., host-invisible) crypto operations (signing, authenticating, encrypting, etc.), thus ensuring that known and authorized sources produce each piece of data. At the data storage level, an adapted version of the BigchainDB platform was proposed. Based on blockchain technology, BigchainDB guarantees decentralization (no single entity controls the data), scalability (multiple endpoints to which data can be delivered), and, last but not least, data immutability. The limitation of the proposed mechanism is its weakness against coercing participants. Since the involved platform forms a coalition of trusted organizations, there is a possibility of misbehavior.

In the final part of the thesis, we presented the blockchain as a mechanism to secure the IoT devices by applying the security patch in a distributed manner (Chapter 8). We described and implemented a pay-per-piece patch update mechanism for IoT software updates using a distributed storage network. The proposal combined a distributed file-sharing network and lightning network-based payment channels to transfer patch updates to the IoT devices. We presented a formal specification of the proposed approach using the TLA+ formal language and checked the model's correctness using the TLC model checker. We evaluated the resilience of the proposed framework against known threats such as reward interception, malformed channel, Denial-of-service (DoS) redeem with false message, exhausting resources, Blockchain-related attacks, compromising a firmware's integrity, software downgrade attack, greedy distributor, and greedy IoT node. The framework's distribution latency, financial analysis, and scalability were experimented with and evaluated. We can conclude that a blockchain-based distribution network ensures the security guarantees simultaneously, making it scalable.

Future work

The obtained results encourage future work in all three fields of the IoT framework. With expansion of IoT technologies, numerous different assets are completely embedded for comprehensive IoT applications. IoT system features such as node heterogeneity, open environment, and multiparty resource sharing raises new requirements for access control models and mechanisms. A further investigation needs to be carried out with the following aspects of access control for IoT. An immediate challenge is on performance. The scale of the IoT is growing sharply and managing IoT devices that grow exponentially is a great challenge to the performance of blockchain. The throughput and latency issues of the blockchain may affect the efficiency of IoT management. To a certain extent we tried to address these issues in our proposals. However, a more comprehensive study is required. In particular, the blockchain based on the PoW consensus not only has the problem of throughput and delay, but also consumes a lot of computing resources. Compared with the public blockchain, the consortium blockchain has higher throughput and stronger control ability; however, it requires higher trust requirements than the public blockchain. Therefore, combining the consortium blockchain and the public blockchain is a potential solution. Other solutions are also possible, which remain to be studied. Another challenging issue is on user privacy protection. Blockchain uses consensus to maintain a ledger. All the nodes are able to see the transactions. Enforcing access control on the chain may leak user's privacy because whether subjects are able to access an object is recorded on the chain. In order to prevent such leakage, techniques such as zero-knowledge proof could be used for privacy protection. However, it may incur more complexity and inefficiency. The design also depends on the detailed application. Security of the smart contract that enforces access control is also an issue. There are also many security issues about the blockchain itself. Although the blockchain is considered to contribute to the security protection of the IoT, the combination of the two technologies however is still difficult to guarantee the overall security of the IoT system. Smart contracts may be implemented with flaws. For example, Ethereum was found to have transaction order dependencies, time stamp dependencies, re-entrant attacks, contract calls vulnerabilities, etc., which could be exploited by hackers. This may lead to system anomalies, unauthorized access, privacy issues, etc., which is thus also a major threat to the IoT system security. Finally, multiparty authorization's conflicts is also an issue. Many RBAC-related proposals focus on incorporating interpersonal relationships in access decision making. They assume that resources are owned by a single entity, ignoring the characteristic of multiparty sharing. More efforts are required to focus on the policy conflict resolution caused by different authorizations. Also, authentication and anonymous protection of physical devices in the

IoT should be given more attention. This will ensure the trustworthiness of the data source, privacy, and data availability.

With respect to data acquisition, storage, and visualization, another research challenge is to improve data provenance reliability. In existing researches, blockchain is used to protect the provenance data; but in fact, the blockchain cannot really guarantee the reliability of the data source which is input by real world entities. Similarly, the blockchain can build trust in different trust domains, but it only guarantees that the data exchange and storage is not tampered with and leaked. Indeed, the blockchain is hard to recognize the validity of the input data. Therefore, it is often necessary to seek a way to prove that the input data is correct, which is difficult and application dependent. Cryptographic mechanisms such as authentication and digital signature could be used to establish such a reliability. This approach may add costs for the blockchain and smart contracts. Other approaches remain to be studied. Similar data provenance can be applied to other fields like developing a platform that enables the attestation of the 3D designs and promotes truthfulness.

