https://doi.org/10.26219/heal.aueb.9346

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
School of Information Sciences and Technology
Department of Informatics

Distributed Ledger Technologies meet the Internet of Things: Security, Interoperability, and Advanced Access Control

A dissertation submitted in fulfillment of the requirements for the degree

Doctor of Philosophy

in

Computer Science

by

Iakovos Pittaras

Athens
June 2025

https://doi.org/10.26219/heal.aueb.9346

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all, I would like to express my deepest gratitude to my Ph.D. advisor, Professor George C. Polyzos, for his patient guidance, unwavering support, and belief in my abilities. The opportunities he provided have been invaluable throughout the course of my doctoral studies. His constant motivation, advices, insightful critiques, and patience have not only shaped me as a researcher but also as a person, with his mentorship being a cornerstone of my academic development.

I am also sincerely thankful for the assistance given by Professor Vasilios A. Siris, Assistant Professor Spyridon Voulgaris, and Professor George Xylomenos. Their valuable comments, their constructive feedback, and the thought-provoking questions were crucial in helping me refine my arguments and deepen my research. Their expertise not only guided my work but also enriched my understanding of key research concepts and methodologies.

Many thanks to all senior researchers and professors with whom I had the great opportunity to collaborate during my Ph.D studies. Thank you, Dr. Nikos Fotiou, for guiding me at the early stages of my Ph.D. studies. Our interaction and your expertise have been instrumental in shaping the direction and the quality of my work. Thank you, Dr. Yannis Thomas for giving me life lessons and inspiring me to develop mechanisms to entertain myself, both inside and outside the lab. I also want to thank Christos Karapapas, a great friend and colleague, with whom I enjoyed a wonderful collaboration. Together, we constantly motivated each other and had endless conversations and great fun. I would also like to thank Professor George D. Stamoulis for the great cooperation we had during my time as T.A. and throughout my studies.

I had the privilege of working at the Mobile Multimedia Laboratory with fantastic colleagues and friends. Dr. Merkouris Karaliopoulos, Dr. Livia Chatzieleftheriou, and Dr. Konstantinos Tsioutas were the first people I met there. They were very supportive and working with them was a pleasure and always educating. I would also thank Spiros Chadoulos, who joined the MMLAB almost at the same time as me, and Yiannis Papageorgiou, Alexandros Antonov, Vasilis Kalos, Chalima Dimitra Nassar Kyriakidou, and Athanasia Maria Papathanasiou, who

https://doi.org/10.26219/heal.aueb.9346

represent the new generation of the lab. They are great friends and I wish them the best.

I would like to extend my thanks to all my friends, who stood to me during my Ph.D. journey. Special thanks go to Nikos Yfantis, Konstantinos Yfantis, Konstantinos Skoumas, Manolis Partsinevelos, Konstantinos Tsivolas, Dimitris Agathaggelos, and Nafsika Oikonomou for being great friends over the years.

Last but surely not least, my warmest thanks to my parents, Marina and Agapitos, who have always supported me unconditionally, helping me become the person I am today. Your constant support allowed me to pursue a Ph.D. degree without distractions. Finally, thank you Maro, for being a great sister and always being there for me.

VITA

2017	Diploma in Informatics, Athens University of Economics and Business, Greece
2019	M.Sc. in Computer Science, Athens University of Economics and Business, Greece
2025	Ph.D. in Computer Science, Athens University of Economics and Business, Greece

PUBLICATIONS

Journal Publications

I. Pittaras, N. Fotiou, C. Karapapas, V. A. Siris, and G. C. Polyzos, "Secure smart contract-based digital twins for the Internet of Things," in *Blockchain: Research and Applications*, vol. 5, no. 1, 2024

N. Fotiou, I. Pittaras, V. A. Siris, G. C. Polyzos, and P. Anton, "A privacy-preserving statistics marketplace using local differential privacy and blockchain: An application to smart-grid measurements sharing," in *Blockchain: Research and Applications*, vol. 2, no. 1, 2021

I. Pittaras, N. Fotiou, V. A. Siris, and G. C. Polyzos, "Beacons and Blockchains in the Mobile Gaming Ecosystem: A Feasibility Analysis," in *Sensors*, vol. 21, no. 3:862, 2021

Refereed International Conference and Workshop Papers

C. Karapapas, I. Pittaras, G. C. Polyzos, and C. Patsakis, "Hello, won't you tell me your name?: Investigating Anonymity Abuse in IPFS," in *Proceedings of the 13th International Workshop on Cyber Crime (IWCC)*, Ghent, Belgium, August 2025

Y. Thomas, N. Fotiou, I. Pittaras, and G. Xylomenos, "Secure and Efficient Data spaces over Named Data Networking," *in Proceedings of the 2025 IFIP Networking Conference (IFIP Networking)*, Limassol, Cyprus, May 2025

C. D. Nassar Kyriakidou, I. Pittaras, A. M. Papathanasiou, G. Xylomenos, and G. C. Polyzos, "Performance of Smart Contract-based Digital Twins for the Internet of Things," in *Proceedings of the 2nd ACM International Workshop on Middleware for Digital Twins (Midd4DT)*, Hong Kong, China, December 2024

A. M. Papathanasiou, C. D. Nassar Kyriakidou, I. Pittaras, and G. C. Polyzos, "Smart contract-based decentralized mining pools for Proof-of-Work blockchains," in *Proceedings of the 2024 IEEE International Conference on Blockchain*, Copenhagen, Denmark, August 2024

N. Fotiou, C. D. Nassar Kyriakidou, A. M. Papathanasiou, I. Pittaras, Y. Thomas, and G. Xylomenos, "Certificate Management for Cloud-Hosted Digital Twins," in *Proceedings of 29th IEEE Symposium on Computers and Communications (ISCC)*, Paris, France, June 2024

C. D. Nassar Kyriakidou, A. M. Papathanasiou, I. Pittaras, N. Fotiou, Y. Thomas, and G. C. Polyzos, "Attribute-based Access Control Utilizing Verifiable Credentials for Multi-Tenant IoT Systems," in *Proceedings of the 4th IEEE International Conference on Electronic Communications, Internet of Things, and Big Data (ICEIB)*, Taipei, Taiwan, April 2024

I. Pittaras and G. C. Polyzos, "Multi-tenant, Decentralized Access Control for the Internet of Things," in *Proceedings of the 2023 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)*, Bali, Indonesia, November 2023

Y. Thomas, N. Fotiou, I. Pittaras, G. Xylomenos, S. Voulgaris, and G. C. Polyzos, "Peer clustering for the InterPlanetary File System," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Future of Internet Routing & Addressing (Fira 23)*, New York, USA, September 2023

N. Fotiou, I. Pittaras, S. Chadoulos, V. A. Siris, G. C. Polyzos, N. Ipiotis, and S. Keranidis, "Authentication, Authorization, and Selective Disclosure for IoT data sharing using Verifiable Credentials and Zero-Knowledge Proofs," in *Proceedings of the 5th International Workshop on Emerging Technologies for Authorization and Authentication (ETAA)*, Copenhagen, Denmark, September 2022

A. M. Papathanasiou, C. D. Nassar Kyriakidou, I. Pittaras, and G. C. Polyzos, "R?ddle: A Fully Decentralized Mobile Game for Fun and Profit," in *Proceedings of the 4th International Congress on Blockchain and Applications*, L'aquila, Italy, 2022

C. Karapapas, G. Syros, I. Pittaras, and G. C. Polyzos, "Decentralized NFT-based Evolvable Games," in *Proceedings of the 4th Conference on Blockchain Research and Applications for Innovative Networks and Services (BRAINS)*, Paris, France, September 2022

I. Pittaras and G. C. Polyzos, "Secure and Efficient Web of Things Digital Twins using Permissioned Blockchains," in *Proceedings of the 7th International Conference on Smart and Sustainable Technologies (SpliTech)*, Split and Bol, Croatia, July 2022

I. Pittaras, N. Fotiou, C. Karapapas, V. A. Siris, and G. C. Polyzos, " Secure, Mass Web of Things Actuation Using Smart Contract-based Digital Twins," in *Proceedings of the 27th IEEE Symposium on Computers and Communications (ISCC)*, Rhodes, Greece, June 2022

N. Fotiou, E. Faltaka, V. Kalos, A. Kefala, I. Pittaras, V. A. Siris, and G. C. Polyzos, "Continuous authorization over HTTP using Verifiable Credentials and OAuth 2.0," in *Proceedings of the Open Identity Summit 2022 (OID 2022)*, Lyngby, Denmark, 2022

C. Karapapas, I. Pittaras, and G. C. Polyzos, "Fully Decentralized Trading Games with Evolvable Characters using NFTs and IPFS," in *Proceedings of the Workshop on Decentralizing the Internet with IPFS and Filecoin (DI2F)*, Espoo, Finland, June 2021

C. Karapapas, I. Pittaras, N. Fotiou, and G. C. Polyzos, "Ransomware as a Service using Smart Contracts and IPFS," in *Proceedings of the 2nd IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Toronto, Canada, May 2020

N. Fotiou, I. Pittaras, V. A. Siris, S. Voulgaris, and G. C. Polyzos, "OAuth 2.0 authorization using blockchain-based tokens," in *Proceedings of the 3rd NDSS Workshop on Decentralized IoT Systems and Security (DISS)*, San Diego, CA, USA, February 2020

N. Fotiou, I. Pittaras, V. A. Siris, S. Voulgaris, and G. C. Polyzos, "Secure IoT access at scale using blockchains and smart contracts," in *Proceedings of the 8th WoWMoM Workshop on the Internet of Things: Smart Objects and Services (IoTSoS)*, Washington DC, USA, June 2019

Refereed International Conference and Workshop Demos, Abstracts, and Posters

Y. Thomas, N. Fotiou, I. Pittaras, and G. Xylomenos, "Poster: Named Data Networking for Data Spaces," in *Proceedings of 29th IEEE Symposium on Computers and Communications (ISCC)*, Paris, France, June 2024

I. Pittaras and G. C. Polyzos, "SmartTwins: Secure and Auditable DLT-based Digital Twins for the WoT," in *Proceedings of the 18th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Los Angeles, CA, USA, May 2022