For a blockchain-based patch distribution framework, the statistical and cost analysis of the framework in a real-time scenario needs to be conducted to evaluate production readiness. Also, other solutions for blockchain scalability needs to be studied. The leading solution currently discussed in the blockchain literature makes use of the concept of sharding (163; 164; 165; 166; 167; 168; 169; 170). The key idea behind sharding is to divide or split the network into subsets, called shards; each shard will be working on a different set of transactions, rather than the entire network processing the same transactions. This allows the network to scale with the numbers of shards, allowing the throughput and the storage to achieve high efficiency, yet potentially compromising the security.

Finally, a common direction for any blockchain-based mechanism is to develop a formal modeling and verification solution to evaluate any proposed solution. Using tools like Scyther (171), a mechanism for formally evaluating can be developed. Also, ensuring the security of smart contracts is also challenging. A smart contract is a piece of code that can be executed automatically. However, code could also be vulnerable and be attacked. Indeed, real world contracts have been attacked (172). When employing smart contracts to protect IoT data security, approaches to ensure the contract code and logic security worth studying.

Bibliography

[1] L. Goasduff, “Gartner says 5.8 billion enterprise and automotive iot endpoints will be in use in 2020,” 2019. [https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io], 2019.

[2] W. Bamiduro and R. v. d. Meulen, “Gartner says worldwide iot security spending will reach $1.5 billion in 2018,” 2018. [https://www.gartner.com/en/newsroom/press-releases/2018-03-21-gartner-says-worldwide-iot-security-spending-will-reach-1-point-5-billion-in-2018], 2018.

[3] C. Petrov, “47 stunning internet of things statistics 2020 [the rise of iot],” 2020. [https://techjury.net/stats-about/internet-of-things-statistics/gref], 2020.

[4] D. Lynkova, “Iot statistics and trends to know in 2020,” 2019. [https://leftronic.com/internet-of-things-statistics/], 2019.

[5] S. R. Department, “Internet of things (iot) connected devices installed base worldwide from 2015 to 2025,” 2016. [https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/], 2018.

[6] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito, “Stack4things: a sensing-and-actuation-as-a-service framework for iot and cloud integration,” *Annals of Telecommunications*, vol. 72, pp. 53–70, Feb 2017.

[7] T. Morris, “Trusted platform module.,” 2011.

[8] J. Poon and T. Dryja, “The bitcoin lightning network: Scalable off-chain instant payments,” 2016.

[9] L. Lamport, *Specifying systems: the TLA+ language and tools for hardware and software engineers.* Addison-Wesley Longman Publishing Co., Inc., 2002.

[10] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, “Blockchain and iot integration: A systematic survey,” *Sensors*, vol. 18, no. 8, p. 2575, 2018.

[11] N. Tapas, G. Merlino, and F. Longo, “Blockchain-based iot-cloud authorization and delegation,” in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 411–416, IEEE, 2018.

[12] N. Tapas, F. Longo, G. Merlino, and A. Puliafito, “Experimenting with smart contracts for access control and delegation in iot,” *Future Generation Computer Systems*, 2020.

[13] L. Ferretti, F. Longo, M. Colajanni, G. Merlino, and N. Tapas, “Authorization transparency for accountable access to iot services,” in *2019 IEEE International Congress on Internet of Things (ICIOT)*, pp. 91–99, IEEE, 2019.

[14] N. Tapas, G. Merlino, F. Longo, and A. Puliafito, “Blockchain-based publicly verifiable cloud storage,” in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 381–386, IEEE, 2019.

[15] A. Khare, G. Merlino, F. Longo, A. Puliafito, and O. P. Vyas, “Toward a trustless smart city: the# smartme experience,” in *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pp. 204–207, IEEE, 2019.

[16] A. Khare, G. Merlino, F. Longo, A. Puliafito, and O. P. Vyas, “Design of a trustless smart city system: the# smartme experiment,” *Internet of Things*, vol. 10, p. 100126, 2020.