N. Fotiou, I. Pittaras, V. A. Siris, S. Voulgaris, and G. C. Polyzos, "Securing IoT services using DLTs and Verifiable Credentials," in *Proceeding of the Network and Distributed Systems Security Symposium 2020 (NDSS 2020)*, San Diego, CA, USA, February 2020

https://doi.org/10.26219/heal.aueb.9346

N. Fotiou, I. Pittaras, V. A. Siris, and G. C. Polyzos, "Enabling opportunistic users in multi-tenant IoT system using decentralized identifiers and permissioned blockchains," in *Proceedings of the 2nd ACM Workshop on Internet of Things Security and Privacy (IoT S&P)*, London, UK, November 2019

https://doi.org/10.26219/heal.aueb.9346

ABSTRACT OF THE DISSERTATION

Distributed Ledger Technologies meet the Internet of Things: Security, Interoperability, and Advanced Access Control

by

Iakovos Pittaras

Doctor of Philosophy in Computer Science

Athens University of Economics and Business, Athens, 2025

Professor George C. Polyzos, Chair

The Internet of Things (IoT), where ordinary physical devices and appliances are accessed through the Internet, communicate with each other, and operate unattended and autonomously, reshapes the status quo of the modern Internet and many of its underlying mechanisms. The IoT is an enticing paradigm receiving increasing attention from the research community due to the multitude of important and often sensitive services that provides across diverse use cases. However, the proliferation of IoT devices and applications requiring cooperation among heterogeneous devices and the exchange of (often sensitive) information between multiple entities, having no or little trust relationships, highlights the need for seamless interoperability, enhanced intrinsic security, decentralization, and auditable, fine-

https://doi.org/10.26219/heal.aueb.9346

grained, decentralized, and flexible access control. Consequently, although in principle the IoT appears to offer numerous benefits, when it comes to its realization many concerns are raised. In particular, one of the most critical challenges that the IoT must address is the protection of IoT devices and data generated within IoT systems. Yet, most conventional access control solutions, designed for centralized, less dynamic environments and for non resource-constrained devices, have been proven inefficient and inadequate for IoT's unique demands. It has, therefore, become evident that existing IoT solutions need to be reassessed and new access control solutions have to be developed.

This dissertation argues that *Distributed Ledger Technologies* (DLTs) offer a promising approach to meet the (security) requirements of both IoT systems and IoT access control solutions. Our contribution towards that direction is twofold. First, we analyze the related literature and derive key requirements that IoT access control should fulfill. Based on these, we propose novel DLT-based access control solutions that leverage smart contracts to issue and manage Access Control Tokens (ACTs) and to serve as decentralized Policy Decision Points (PDPs), thereby offering strong security properties. To demonstrate feasibility, we integrate our solutions with the OAuth 2.0 protocol. In addition, we explore how the consensus mechanism of Hyperledger Fabric can itself serve as a PDP for access control in multi-tenant, collaborative IoT systems. Second, we demonstrate how blockchain technology can be integrated into IoT systems and architectures, highlighting its advantages in use cases, such as gaming and digital twining. We show how this integration, combined with the Web of Things (WoT) paradigm, can enhance IoT systems and applications, especially in terms of security and interoperability.

https://doi.org/10.26219/heal.aueb.9346

Chapter 1

Introduction

The technological landscape has undergone a significant transformation in recent years. Emerging systems and applications depart from traditional centralized paradigms to decentralized ecosystems of interconnected devices. The *Internet of Things* (IoT) represents a salient paradigm of this transformation that is rapidly gaining attention, reshaping the status quo of the current Internet and Web. Unlike the traditional Internet, the IoT enables physical devices, ranging from home appliances to industrial machinery, to communicate with each other autonomously. Specifically, the IoT is envisioned to be an ecosystem of interconnected devices, e.g., sensors and actuators, that can collect, share (sensors), and act (actuators) on data, having as a goal to merge the cyber with the real world, providing a multitude of services that improve the quality of our lives.

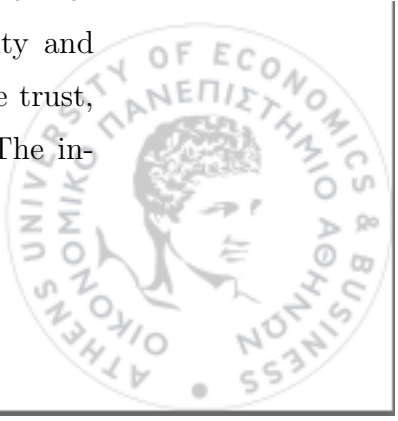
The IoT is rapidly gaining adoption due to its wide-ranging applications and offered potentialities that have great impact on several aspects of everyday life. According to Cisco, 500 billion devices will be connected to the Internet by 2030.[1] The increasing adoption of the IoT spans diverse sectors, as it has the ability to address critical everyday and industrial challenges [1]. For instance, in healthcare, the IoT supports real-time patient monitoring and remote care, among others, significantly improving healthcare services. In agriculture, the IoT enables smart irrigation and soil monitoring, enhancing productivity and promoting sus-

[1] https://www.cisco.com/c/dam/global/fr_fr/solutions/data-center-virtualization/big-data/solution-cisco-sas-edge-to-entreprise-iot.pdf

tainable farming. In urban environments, the use of the IoT can lead to mitigation of traffic congestion, enhance energy efficiency, and bolster public safety. Similarly, manufacturing can benefit from the IoT by enabling digital twining that facilitates predictive maintenance and streamlining supply chain operations, driving the principles of Industry 4.0. These applications demonstrate the IoT's vast potential to enhance productivity, sustainability, and quality of life, representing just a few of the many areas and use cases in which the IoT can be integrated.

However, despite its enormous potential, the IoT faces several challenges that must be addressed to realize its full promise. The proliferation of IoT devices has introduced interoperability issues due to the lack of standardization across protocols and manufacturers. In addition, security and privacy challenges have become critical as IoT systems handle vast amounts of private and sensitive data [2]. The first step into securing IoT systems, devices, and data is through access control, as it is often considered the backbone technology to ensure security. The ultimate goal of access control is to prevent unauthorized access to resources and unauthorized flow of information. However, most IoT devices are resource-constrained, hence they cannot support the current complex security solutions, constituting traditional access control and security solutions inappropriate for the IoT. Moreover, the reliance on centralized infrastructures conflicts with the IoT's decentralized nature and can lead to single point of failures and inefficiencies. Consequently, it is clear that the current Internet and Web architectures and security solutions cannot effectively and efficiently handle various challenges introduced by the new paradigms, including security, privacy, access control, interoperability, and decentralization, due to the shortcomings in their original design.

In this context, blockchains and Distributed Ledger Technologies (DLTs) [3] more general (for simplicity, we will use these terms interchangeably throughout the dissertation) have emerged as a promising solution to address the challenges that the IoT currently faces. DLTs, originally developed as the foundation for cryptocurrencies, such as the Bitcoin [4], have gained immense popularity and recognition due to their inherent and intriguing properties, such as unique trust, transparency, decentralization, auditability, immutability, and security. The in-

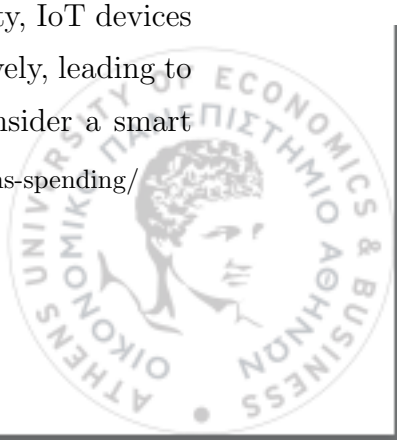


creasing popularity of blockchain solutions is reflected in the global spending on these solutions, which is projected to reach $19 billion by the end of 2024.[2] Blockchain technology has increasingly being adopted in various domains, including supply chains [5], the energy sector [6], smart agriculture [7], etc., showcasing its versatility and transformative potential. Especially in the context of the IoT and IoT security, these properties align well with the IoT's needs, where devices must interact autonomously and often involve multiple entities with no trust relationships between them. By many, DLTs and smart contracts are expected to revolutionize [8] and bring "democracy" in the IoT domain [9], facilitating alternative communication paradigms and enabling novel security mechanisms. However, the potential of DLTs to redefine traditional systems is confined, as they come with some drawbacks, including poor performance and high costs, which require careful consideration. Overcoming these limitations or balancing the gains and the drawbacks will be crucial for integrating IoT and blockchains, paving the way for a secure, interoperable, and decentralized future.

1.1 Motivation for the dissertation

The IoT is envisioned to improve the quality of our lives, by seamlessly interconnecting cyber and real worlds. However, realizing this vision requires addressing several significant challenges. Currently, there is a plethora of IoT devices produced by different manufacturers, each employing different protocols and data formats. On top of that, IoT ecosystems involve multiple business entities with their own and often competing goals, objectives, and priorities. This diversity has resulted in IoT systems being fragmented and far from the goal of one unified and interoperable IoT envisioned. Interoperability is a crucial property for IoT systems, as it enables diverse devices and applications to work together, regardless of their underlying protocols and technologies. Without interoperability, IoT devices from different manufacturers cannot communicate easily and effectively, leading to isolated "silos" of information and functionality. For instance, consider a smart

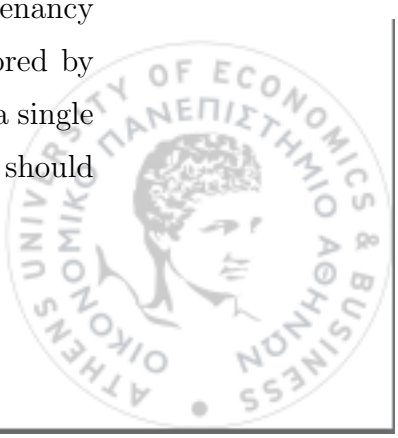
[2] https://www.statista.com/statistics/800426/worldwide-blockchain-solutions-spending/



home environment, where there are IoT devices, such as temperature sensors and thermostats. These IoT devices should communicate seamlessly to maximize effectiveness and work autonomously, e.g., when the temperature drops below 15° C, the heating system should be activated automatically. Without such interaction, their functionality and efficiency are significantly limited. Beyond that, there is also application-level fragmentation in IoT ecosystems. Most manufacturers introduce their own client applications, forcing end-users to install, and switch between, multiple applications just to monitor or control their IoT devices. This proliferation of vendor-specific applications further undermines the unified IoT experience.

In addition, as the IoT becomes more and more popular and integrated into daily life, it is clear that it generates and manages sensitive and even personal data in some cases, e.g., health data generated by wearable devices. Furthermore, the environment in which IoT devices need to operate is mostly unpredictable, highly dynamic, and continuously changing, and since the real world can be directly impacted, given the pervasiveness of IoT systems, security and privacy are serious concerns and are considered two of the major challenges that the IoT is facing [2]. Securing IoT devices and services requires complex security solutions using advanced cryptographic techniques and algorithms. However, these solutions have not been designed for the IoT, in which many IoT devices are usually resource-constrained and lack the computational power to handle these complex operations. Therefore, more lightweight solutions tailored to the limitations of IoT devices are required.

The first step into securing the IoT is through access control, since it is considered as a foundational aspect to ensure security. In a nutshell, access control is a set of methods that can tag, organize, and manage data and devices in a system [10]. Yet, traditional access control models and mechanisms are not well suited for the IoT, since they are too rigid, structured, and require complex operations. Moreover, they fail to accommodate the collaborative nature and multi-tenancy of the IoT, where resources and devices may be owned by one entity, stored by another entity, and might refer to other entities. In such cases, there is not a single entity solely responsible for the access control, but all the involved entities should



collaborate to achieve that. Except of the above, traditional access control solutions have been proven inadequate for the IoT, due to their reliance on centralized infrastructures. There are cases, where they even be deemed insecure by emerging security paradigms, such as the Zero Trust paradigm [11].

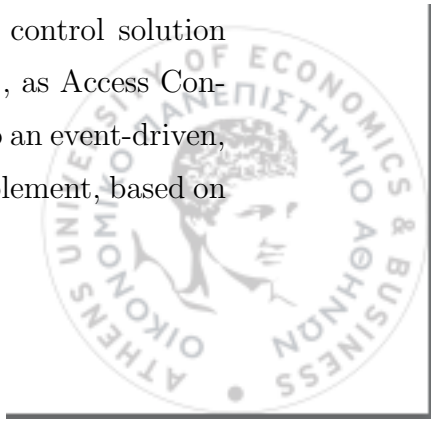
The decentralized nature of DLTs aligns well with the aforementioned challenges of IoT systems. DLT is a new and exciting technology with many intriguing properties, which can help significantly in building secure IoT access control mechanisms and IoT solutions. There are already some approaches that have explored the use of DLTs to address these challenges. However, the use of DLTs is a double-edged sword, as it comes also with some drawbacks, such as performance and costs issues. Therefore, careful consideration is required.

These observations, give us a strong motive to: (i) elicit concrete (security) requirements for IoT access control, (ii) revisit key blockchain-based access control solutions tailored to the IoT, and (iii) design new blockchain-based access control solutions for the IoT, alongside secure, decentralized, and interoperable blockchain-based IoT architectures and systems.

1.2 Contributions

Our contributions, which we present in this dissertation, are the following.

- We review and study two popular blockchains, which are used in the dissertation, establishing their key properties. The two blockchains are Ethereum [12] and Hyperleger Fabric [13]. We also review key access control models and architectures.
- We gather IoT requirements related to access control and security. Based on these requirements, we review and discuss related state of the art solutions.
- We design, implement, and evaluate a token-based access control solution that leverages custom blockchain-based ERC-20 tokens [14], as Access Control Tokens (ACTs). Our access control solution is applied to an event-driven, large-scale IoT management solution that we design and implement, based on



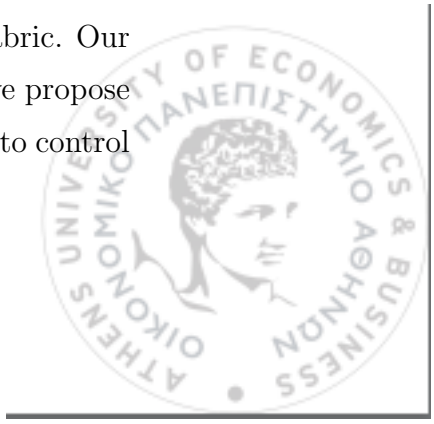
a proposed blockchain-based IoT architecture, offering mass actuation. Furthermore, we demonstrate the use of the proposed ACTs in a multi-tenant smart city scenario.

- We design, implement, and evaluate a new type of ACT, utilizing the ERC-721 token standard [15]. The proposed type of ACT supports auditing, accountability, proof-of-possession, and added value token management services, including delegation, fair exchange, and fast revocation. We integrate our solution with the OAuth 2.0 protocol [16].

- We design, implement, and evaluate a decentralized consensus-based access control solution for multi-party, collaborative IoT environments, based on Hyperledger Fabric blockchain. Furthermore, we design, implement, evaluate, and compare two distinct approaches for implementing consensus-based access control decision. The first approach utilizes smart contracts and the blockchain to act as Policy Decision Point (PDP), while the second approach utilizes the actual consensus mechanism of Hyperledger Fabric, i.e., *the endorsement policy*, to act as PDP.

- We explore the adoption of the IoT and DLTs in the mobile gaming sector, demonstrating how their combination unlocks new business opportunities and enables innovative business models. We evaluate the added overhead introduced by blockchain technology, such as transaction delays and costs, by analyzing both public and private blockchains and their combination. The evaluation is performed in a simulated location-based, context-aware mobile gaming ecosystem, based on defined Key Performance Indicators (KPIs).

- We design, implement, and evaluate secure, decentralized, reliable, auditable, and flexible smart contract-based digital twins for IoT devices utilizing smart contracts and the Web of Things (WoT) standard [17]. We implement our solution in two different blockchains, Ethereum and Hyperledger Fabric. Our design offers increased security and interoperability. In addition, we propose using the smart contract-based digital twins as transparent proxies to control but also isolate the actual IoT devices.



- We present the innovative application of blockchain technology and smart contracts in other domains than the IoT, including blockchain-based games that utilize Non-Fungible Tokens (NFTs) and transparent and fair-exchange marketplaces. We showcase the benefits and the novel features that are enabled due to the use of the blockchain technology.

1.3 Dissertation outline

The remainder of the dissertation is organized as follows. Chapter 2 presents the key properties of blockchain technology and provides a review of two major blockchain technologies, Ethereum and Hyperledger Fabric. Additionally, it provides a review of key access control models and presents an access control architecture, which is used as a reference access control architecture in the following chapters. Chapter 3 presents IoT security and access control requirements and reviews existing (blockchain-based) access control solutions specifically designed for the IoT. Chapter 4 details our novel blockchain-based access control solutions for the IoT. Chapter 5 presents the integration of blockchain technology in IoT systems and architectures, focusing on the cases of mobile gaming and digital twining. Chapter 6 presents the application of blockchain technology in other domains than the IoT. Finally, Chapter 7 presents the conclusions of this dissertation and some potential directions for future research.

Chapter 2

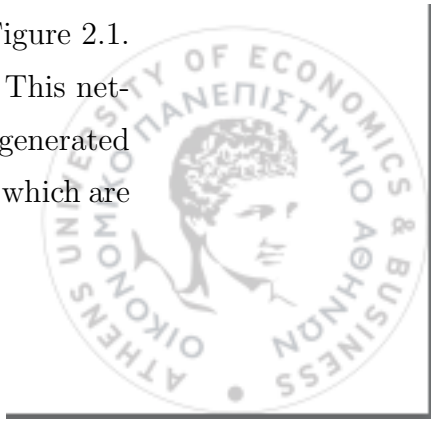
Distributed Ledger Technologies and Access control

In this chapter, we examine the blockchain technology, establishing its key properties, and we review two popular blockchains, namely Ethereum and Hyperledger Fabric. Additionally, we examine key access control models, ultimately selecting a representative access control architecture to serve as a reference architecture in the following chapters.

2.1 Distributed Ledger Technologies

Blockchains [3] have emerged as a groundbreaking technology with the potential to transform a wide range of industries. At its core, this technology provides a decentralized and tamper-resistant method for recording and verifying transactions. Unlike traditional systems, blockchains do not rely on trusted centralized authorities.

A blockchain is an append-only ledger of transactions distributed throughout a network of mutually non-trusting nodes. Hence, they are also referred to as DLTs. A simplified version of a blockchain network is illustrated in Figure 2.1. The nodes participating in the blockchain form a peer-to-peer network. This network is used for connecting the nodes and propagating transactions and generated blocks, based on a gossip protocol. The ledger organizes data into blocks, which are



https://doi.org/10.26219/heal.aueb.9346

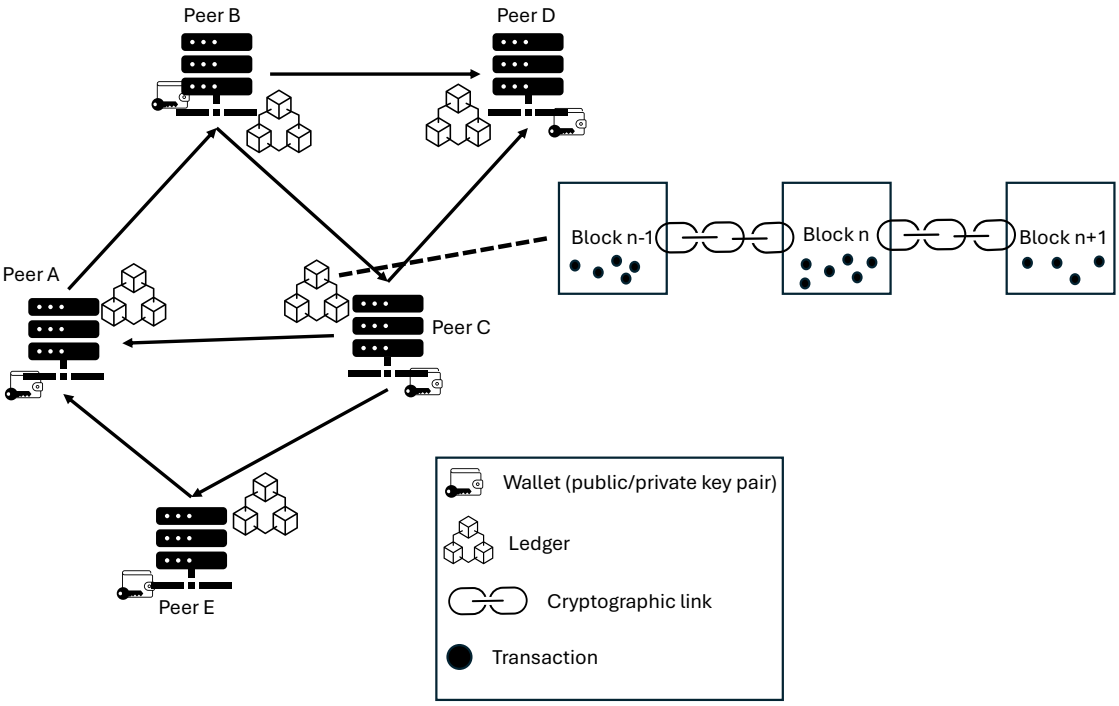
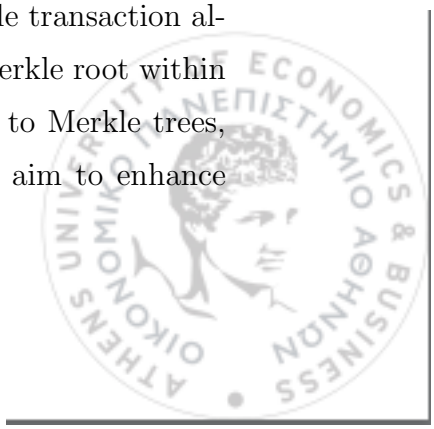


Figure 2.1: High level representation of a blockchain network and its core components.