[17] N. Tapas, Y. Yitzchak, F. Longo, A. Puliafito, and A. Shabtai, “P4uiot: Pay-per-piece patch update delivery for iot using gradual release,” *Sensors*, vol. 20, no. 7, p. 2156, 2020.

[18] J. Crampton and H. Khambhammettu, “Delegation in role-based access control,” *International Journal of Information Security*, vol. 7, no. 2, pp. 123–136, 2008.

[19] K. Hasebe, M. Mabuchi, and A. Matsushita, “Capability-based delegation model in rbac,” in *Proceedings of the 15th ACM symposium on Access control models and technologies*, pp. 109–118, 2010.

[20] J. B. Joshi and E. Bertino, “Fine-grained role-based delegation in presence of the hybrid role hierarchy,” in *Proceedings of the eleventh ACM symposium on Access control models and technologies*, pp. 81–90, 2006.

[21] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, “A community authorization service for group collaboration,” in *Proceedings Third International Workshop on Policies for Distributed Systems and Networks*, pp. 50–59, IEEE, 2002.

[22] J. Wainer, A. Kumar, and P. Barthelmess, “Dw-rbac: A formal security model of delegation and revocation in workflow systems,” *Information Systems*, vol. 32, no. 3, pp. 365–384, 2007.

[23] L. Zhang, G.-J. Ahn, and B.-T. Chu, “A rule-based framework for role-based delegation and revocation,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 3, pp. 404–441, 2003.

[24] J. Qiu, Z. Tian, C. Du, Q. Zuo, S. Su, and B. Fang, "A survey on access control in the age of internet of things," *IEEE Internet of Things Journal*, 2020.

[25] N. Foukia, D. Billard, and E. Solana, "Pisces: A framework for privacy by design in iot," in *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pp. 706–713, IEEE, 2016.

[26] . ISO/IEC, "Information technology—security techniques—privacy framework, jtc 1/sc 27," 2011. [http://www.iso.org/iso/isocatalogue/cataloguetc/cataloguedetail.htm?csnumber=45123], 2011.

[27] J. H. Park, S. Gritzalis, C.-H. Hsu, M. Karyda, and S. Kokolakis, "Privacy and fair information practices in ubiquitous environments," *Internet Research*, 2009.

[28] A. Skendžić, B. Kovačić, and E. Tijan, "General data protection regulation—protection of personal data in an organisation," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1370–1375, IEEE, 2018.

[29] F. H. Cate, "The failure of fair information practice principles," *Consumer protection in the age of the information economy*, 2006.

[30] Z. Tian, X. Gao, S. Su, J. Qiu, X. Du, and M. Guizani, "Evaluating reputation management schemes of internet of vehicles based on evolutionary game theory," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 5971–5980, 2019.

[31] Z. Tian, S. Su, W. Shi, X. Du, M. Guizani, and X. Yu, "A data-driven method for future internet route decision modeling," *Future Generation Computer Systems*, vol. 95, pp. 212–220, 2019.

[32] V. G. Cerf, "Access control and the internet of things," *IEEE Annals of the History of Computing*, vol. 19, no. 05, pp. 96–c3, 2015.

[33] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, "Senshare: transforming sensor networks into multi-application sensing infrastructures," in *European Conference on Wireless Sensor Networks*, pp. 65–81, Springer, 2012.

[34] T. ETSI, "102 690: Machine-to-machine communications (m2m)," *Functional architecture*, 2011.

[35] L. Zingales, "Causes and effects of the lehman brothers bankruptcy," *Committee on Oversight and Government Reform US House of Representatives*, pp. 23–25, 2008.

[36] D. Ferraiolo, R. Kuhn, and R. Sandhu, "Rbac standard rationale: Comments on "a critique of the ansi standard on role-based access control"," *IEEE Security Privacy*, vol. 5, pp. 51–53, Nov 2007.

[37] L. Seitz, G. Selander, and C. Gehrmann, “Authorization framework for the internet-of-things,” in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th Int. Symp. and Workshops on*, pp. 1–6, 2013.