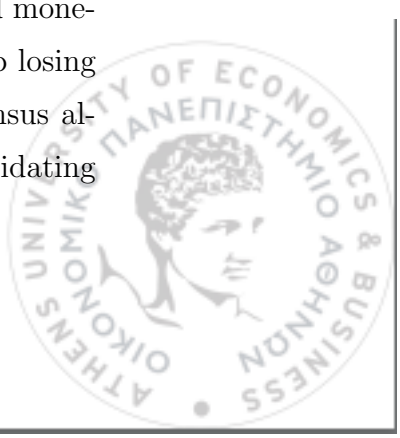
cryptographically linked to form a chain. Each block of the blockchain contains a collection of validated transactions, i.e., state transitions, organized in a data structure, and a reference to the previous block through a cryptographic hash. This cryptographic linking ensures that the ledger maintains a tamper-proof history of transactions. In most blockchains, like Bitcoin [4], the data structure used to store transactions within a block, is Merkle trees. Merkle trees organize transaction data into a tree-like structure, where each leaf node represents a transaction hash, and parent nodes represent the hashes of their child nodes. The root of the tree summarizes all transactions in the block. A data structure like that is beneficial, as it enables efficient verification, since the verification can be done without requiring access to the full block, data integrity, as any alteration in a single transaction alters the Merkle root, and compact storage, by storing only the Merkle root within the block. Emerging blockchain designs may utilize alternatives to Merkle trees, such as Directed Acyclic Graphs (DAGs) or hashgraphs, which aim to enhance



scalability and performance. The blockchain's structure ensures that once a transaction is included in a block and appended to the chain, its contents cannot be altered without invalidating subsequent blocks, enabling inherently *immutability* of actions and transactions.

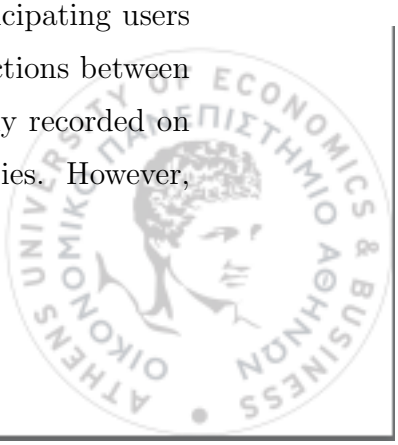
The most critical component for the operation of blockchains is the consensus mechanism, as it solves the problem of synchronizing the state of the ledger, ensuring that all participating mutually non-trusting nodes agree on the same state. Consensus mechanisms validate transactions, facilitate the addition of new blocks to the blockchain, and maintain the integrity and consistency of the distributed ledger. In decentralized networks, achieving consensus is particularly challenging, as there is not a single trusted centralized entity to decide it. Instead, consensus mechanisms establish *unique trust* between mutually non-trusting entities, ensuring *reliability* and *integrity* in the blockchain. These mechanisms operate based on a set of consensus rules that define what constitutes a valid transactions and block. The main idea behind the consensus mechanisms is to incentivize the participating users to act honestly. The nature of these incentives depends on the specific consensus algorithm used. Various consensus algorithms have been developed, tailored to the requirements of different blockchain types, with the most popular of them being the Proof of Work (PoW), Proof of Stake (PoS), Proof of Authority (PoA), and Byzantine Fault Tolerance (BFT) algorithms, such as the practical Byzantine Fault Tolerance (pBFT) algorithm [18], which are algorithms designed to solve the classical Byzantine General's problem [19] enabling consensus in distributed environments despite the presence of malicious nodes.

The PoW algorithm was introduced by the creator of Bitcoin. In these algorithms, nodes, called miners, solve computationally intensive puzzles to validate transactions and propose new blocks. This method is extremely energy-intensive in order to prevent attacks by making them economically unfeasible. Honest nodes, who validate successfully transactions and create valid blocks, are rewarded monetarily. Otherwise, dishonest nodes or those proposing invalid blocks end up losing the cost of energy used for the validation. A more energy efficient consensus algorithm is the PoS. In PoS, there is a set of validators responsible for validating



transactions and creating blocks. Anyone holding the blockchain specific cryptocurrency can become a validator. Validators take turns proposing and voting on the next valid block. Each validator's vote has different weight, depending on the size of its deposit, called stake. A validator risks losing its stake, if the block he staked on is rejected by the majority. On the other hand, for every block that is accepted by the majority, validators earn a reward proportional to their deposited stake, incentivizing honest participation and penalizing dishonest behaviors. PoA algorithms operate similarly to PoS, but replace the staking of cryptocurrencies with the validators' reputation. Validators are pre-approved and their credibility is at stake. If they do not act honestly, they end up losing their reputation and being removed from the validator set. In pBFT, all the nodes take turns to vote the next block. pBFT guarantees safety and liveness, as long as fewer than one-third of nodes are malicious. The BFT algorithms, such as pBFT, are usually used in permissioned blockchains, where participants are known and trusted or semi-trusted.

Another strong point of the blockchain technology is the support for execution of immutable computer programs, called *smart contracts*, which run deterministically within the blockchain. Smart contracts are immutable, in the sense that once deployed, their code cannot be modified or deleted. Unlike the traditional software, the only way to modify a smart contract is to deploy a new instance of it (however, there are some blockchains that allow the modification of the smart contracts after their deployment). Therefore, users can be confident that what they develop will always be executed as they developed it and it will always be respected. Furthermore, smart contracts are executed deterministically, ensuring that the outcome of their execution is identical for all participants, given the context of the transaction that initiated its execution and the state of the blockchain at the moment of the execution. Additionally, smart contracts ensure *tamper-proof* execution and enhance *transparency* by allowing all the participating users to verify the smart contract's source code and its outcomes. Interactions between users and smart contracts result in transactions that are immutably recorded on the ledger, preserving an *auditable* and secure record of all activities. However,

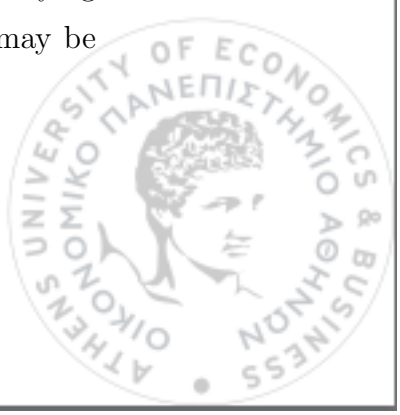


the immutability of smart contracts also introduces challenges. If there is a flaw in smart contract's logic or a bug in its source code, it will exist forever. Bugs and flaws in smart contracts can be extremely costly and easily exploited, as demonstrated by the DAO attack,[1] which occurred in June 2016 and resulted in the theft of 3.6 million Ether ($60 million in that time). For these reasons, many tools that analyze the smart contract's source code and detect possible flaws have been proposed [20].

The blockchains can be broadly categorized into two types, based on their access and governance models, even though finer distinctions can be made in some cases. The two main categories are public, permissionless blockchains and private, permissioned blockchains. Other categories include hybrid blockchains, consortium blockchains, etc. [21]. Public blockchains are open to anyone, who wishes to participate. Anyone can observe them, join the network, submit transactions, and participate on the consensus. Public blockchains are fully decentralized and typically use consensus algorithms like PoW and PoS. Two well known implementations of this type of blockchain are Bitcoin and Ethereum. On the other hand, private blockchains are restricted only to participants that have the right credentials. The identity of each participant is predefined and usually cannot change. Access and permissions are usually controlled by a central authority or a consortium. Consortium blockchains are governed by a group of organizations, providing a balance between decentralization and control. The protocols and algorithms used to achieve consensus on private blockchains are simpler and are based on solutions to the Byzantine General's Problem. A well known implementation of this type of blockchains is Hyperledger Fabric.

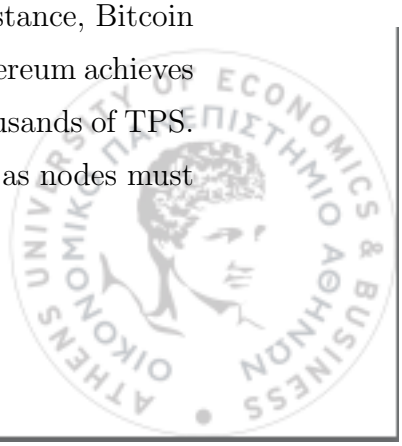
To participate in a blockchain network, independently of its type, a user must first create a cryptographic identity. This usually involves generating a public-private key pair. The private key is used for signing transactions, while the public key acts as the user's blockchain "address" and is used for verifying the signature. Depending on the blockchain type, additional certificates may be required.

[1] https://en.wikipedia.org/wiki/The_DAO



To sum up, the key characteristics and inherent properties of the blockchain technology are the following. Initially, *decentralization* is an intrinsic property of blockchain technology, enabled by the design of the blockchain's architecture and consensus mechanisms, which eliminates the need for centralized authorities. Blockchains reduce the reliance on a single, trusted entity, which usually constitutes a single point of failure. As a result, blockchains enhance *reliability* and *availability*. Furthermore, the distributed nature of blockchains makes them *resilient to faults and malicious users*, provided the majority of participants remain honest. Another key characteristic of blockchains is *immutability*. Once data, smart contracts, and transactions are recorded on the blockchain, they cannot be altered nor deleted without the consensus and awareness of the network. This ensures *integrity* of the ledger and prevents tampering. Also, the transactions are recorded on the ledger in a verifiable manner, enabling *traceability*, *auditability*, and *non-repudiation*. In addition, blockchains enhance *transparency* and offer varying levels of it, depending on the use case, i.e., public blockchains are fully transparent, while private blockchains are transparent to the participants. Finally, the consensus algorithms that blockchains rely on ensure agreement between mutually non-trusting nodes, without a single, trusted authority, ensuring enhanced *trust* within the system.

While the blockchain technology offers all the aforementioned properties, it also faces several significant challenges that limit its adoption, including scalability, performance, high costs, privacy concerns, and interoperability. Blockchains, and especially public blockchains, struggle to handle a high volume of transactions efficiently, due to the reliance on consensus mechanisms, like PoW. These scalability issues result in network congestion and high transaction fees, which can often be prohibitive for widespread adoption. Furthermore, achieving consensus across a large distributed network incurs significant communication overhead. This affects the speed of transaction validation and block propagation, making the blockchains exhibit worst performance compared to centralized systems. For instance, Bitcoin can handle approximately 7 Transactions Per Second (TPS) and Ethereum achieves around 30 TPS, while centralized systems, such as Visa, achieve thousands of TPS. Blockchains also impose high storage requirements on participants, as nodes must



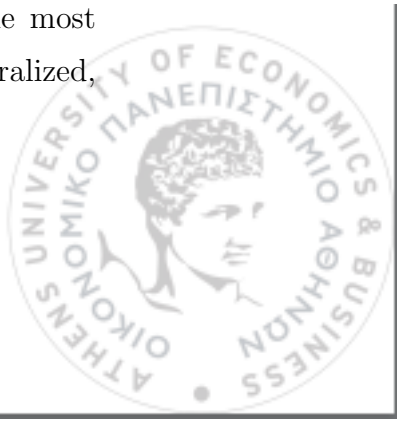
store the entire ledger. For example, the size of Ethereum's ledger currently exceed 1,220 GB.[2] This growing storage demand presents scalability concerns, especially for smaller participants with limited resources. In addition, the public nature of blockchains can conflict with privacy requirements, making them unsuitable for certain use cases, where privacy and confidentiality is critical. Finally, it is evident that a wide spectrum of diverse blockchains exists, each utilizing different technologies, e.g., consensus algorithms, and offering distinct properties, e.g., public vs private blockchains. Therefore, the "one blockchain rules them all" paradigm is far from true. Instead, securely and efficiently interconnecting these heterogeneous blockchains to leverage their unique advantages becomes increasingly important. Addressing all these challenges is essential to unlock the full potential of blockchain technology.

2.1.1 Ethereum

Ethereum [12], introduced in 2015, is one of the most popular public, permissionless blockchains. It is an open source, decentralized blockchain platform capable of executing immutable distributed applications, smart contracts. In essence, Ethereum is a deterministic state machine, consisting of a globally accessible singleton state and a virtual machine that applies changes to the state. Ethereum uses a blockchain to synchronize and store the system's state changes, alongside a cryptocurrency, called Ether (ETH). Ethereum is designed to be a general purpose programmable blockchain that runs a virtual machine, the Ethereum Virtual Machine (EVM), capable of executing code of any complexity, in a Turing complete language, such as Solidity.[3] As for the consensus mechanism, initially, Ethereum used PoW, but it transitioned to PoS, as part of the Ethereum 2.0 upgrade. In particular, the consensus mechanism the Ethereum uses now is based on Casper [22].

Unlike other popular public blockchains, such as Bitcoin, Ethereum allows users to build Decentralized Applications (DApps), making it one of the most widely used blockchains. A DApp is an application built on top of decentralized,

[2]https://ycharts.com/indicators/ethereum_chain_full_sync_data_size
[3]https://docs.soliditylang.org/



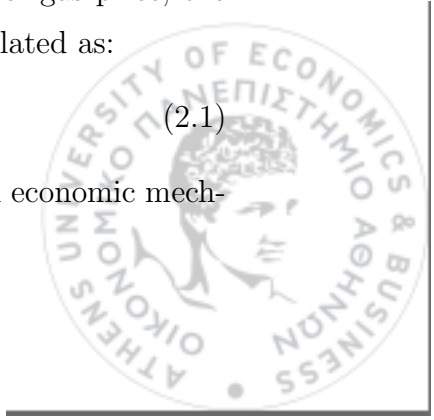
peer-to-peer services. In general, DApps represent a broader perspective than a smart contract and they might include other decentralized components, such as a decentralized storage protocol, like the InterPlanetary File System (IPFS) [23]. However, on blockchains, DApps are primarily realized as smart contracts.

Smart contracts, in Ethereum, are executed deterministically within the context of EVM. However, they operate with a limited execution scope. Specifically, they can only access their own state and information about previous blocks. Thus, they cannot communicate with the "outside world," e.g., sending HTTP requests to a server. Each smart contract has a unique address on the blockchain, which is derived from the smart contract's creation transaction, as a function of the originating account and a nonce. The deployment of a smart contract to the blockchain does not imply that the smart contract will be executed automatically in the background. Smart contracts can include functions and events. Functions are the primary way of defining the behavior and actions that a smart contract can perform. Functions can inspect and alter the blockchain's state. In contrast, events are a type of logging mechanism, primarily used for signaling changes or specific conditions within the smart contract to the outside world, without affecting the smart contract or blockchain's state. These events are stored in the transaction logs, when the smart contract executes, and they are accessible using the Ethereum blockchain.

To execute an action of a smart contract, gas is required. Gas is Ethereum's unit for measuring the computation and storage resources required to perform an action on Ethereum. Gas accounts every computation step performed by transactions and smart contract's execution. Thus, each operation performed on Ethereum, costs an amount of gas. Gas is divided into gas cost and gas price. Gas cost is the number of units of gas required to perform an operation on Ethereum, while gas price is the amount of ether, we are willing to pay per unit of gas, when we perform an operation on Ethereum. The bigger the amount of gas price, the faster the operation will be executed. Essentially, gas fee is calculated as:

$$GasFee = GasCost * GasPrice \tag{2.1}$$

Even after the transition from PoW to PoS, gas fee remains as an economic mech-



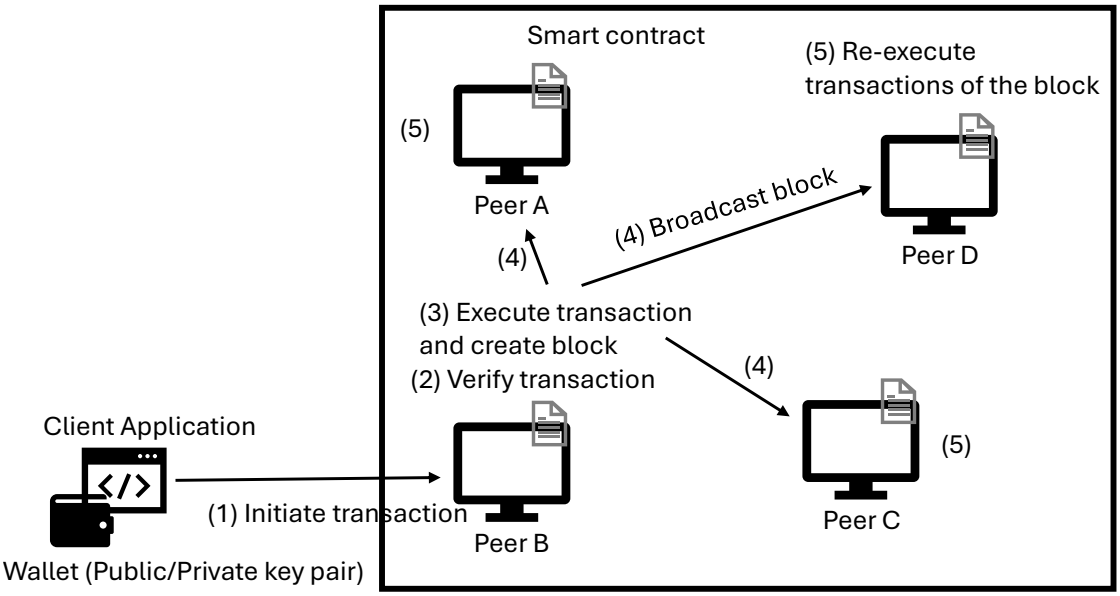


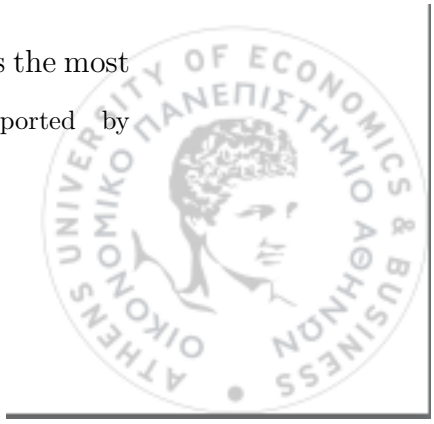
Figure 2.2: Transaction flow in Ethereum blockchain.

anism to allocate resources efficiently.

To participate in the Ethereum network and execute smart contracts, a user must own an "account" on the blockchain. An account is essentially a public-private key pair and has an address on the blockchain, which is derived from the public key of the user and is a unique identifier of the account. The private key is used for signing transactions. Furthermore, a user may own an Ethereum "full node" and interact directly with the blockchain, or he may relay his transactions through another full node that also acts as a Remote Procedure Call (RPC) server. Each design choice has its trade-offs. Maintaining a full node requires continuous network connectivity and some non-negligible storage space for storing the complete Ethereum blockchain,[4] whereas relying transactions through an RPC server entails the risk that the RPC server is offline or acts maliciously and drops messages. Thus, the RPC server is a single point of failure and trust. To avoid that we can be connected to many RPC servers from many providers.

The transaction flow in Ethereum, illustrated in Figure 2.2, follows the most

[4]The size of the Ethereum blockchain on 15 Jan. 2025 was reported by https://ycharts.com/indicators/ethereum_chain_full_sync_data_size to be 1219.20GB



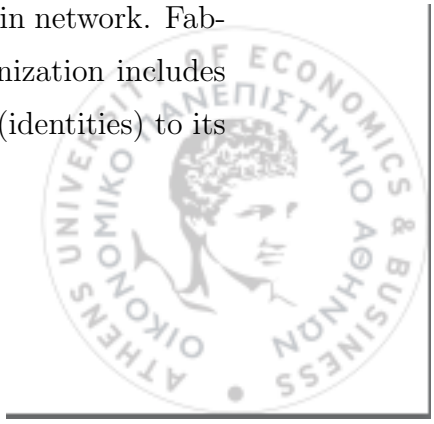
popular model, which is the *order-execute* model. The flow goes as follows. Initially, a user initiates a transaction, which includes the sender address, a recipient address, value, e.g., amount of Ethers to be transferred, gas limit and price, data field, a nonce, and the sender's digital signature. Then, the transaction is propagated to the Ethereum network, where nodes verify the transaction's validity. In particular, the valid transactions are placed into a pool of transactions, where are selected by validators, based on a priority, determined by gas fees. Then, a validator executes the transaction and the transaction is ordered in a block, which is broadcast to other nodes/validators. The other nodes re-execute all transactions in a block and if the block is valid, they append it on the ledger and the blockchain's state is updated.