[38] B. Anggorojati, P. N. Mahalle, N. R. Prasad, and R. Prasad, “Capability-based access control delegation model on the federated iot network,” in *The 15th International Symposium on Wireless Personal Multimedia Communications*, pp. 604–608, Sept 2012.

[39] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, “Towards a novel privacy-preserving access control model based on blockchain technology in iot,” in *Europe and MENA Cooperation Advances in Information and Communication Technologies*, pp. 523–533, Springer, 2017.

[40] L. Y. Chen and H. P. Reiser, “Distributed applications and interoperable systems, 17th ifip wg 6.1 international conference, dais 2017, held as part of the 12th international federated conference on distributed computing techniques, discotec 2017, neuchâtel, switzerland, june 19–22, 2017,” Springer, 2017.

[41] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, “Medshare: Trust-less medical data sharing among cloud service providers via blockchain,” *IEEE Access*, vol. 5, pp. 14757–14767, 2017.

[42] D. Di Francesco Maesa, P. Mori, and L. Ricci, “Blockchain based access control services,” 2018.

[43] M. Steichen, B. Fiz Pontiveros, R. Norvill, W. M. Shbair, and R. State, “Blockchain-based, decentralized access control for ipfs,” 2018.

[44] S. Rouhani, V. Pourheidari, and R. Deters, “Physical access control management system based on permissioned blockchain,” 2018.

[45] U. Uchibeke, S. Kassani, and R. Deters, “Blockchain access control ecosystem for big data security,” 2018.

[46] D. Recordon and D. Reed, “Openid 2.0: a platform for user-centric identity management,” in *Proceedings of the second ACM workshop on Digital identity management*, pp. 11–16, ACM, 2006.

[47] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, “A survey on essential components of a self-sovereign identity,” *Computer Science Review*, vol. 30, pp. 80–86, 2018.

[48] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig, “Authentication and authorization for constrained environments (ace) using the oauth 2.0 framework (ace-oauth),” internet-draft, Dec. Dec. 2018.

[49] P. P. Pereira, J. Eliasson, and J. Delsing, "An authentication and access control framework for coap-based internet of things," in *IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society*, 2014.

[50] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *IEEE Symp. Security and Privacy*, 2013.

[51] V. Karande, E. Bauman, Z. Lin, and L. Khan, "SGX-log: Securing system logs with sgx," in *Proc. ACM Asia Conf. Computer and Communications Security*, 2017.

[52] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside sgx enclaves with branch shadowing," in *Proc. 26th USENIX Security Symp.*, 2017.

[53] R. Xu, Y. Chen, E. Blasch, and G. Chen, "Blendcac: A blockchain-enabled decentralized capability-based access control for iots," *arXiv preprint arXiv:1804.09267*, 2018.

[54] H. Gardiyawasam Pussewalage and V. Oleshchuk, "Blockchain based delegatable access control scheme for a collaborative e-health environment," 08 2018.

[55] R. Di Pietro, X. Salleras, M. Signorini, and E. Waisbard, "A blockchain-based trust system for the internet of things," in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, pp. 77–83, ACM, 2018.

[56] A. Tomescu and S. Devadas, "Catena: Efficient non-equivocation via bitcoin," in *Proc. 38th IEEE Symp. Security and Privacy*, 2017.

[57] O. Novo, "Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT," *IEEE Internet of Things Journal*, vol. 5, Apr. 2018.

[58] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: a system for secure multi-party computation," in *Proc. 15th ACM Conf. Computer and Communications Security*, 2008.

[59] B. Laurie, A. Langley, and E. Kasper, "RFC6962: Certificate Transparency," rfc, Jun. 2013.

[60] J. Bonneau, "Ethiks: Using ethereum to audit a coniks key transparency log," in *Proc. Int'l Conf. Financial Cryptography and Data Security*, Springer, 2016.

[61] L. Ferretti, F. Longo, M. Colajanni, G. Merlino, and N. Tapas, "Authorization transparency for accountable access to iot services," in *Proceedings of the Third International Conference on Internet of Things (ICIOT 2019)*, Jul 2019.

[62] A. Miller, M. Hicks, J. Katz, and E. Shi, "Authenticated data structures, generically," in *Proc. 41st ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages*, 2014.