Finally, a critical part of the Ethereum ecosystem is the Ethereum Improvement Proposals (EIPs) and the Ethereum Request for Comments (ERCs), which drive the evolution and standardization of the Ethereum ecosystem.[5] EIPs are formal documents that describe standards for the Ethereum platform, including core protocol specifications, client APIs, and smart contracts standards. They are vital for ensuring that changes to the Ethereum blockchain are thoroughly reviewed, tested, and agreed upon the community. ERCs represent a specific subset of EIPs, focused on defining standards for Ethereum smart contracts. These standards help developers to build interoperable DApss that can seamlessly interact with wallets and other components of the ecosystem.

2.1.2 Hyperledger Fabric

On the other hand, a popular implementation of a private, permissioned blockchain is Hyperledger Fabric (for simplicity, we will refer to is as Fabric in the remainder of the dissertation) [13], introduced by the Linux Foundation in 2016. Fabric is a consortium blockchain, meaning that a Fabric network consists of multiple organizations that come together to form the blockchain network. Fabric offers a highly modular and scalable architecture. Each organization includes its own Certificate Authority (CA) that issues X.509 certificates (identities) to its

[5] https://eips.ethereum.org/



https://doi.org/10.26219/heal.aueb.9346

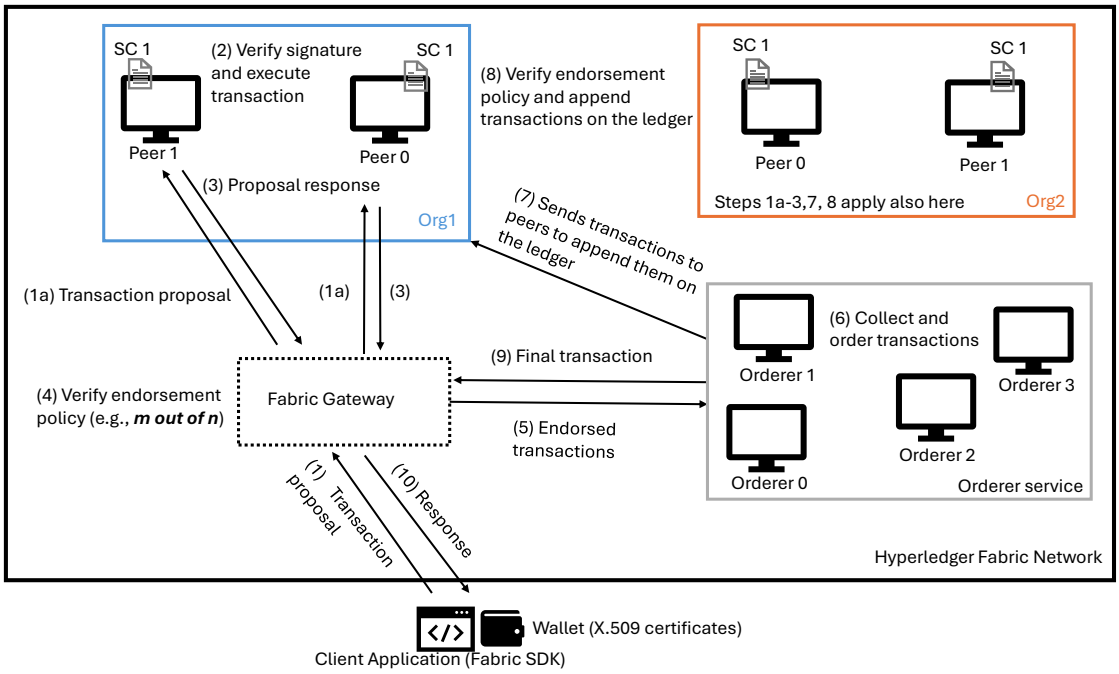
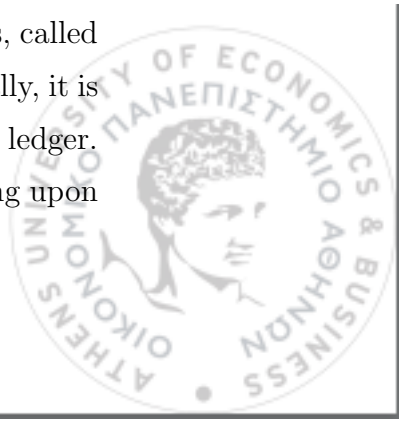


Figure 2.3: Transaction flow in Hyperledger Fabric blockchain.

members. In Fabric, membership to the network is controlled by a shared, globally defined component, called Membership Service Provider (MSP). The MSP implementation follows the Public Key Infrastructure (PKI) model, identifying which CAs are authorized to issue valid certificates and mapping the trusted CAs to organizations. Fabric, like other popular blockchains, such as Ethereum, allows the execution of smart contracts, known as chaincodes, written in any general-purpose programming language, such as JavaScript, Java, and Go. Smart contracts in Fabric, in contrast to other popular blockchains, can communicate directly with the "outside world," allowing them to send requests to external servers, call APIs, and more.

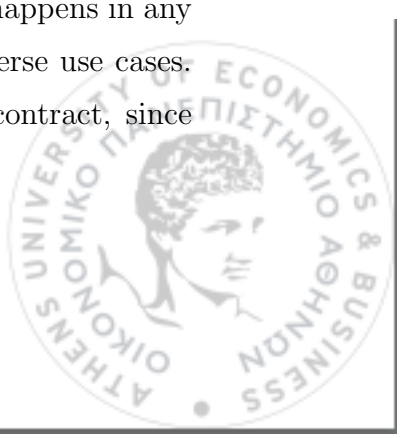
Fabric introduces a new model for transactions, called *execute-order-validate*. As illustrated in Figure 2.3, the transaction flow in Fabric is slightly different from Ethereum. Initially, the transaction is executed by all the appropriate peers, called endorsing peers, then it is validated against an *endorsement policy*, and finally, it is ordered in a block, based on the consensus protocol, and committed to the ledger. In Fabric, the consensus mechanism encompasses more than simply agreeing upon



the order of transactions and blocks. Consensus is defined as the full-circle verification of the correctness of a set of transactions and includes the usage of endorsement policies and the ordering service. In the ordering phase, the results are gathered by the orderers, which create and broadcast blocks to the network. The orderers are orchestrated through a Crash Fault Tolerant (CFT)-based consensus algorithm, such as Kafka and Raft. The endorsement policy [24] enables the selection of a set of Fabric nodes, e.g., "*n out of m*" that are required to agree on the result of a smart contract invocation for the invocation to be considered valid by the network. Endorsement policies can capture several different types of security models, from tolerating up to a certain threshold of malicious nodes to allowing state changes only by a specific set of nodes. Endorsement policies are defined per smart contract. During a transaction flow, the endorsement policy is validated twice, at the Fabric gateway and at the peer level. In particular, the Fabric gateway, which is usually a peer in the blockchain network (it is not the same peer in each transaction), when it collects all the responses from all the appropriate peers, it validates if the endorsement policy is satisfied (step 4 in Figure 2.3). Then, when orderers broadcast the block to peers to append it on the ledger, the peers are validating the endorsement policy again (step 8 in Figure 2.3).

Another difference between the Fabric blockchain and the others is the flexibility in smart contract development and deployment. In general, after the deployment of a smart contract, all the participating peers must have the same state and the same source code of that smart contract. However, a new feature of Fabric, introduced in Fabric v2.0,[6] allows the same smart contract not to be identical across the members of the network. Organizations can slightly modify a smart contract, e.g., to perform different validations in the interest of their organization. Nevertheless, a transaction will be validated and committed to the ledger, only if the required number of organizations endorse the transactions with matching results (the endorsement policy should be fulfilled), as it happens in any other case. This feature allows greater flexibility and supports diverse use cases. However, this result in non-deterministic execution of the smart contract, since

[6]https://hyperledger-fabric.readthedocs.io/en/release-2.5/whatsnew.html

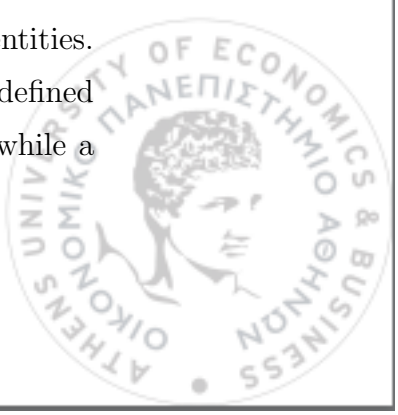


the execution output can differ across the peers. Therefore, it requires careful definition of the endorsement policies to ensure that the overall consensus process remains consistent and secure despite the non-deterministic nature of the individual executions.

2.2 Access control

Access control is a fundamental aspect of security, ensuring that only authorized users can access, modify, or interact with resources, data, and systems. Essentially, access control is a set of methods that tag, organize, and manage data and devices within a system. An efficient access control solution should be able to identify who has access to what, when, and under which conditions [10]. The main components of an access control system are policies, which define authorization requirements according to which access control is regulated, a model that provides a formal representation of access control policies, and a mechanism, which defines the low-level implementation of the access control model [25].

The access control model is the most critical part of an access control solution. The problem of designing an efficient and secure access control model is long-standing. The first access control models were proposed in the early 1970s and were designed according to the military security policies, which solve the access control problem with confidential hierarchy information [26, 27]. As technology advanced, the demands for more robust access control solutions grew, leading to the development of more sophisticated models. In particular, in 1980, the U.S. Department of Defense (DoD) published the Trusted Computer System Evaluation Criteria (TCSEC) [28], also called "Orange Book" that divides the access control models into two categories, the Discretionary Access Control (DAC) [29] and the Mandatory Access Control (MAC) [30]. In DAC model, the owner of a resource determines the permissions granted to other users. It is based on identity-based access control, where access rights are assigned to users based on their identities. On the other hand, MAC relies on a set of system rules rather than on rules defined by the owner. An example of DAC is file permissions in a Unix system, while a

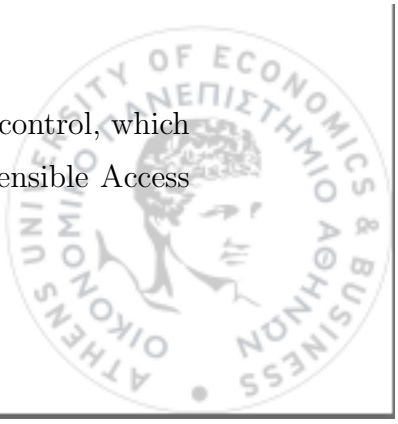


MAC example is the classified documents in a government system.