[63] L. Ferretti, M. Marchetti, M. Andreolini, and M. Colajanni, "A symmetric cryptographic scheme for data integrity verification in cloud databases," *Information Sciences*, vol. 422, pp. 497 – 515, 2018.

[64] D. W. Jones and T. C. Bowersox, "Secure data export and auditing using data diodes," *technology*, vol. 6, p. 7, 2006.

[65] M. Colajanni and M. Marchetti, "A parallel architecture for stateful intrusion detection in high traffic networks," in *Proc. of the IEEE/IST Workshop on" Monitoring, attack detection and mitigation"(MonAM 2006), Tuebingen, Germany*, 2006.

[66] M. Marchetti and D. Stabili, "Anomaly detection of CAN bus messages through analysis of ID sequences," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1577–1583, IEEE, 2017.

[67] D. Fauri, M. Kapsalakis, D. R. dos Santos, E. Costante, J. den Hartog, and S. Etalle, "Leveraging semantics for actionable intrusion detection in building automation systems," in *International Conference on Critical Information Infrastructures Security*, pp. 113–125, Springer, 2018.

[68] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *2007 44th ACM/IEEE Design Automation Conference*, pp. 9–14, IEEE, 2007.

[69] S. Srinivas, D. Balfanz, E. Tiffany, A. Czeskis, and F. Alliance, "Universal 2nd factor (u2f) overview," *FIDO Alliance Proposed Standard*, pp. 1–5, 2015.

[70] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in fog computing for iot: Review, enabling technologies, and research opportunities," *Future Generation Computer Systems*, vol. 87, pp. 278–289, 2018.

[71] J. Renes, P. Schwabe, B. Smith, and L. Batina, "μkummer: efficient hyperelliptic signatures and key exchange for microcontrollers," in *Cryptographic Hardware and Embedded Systems – CHES 2016* (B. Gierlichs and A. Poschmann, eds.), vol. 9813 of *Lecture Notes in Computer Science*, pp. 301–320, Springer-Verlag Berlin Heidelberg, 2016. Document ID: b230ab9b9c664ec4aad0cea0bd6a6732, http://cryptojedi.org/papers/#mukummer.

[72] L. Ferretti, M. Marchetti, and M. Colajanni, "Fog-based secure communications for low-power iot devices," *ACM Trans. Internet Technol.*, vol. 19, pp. 27:1–27:21, Mar. 2019.

[73] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "Vrased: A verified hardware/software co-design for remote attestation," in *Proc. 28th USENIX Security Symposium*, pp. 1429–1446, 2019.

[74] M. M. Villegas, C. Orellana, and H. Astudillo, “A study of over-the-air (ota) update systems for cps and iot operating systems,” in *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, ECSA ’19, (New York, NY, USA), pp. 269–272, ACM, 2019.

[75] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” 2014.

[76] R. Tamassia, “Authenticated data structures,” in *European symposium on algorithms*, pp. 2–5, Springer, 2003.

[77] O. Sefraoui, M. Aissaoui, and M. Eleuldj, “Openstack: toward an open-source solution for cloud computing,” *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[78] D. Bruneo, S. Distefano, F. Longo, G. Merlino, and A. Puliafito, “IoT-cloud authorization and delegation mechanisms for ubiquitous sensing and actuation,” in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pp. 222–227, Dec 2016.

[79] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[80] G. Asharov and C. Orlandi, “Calling out cheaters: Covert security with public verifiability,” in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 681–698, Springer, 2012.

[81] C. Hong, J. Katz, V. Kolesnikov, W.-j. Lu, and X. Wang, “Covert security with public verifiability: Faster, leaner, and simpler,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 97–121, Springer, 2019.

[82] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang, “Enabling security in cloud storage slas with cloudproof,” in *Proc. 2011 USENIX Annual Technical Conf.*, 2011.

[83] L. Ferretti, M. Marchetti, and M. Colajanni, “Verifiable delegated authorization for user-centric architectures and an oauth2 implementation,” in *Proc. IEEE 41st Conf. Computer Software and Applications*, 2017.

[84] N. Tapas, G. Merlino, and F. Longo, “Blockchain-based iot-cloud authorization and delegation,” in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 411–416, 2018.

[85] IETF, “RFC 6749: The OAuth 2.0 Authorization Framework,” Oct. 2012.

[86] IETF, “RFC 6819: OAuth 2.0 Threat Model and Security Considerations,” Jan. 2013.

[87] A. Andreoli, L. Ferretti, M. Marchetti, and M. Colajanni, “Enforcing correct behavior without trust in cloud key-value databases,” in *Proc. IEEE Int’l Conf. Cyber Security and Cloud Computing*, 2015.

[88] Google, “Trillian: General Transparency.” https://github.com/google/trillian/tree/67395f8e7cf7f94ef80ac1846ec10c49a4d6550f, Nov. 2018.

[89] D. D. F. Maesa, P. Mori, and L. Ricci, “A blockchain based approach for the definition of auditable access control systems,” *Computers & Security*, vol. 84, 2019.

[90] F. Magnanini, L. Ferretti, and M. Colajanni, “Efficient license management based on smart contracts between software vendors and service providers,” in *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, pp. 1–6, IEEE, 2019.

[91] “Forbes.” https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#1961125460ba, accessed on 2019-02-08.

[92] S. Zawoad, R. Hasan, and K. Islam, “Secprov: Trustworthy and efficient provenance management in the cloud,” in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 1241–1249, IEEE, 2018.

[93] N. Kaaniche and M. Laurent, “A secure client side deduplication scheme in cloud storage environments,” in *NTMS 2014: 6th International Conference on New Technologies, Mobility and Security*, pp. 1–7, 2014.

[94] M. Anisetti, C. A. Ardagna, E. Damiani, F. Gaudenzi, and R. Veca, “Toward security and performance certification of open stack,” in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pp. 564–571, IEEE, 2015.

[95] S. Ibba, A. Pinna, M. Seu, and F. E. Pani, “Citysense: blockchain-oriented smart cities,” in *Proceedings of the XP2017 Scientific Workshops*, pp. 1–5, 2017.

[96] J. Xie, H. Tang, T. Huang, F. R. Yu, R. Xie, J. Liu, and Y. Liu, “A survey of blockchain technology applied to smart cities: Research issues and challenges,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2794–2830, 2019.

[97] K. Nam, C. S. Dutt, P. Chathoth, and M. S. Khan, “Blockchain technology for smart city and smart tourism: latest trends and challenges,” *Asia Pacific Journal of Tourism Research*, pp. 1–15, 2019.

[98] A. Pieroni, N. Scarpato, L. Di Nunzio, F. Fallucchi, and M. Raso, “Smarter city: smart energy grid based on blockchain technology,” *Int. J. Adv. Sci. Eng. Inf. Technol*, vol. 8, no. 1, pp. 298–306, 2018.

[99] F. Orecchini, A. Santiangeli, F. Zuccari, A. Pieroni, and T. Suppa, “Blockchain technology in smart city: A new opportunity for smart environment and smart mobility,” in *International conference on intelligent computing & optimization*, pp. 346–354, Springer, 2018.

[100] P. K. Sharma, N. Kumar, and J. H. Park, "Blockchain-based distributed framework for automotive industry in a smart city," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4197–4205, 2018.

[101] R. A. Michelin, A. Dorri, M. Steger, R. C. Lunardi, S. S. Kanhere, R. Jurdak, and A. F. Zorzo, "Speedychain: A framework for decoupling data from blockchain for smart cities," in *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pp. 145–154, 2018.

[102] J. Lindsay, "Smart contracts for incentivizing sensor based mobile smart city applications," in *2018 IEEE International Smart Cities Conference (ISC2)*, pp. 1–4, IEEE, 2018.

[103] D. Minoli and B. Occhiogrosso, "Blockchain mechanisms for iot security," *Internet of Things*, vol. 1, pp. 1–13, 2018.

[104] K. Biswas and V. Muthukkumarasamy, "Securing smart cities using blockchain technology," in *2016 IEEE 18th international conference on high performance computing and communications; IEEE 14th international conference on smart city; IEEE 2nd international conference on data science and systems (HPCC/SmartCity/DSS)*, pp. 1392–1393, IEEE, 2016.

[105] E. Reilly, M. Maloney, M. Siegel, and G. Falco, "An iot integrity-first communication protocol via an ethereum blockchain light client," in *2019 IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT)*, pp. 53–56, IEEE, 2019.