Since then, numerous access control models have been proposed to address the evolving demands and requirements of modern systems. Notably, in the late 1990s, the Role-Based Access Control (RBAC) model was introduced [31]. RBAC is an access control model for controlling user access to resources based on their assigned roles. Every user inherits the permissions associated with their roles. Another popular access control model is the Attribute-Based Access Control (ABAC), in which access rights are constrained with respect to the attributes of subjects (users), objects (resources), and actions [32]. While these models remain the most prominent access control models, emerging technologies have prompted the development of addition models. These include the Organization-Based Access Control (OrBAC) [33], Usage Control (UCON) [34], Capability-Based Access Control (CapBAC) [35], and Relationship-Based Access Control (ReBAC) [36], among others. OrBAC is an extension of RBAC model, designed for use in multi-organization environments, allowing policies to be defined and enforced at the organizational level. Namely, users are assigned roles within the context of an organization. UCON is an advanced access control solution that extends traditional access control by introducing the concept of mutability, regarding the policies (dynamic update to policies), and continuous authorization, i.e., the policies are continuously evaluated. In CapBAC, access is granted based on the possession of a capability, typically represented in the form of a token or a credential. The capability specifies the actions a user can perform on a resource. Finally, ReBAC focuses on access control decisions based on the relationships between entities rather than roles or attributes. Relationships are usually modeled as graphs, with access decisions dependent on graph traversal. Each of these models was developed to address specific challenges in access control, offering unique features and solutions tailored to the diverse and dynamic needs of modern systems.

2.2.1 A common reference architecture

In this section, we present a common architecture for access control, which was firstly introduced in the context of the first version of the eXtensible Access



https://doi.org/10.26219/heal.aueb.9346

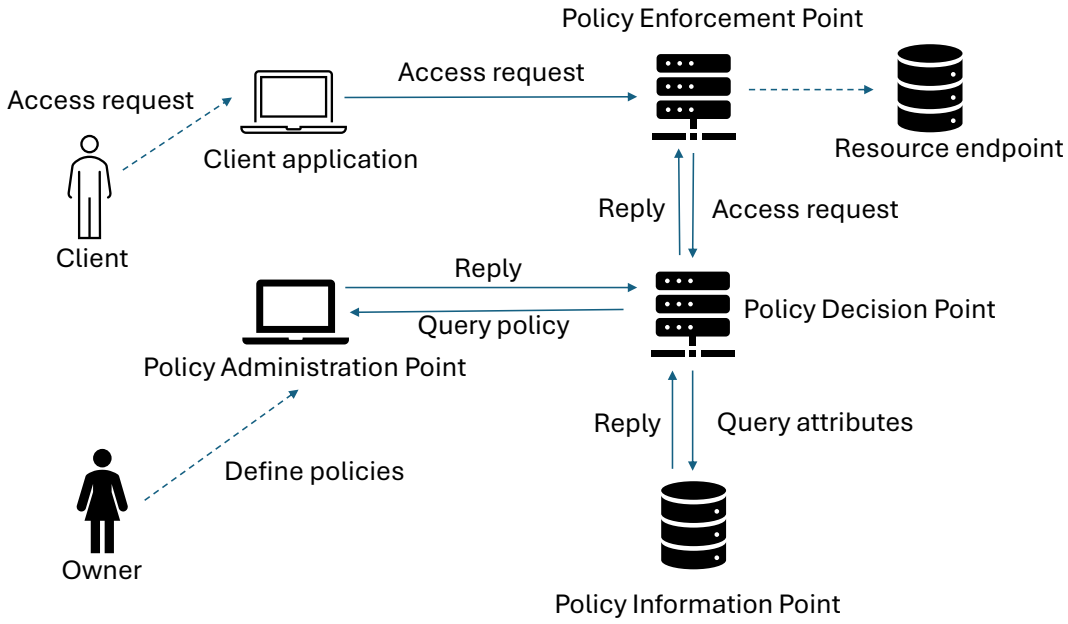
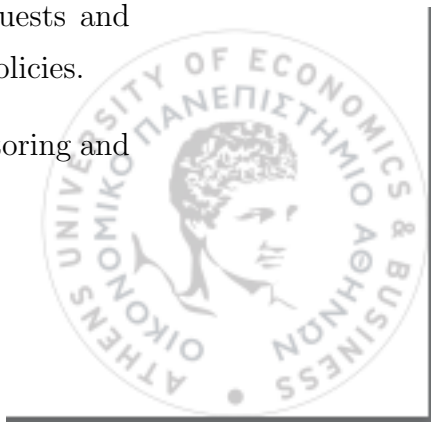


Figure 2.4: Reference access control architecture and entity interactions, based on the OASIS XACML standard.

Control Markup Language (XACML) standard by OASIS [37]. We use this architecture as a reference in the remainder of this dissertation. Although this architecture was introduced in the context of ABAC, it tries to capture the commonalities of all access control solutions that are based on policies.

The reference architecture, illustrated in Figure 2.4, is composed of the following entities:

- Client: The actor that requests access to a resource.
- Client application: Client requests access to a resource through this entity.
- Policy Enforcement Point (PEP): This entity is responsible for enforcing the access control decisions.
- PDP: This entity is responsible for evaluating access control requests and making access control decisions, based on defined access control policies.
- Policy Administration Point (PAP): This entity is responsible for storing and managing the access control policies.



- Policy Information Point (PIP): This entity is responsible for storing client's identity or other attributes and relevant information about the client.
- Owner: The actor that owns the resource and defines the access control policies.

In a nutshell, the entities interact with each other as follows. Initially, the *client*, through the *client application*, initiates an access request to access a protected resource, hosted in a *resource endpoint*. The *PEP*, which acts as a security proxy, intercepts the access request and forwards it to the *PDP*. The PDP receives the request, communicates with the *PIP* to retrieve client's identity or other attributes, roles, etc., needed for the verification of the access control policy (depending on the implemented access control model). The access control policies are stored in the *PAP* and are defined by the *owner* of the protected resource. Then, the PDP validates the retrieved attributes, based on the access control policies, and determines whether access should be granted or denied to that specific user. The access control decision is then relayed back to the PEP, which enforces it by allowing or denying the user access to the protected resource. We should note here that some of these entities may be implemented as a single entity, for instance, the PDP and the PEP can be implemented by the same application.

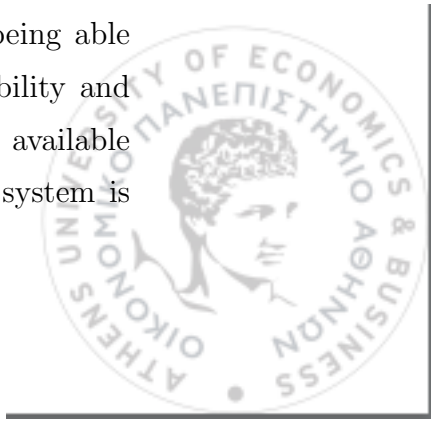
Chapter 3

IoT access control requirements and existing solutions

In this chapter, we elicit the main requirements of IoT systems with a special emphasis on access control and security. Subsequently, based on these requirements, we review and discuss existing IoT access control solutions.

3.1 IoT access control requirements

The main security requirements that any system should satisfy are confidentiality, integrity, and availability. These requirements are also applied to access control solutions. Namely, an access control system should prevent unauthorized divulgation and modification of resources, and assure access to resources only to legitimate users. Additionally, an access control solution should support privacy, which encompasses several properties. Initially, an access control solution should be transparent in the sense that users should know how their data are used, or understand who knows what about them and their data. Through anonymity or pseudoanonymity, access control should support unlinkability and unobservability, in order to ensure that a user can use a resource without others being able to observe him and link specific actions to him. Regarding, the reliability and availability requirements, an access control solution should always be available and support offline mode, making access control decisions even if the system is

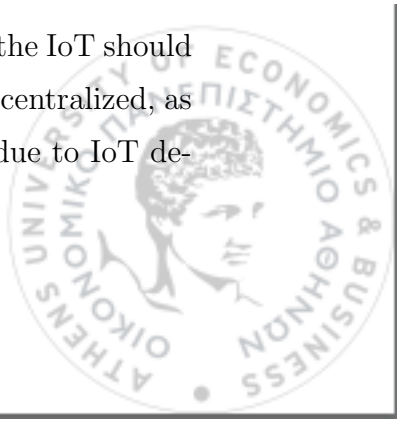


offline or the "decision maker" is absent or not connected. Besides these security requirements, an access control solution should efficiently implement revocation, delegation, and attenuation mechanisms to ensure that access control decisions will always be right, enabling confidentiality and integrity. Finally, it should be usable, meaning that a non-expert user must be able to easily express, manage, and modify access control policies.

Regarding the IoT, a commonly-accepted, standard reference architecture does not exist. The design and the architectural options vary, depending on the use case and the application that is developed. For instance, IoT devices may directly interact with users and applications, or they can be paired with a more powerful device than them, such as a gateway, which interacts with the users and the applications. Furthermore, the IoT has already been applied into a variety of application domains, including smart homes, healthcare, smart buildings, connected vehicles, smart manufacturing and Industry 4.0, smart agricultural, smart grids, and supply chains [1, 38]. Thus, it is clear from the numerous uses cases and design options of the IoT that each case has different requirements that need to be addressed. However, there are some key requirements that any IoT system should satisfy. These requirements are summarized in (architecture) scalability, interoperability – vendor compatibility, multi-agent collaboration, reliability, availability, usability, data privacy, and security [39]. In addition, the security requirements are confidentiality, integrity, reliability, availability, privacy, and usability [2, 39].

The need for efficient and secure access control in the IoT is more critical than in traditional network applications. The main reason is that in the IoT, the generated data is often private and sensitive. In addition, these data do not always lie within the administrative realm of their owner. The increasing need for more secure access control solutions in the IoT with the combination of the peculiarities of the IoT (many design and architecture options, many use cases, etc.) creates new requirements for IoT access control.