[106] G. S. Ramachandran, R. Radhakrishnan, and B. Krishnamachari, "Towards a decentralized data marketplace for smart cities," in *2018 IEEE International Smart Cities Conference (ISC2)*, pp. 1–8, IEEE, 2018.

[107] G. Campobello, S. Serrano, A. Leonardi, and S. Palazzo, "Trade-offs between energy saving and reliability in low duty cycle wireless sensor networks using a packet splitting forwarding technique," *EURASIP Journal on Wireless Communications and Networking*, vol. 2010, no. 1, p. 932345, 2010.

[108] J. Kwon, "Tendermint: Consensus without mining," *Draft v. 0.6, fall*, vol. 1, no. 11, 2014.

[109] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto, "Bigchaindb: a scalable blockchain database," *white paper, BigChainDB*, 2016.

[110] N. Moeller and S. Josefsson, "Ietf draft: Eddsa and ed25519," 2015.

[111] G. Merlino, D. Bruneo, S. Distefano, F. Longo, and A. Puliafito, "Stack4things: integrating iot with openstack in a smart city context," in *2014 International Conference on Smart Computing Workshops*, pp. 21–28, IEEE, 2014.

[112] S. Widup, M. Spitler, D. Hylender, and G. Bassett, "2018 verizon data breach investigations report," 2018.

[113] Laurence Goasduff, "Gartner Says 5.8 Billion Enterprise and Automotive IoT Endpoints Will Be in Use in 2020."

[114] Christo Petrov, "Internet Of Things Statistics 2020 [The Rise Of IoT]."

[115] Darina Lynkova, "IoT Statistics and Trends to Know in 2020."

[116] Statista Research Department, "Internet of Things - number of connected devices worldwide 2015-2025."

[117] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 1093–1110, 2017.

[118] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn, "Iot goes nuclear: Creating a zigbee chain reaction," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 195–212, IEEE, 2017.

[119] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 636–654, IEEE, 2016.

[120] Andrey Muravitsky, Vladimir Dashchenko, Roland Sako, "IoT hack: how to break a smart home… again."

[121] Lily Hay Newman, "An elaborate hack shows how much damage iot bugs can do."

[122] O. Williams-Grut, "Hackers once stole a casino's high-roller database through a thermometer in the lobby fish tank," *Business Insider*, 2018.

[123] Check Point, "The Dark Side of Smart Lighting: Check Point Research Shows How Business and Home Networks Can Be Hacked from a Lightbulb."

[124] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *Network and Distributed Systems Symposium*, 2018.

[125] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 531–548, 2016.

[126] W. He, M. Golla, R. Padhi, J. Ofek, M. Dürmuth, E. Fernandes, and B. Ur, "Rethinking access control and authentication for the home internet of things (iot)," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 255–272, 2018.

[127] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. J. Unviersity, "Contexlot: Towards providing contextual integrity to appified iot platforms.," in *NDSS*, 2017.

[128] Bruce Schneier, "E-mail vulnerabilities and disclosure."

[129] E. M. Schooler, D. Zage, J. Sedayao, H. Moustafa, A. Brown, and M. Ambrosin, "An architectural vision for a data-centric iot: Rethinking things, trust and clouds," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1717–1728, IEEE, 2017.

[130] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[131] David Ramel, "Hundreds of enterprise services reportedly hit by AWS outage."

[132] Gnutella, "Gnutella protocol specification 0.4."

[133] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[134] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, pp. 68–72, 2003.

[135] Tom Warren, "Microsoft to deliver Windows 10 updates using peer-to-peer technology."

[136] Ernesto Van Der Sar, "Bittorrent makes Twitter's server deployment 75x faster."

[137] G. Kreitz and F. Niemela, "Spotify–large scale, low latency, p2p music-on-demand streaming," in *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, pp. 1–10, IEEE, 2010.

[138] Amazon, "Using BitTorrent with Amazon S3."

[139] B. Lee and J.-H. Lee, "Blockchain-based secure firmware update for embedded devices in an internet of things environment," *The Journal of Supercomputing*, vol. 73, no. 3, pp. 1152–1167, 2017.

[140] A. Boudguiga, N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey, "Towards better availability and accountability for iot updates by means of a blockchain," in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 50–58, IEEE, 2017.

[141] J. Lee, "Patch transporter: Incentivized, decentralized software patch system for wsn and iot environments," *Sensors*, vol. 18, no. 2, p. 574, 2018.

[142] O. Leiba, R. Bitton, Y. Yitzchak, A. Nadler, D. Kashi, and A. Shabtai, "Iotpatchpool: Incentivized delivery network of iot software updates based on proofs-of-distribution," *Pervasive and Mobile Computing*, vol. 58, p. 101019, 2019.

[143] K. Liu, D. Zou, and H. Jin, "Uaas: software update as a service for the iaas cloud," in *2015 IEEE International Conference on Services Computing*, pp. 483–490, IEEE, 2015.

[144] X. Zhen-hai and Y. Yong-zhi, "Automatic updating method based on maven," in *2014 9th International Conference on Computer Science & Education*, pp. 1074–1077, IEEE, 2014.

[145] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, pp. 1–7, 2015.

[146] Y. Onuma, Y. Terashima, and R. Kiyohara, "Ecu software updating in future vehicle networks," in *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 35–40, IEEE, 2017.

[147] C. Huth, P. Duplys, and T. Güneysu, "Secure software update and ip protection for untrusted devices in the internet of things via physically unclonable functions," in *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pp. 1–6, IEEE, 2016.

[148] D.-Y. Kim, S. Kim, and J. H. Park, "Remote software update in trusted connection of long range iot networking integrated with mobile edge cloud," *IEEE Access*, vol. 6, pp. 66831–66840, 2017.

[149] S. Popov, "The tangle. white paper.(2017)," *URL: https://iota. org/IOTA_Whitepaper. pdf*, 2017.

[150] E. Adar and B. A. Huberman, "Free riding on gnutella," *First monday*, vol. 5, no. 10, 2000.

[151] D. Hughes, G. Coulson, and J. Walkerdine, "Free riding on gnutella revisited: the bell tolls?," *IEEE distributed systems online*, vol. 6, no. 6, 2005.

[152] S. Kaune, R. C. Rumin, G. Tyson, A. Mauthe, C. Guerrero, and R. Steinmetz, "Unraveling bittorrent's file unavailability: Measurements and analysis," in *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, pp. 1–9, IEEE, 2010.

[153] D. Vorick and L. Champine, "Sia: Simple decentralized storage," *Nebulous Inc*, 2014.

[154] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj a peer-to-peer cloud storage network," 2014.

[155] Viktor Tron, "Swarm alpha public pilot and the basics of Swarm."

[156] Protocol Labs, "Filecoin: A decentralized storage network."

[157] Wuill, Pieter and Nick, Jonas and Ruffing, Tim, "Schnorr Signatures for secp256k1."

[158] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[159] A. A. Imem, "Comparison and evaluation of digital signature schemes employed in ndn network," *arXiv preprint arXiv:1508.00184*, 2015.

[160] Phillip J. Windley, "An overview of Self-Sovereign Identity: the use case at the core of Hyperledger Indy."

[161] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[162] T. Lu, S. Merz, and C. Weidenbach, "Towards verification of the pastry protocol using tla+," in *Formal Techniques for Distributed Systems*, pp. 244–258, Springer, 2011.

[163] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 17–30, 2016.

[164] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 583–598, IEEE, 2018.

[165] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 931–948, 2018.

[166] Z. Team *et al.*, "The zilliqa technical whitepaper," *Retrieved September*, vol. 16, p. 2019, 2017.

[167] A. Manuskin, M. Mirkin, and I. Eyal, "Ostraka: Secure blockchain scaling by node sharding," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 397–406, IEEE, 2020.

[168] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pp. 95–112, 2019.

[169] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proceedings of the 2019 international conference on management of data*, pp. 123–140, 2019.

[170] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," *arXiv preprint arXiv:1708.03778*, 2017.

[171] C. J. Cremers, "The scyther tool: Verification, falsification, and analysis of security protocols," in *International conference on computer aided verification*, pp. 414–418, Springer, 2008.

[172] K. Iyer and C. Dannen, "Gambling," in *Building Games with Ethereum Smart Contracts*, pp. 245–260, Springer, 2018.