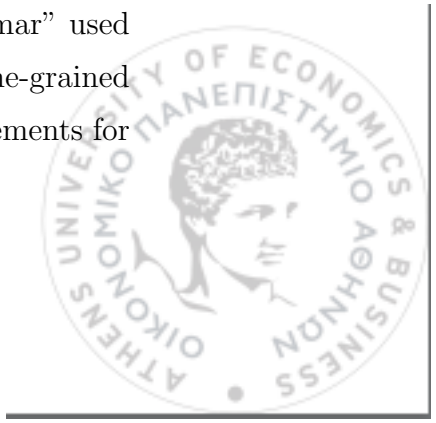
Therefore, an access control solution designed specifically for the IoT should be flexible to be easily adapted to different contexts and use cases, decentralized, as the most IoT architectures are inherently distributed, lightweight, due to IoT de-



Access control security requirements	
Non IoT-specific	IoT-specific
Privacy	Highly granular, fine-grained policies
Transparency	Context-awareness
Usability	User-driven
Confidentiality	Multi-domain policy administration
Reliability	Policy combination
Integrity	Conflict detection
Anonymity/pseudoanonymity	Conflict resolution
Availability	Lightweight implementation
Support for offline mode	Distributed architecture
Revocation, attenuation, and delegation	Flexibility

Table 3.1: IoT access control *security* requirements.

vices resource-constraints, and heterogeneous, in order to combine different devices from multiple manufacturers (vendor-agnostic). In the IoT, there are cases, where IoT devices, systems, and applications are deployed in multi-tenant environments, e.g., a business environment, where many different and potentially mutually non-trusting organizations cooperate with each other to provide services to end-users. Thus, the access control solutions should support interoperability and collaboration among different parties, which do not have any trust relationships. Furthermore, it should be user-driven, enabling users to have full and granular access control over their shared data, since these data might be sensitive. Moreover, given the direct impact of IoT systems in the real world, auditability becomes crucial, as it can be the last resort for recovering from security incidents. Due to the IoT's nature, access control should also support context-awareness regarding its decisions, e.g., allow a user to open the lights of his house only if he is inside the house. Additionally, it should be highly granular, in the sense that the "grammar" used to formulate access control policies should be expressive enough and fine-grained in order not be be limited to fixed access control policies. All the requirements for IoT access control solutions are summarized in Table 3.1.

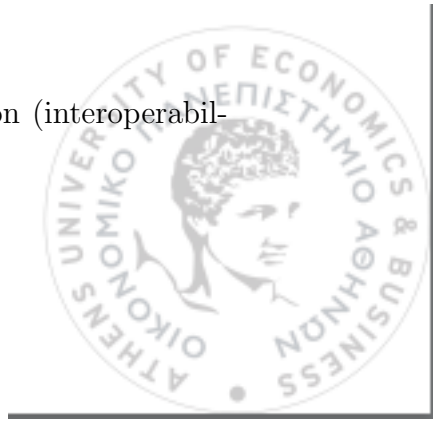


3.2 IoT access control solutions

Traditional access control models and solutions are deemed inappropriate for IoT environments, as they do not address the IoT-specific requirements for access control. First and foremost, most access control solutions are primarily Web-based. Consequently, adopting traditional access control in the IoT typically necessitates the adoption of the WoT approach [17], which aims to integrate IoT devices into the Web through standard Web protocols and technologies. Additionally, most of the traditional access control solutions are inappropriate for defining fine-grained access control policies and permissions based on the context and dynamics of IoT environments, as they use more rigid policies that do not adjust dynamically to such varied conditions. For instance, in RBAC, reaching consensus regarding the definition of a shared role across different applications, systems, and domains is extremely challenging, impacting interoperability. Furthermore, most access control solutions are not lightweight and do not consider the resource constraints of IoT devices, as they often involve complex cryptographic techniques. For instance, ABAC solutions are way too complex, particularly due to the use of XACML [40], which is commonly used in ABAC solutions for representing attributes, their definition, and expressing attribute-based authorization requests. This complexity also deters users, making ABAC solutions less user-friendly. Finally, these solutions often lack a decentralized architecture, with critical components, like the PDP, being centralized and inherently trusted (trust is assumed to be valid under all circumstances).

Thus, we observe that existing access control solutions not only fail to fulfill the IoT's requirements, but also there are many open issues that remain to be addressed [39, 41, 42].These can be summarized as follows:

- Fine-grained access policies
- Usability and user-driven
- Multi-domain policy administration and policy combination (interoperability)



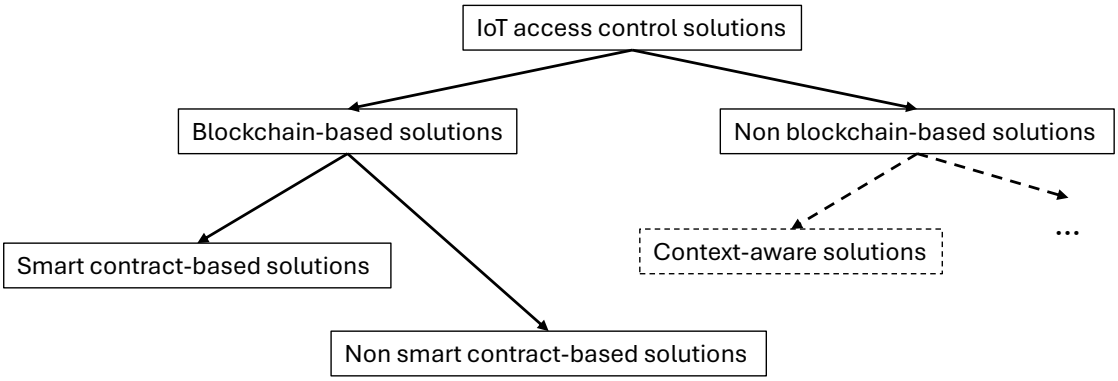


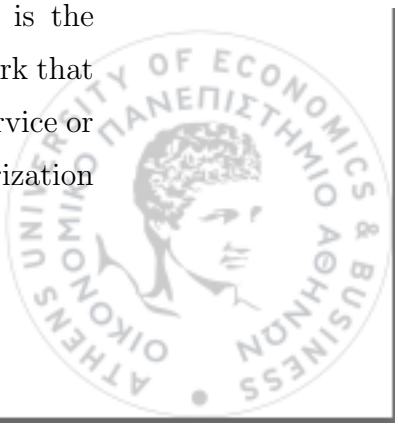
Figure 3.1: A categorization of IoT access control.

- Conflict detection and resolution in multi-party environments (auditability)
- Availability
- Lightweight implementation and distributed architecture

To address the open issues and meet all requirements for IoT access control, new access control solutions tailored to the IoT have been proposed. We categorize the research interest into two main categories of works. The first considers IoT access control solutions using advanced cryptographic techniques and other state of the art technologies, and the second category focuses on access control solutions backed by blockchains. The categorization, we made, of the existing IoT access control solutions is appeared in Figure 3.1. In the following subsections, we present an overview of the state of the art research papers in IoT access control, based on this categorization. For every work, we present its main contributions and the addressed requirements, regarding the IoT.

3.2.1 Non blockchain-based IoT access control

One of the most widely used authorization protocols in the IoT is the OAuth 2.0 protocol [16]. OAuth 2.0 is a token-based authorization framework that enables a third-party application or client to obtain access to a protected service or resource, hosted on a resource server. The access is managed by an authorization



https://doi.org/10.26219/heal.aueb.9346

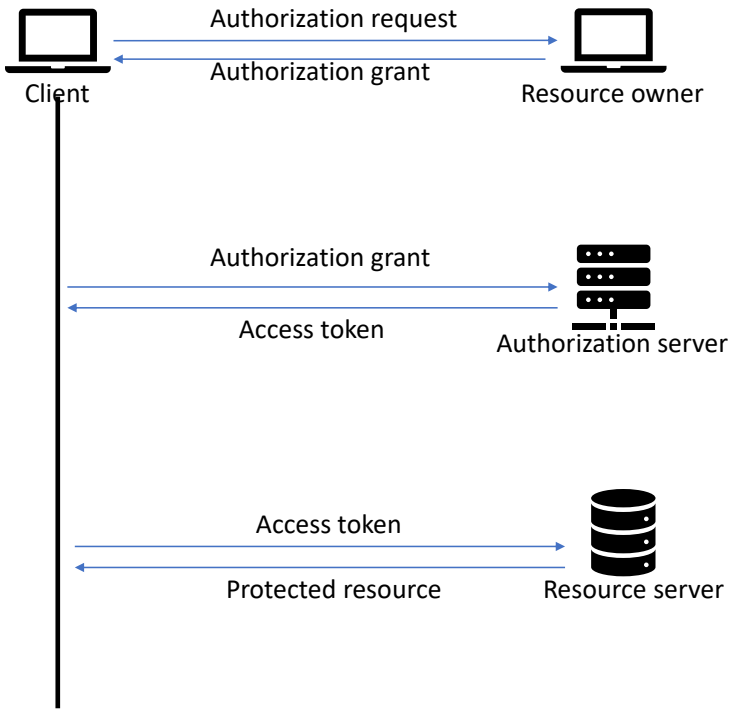
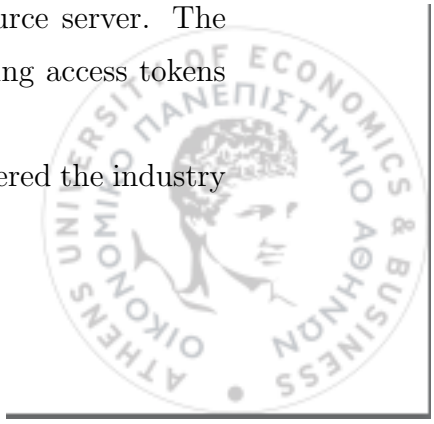


Figure 3.2: OAuth 2.0 entities and interactions.

server, based on the consent of the resource owner. All OAuth 2.0 protocol flows result in the creation of an ACT, which is used by the users to request access to the protected resource. OAuth 2.0 systems are composed of the following entities. A *resource owner*, who is capable of granting access to a protected resource; the *resource server*, which hosts the protected resource and is capable of accepting and responding to access requests; the *client*, who makes requests for access; and finally, the *authorization server*, which issues *ACTs* to clients after successfully authenticating the resource owner and obtaining authorization. The flow and interactions between these entities are depicted in Figure 3.2. As it can be seen, a client first requests an authorization grant from the resource owner, then it uses this grant to obtain an ACT from the authorization server, and finally, it uses the issued ACT to access the protected resource stored in the resource server. The semantics, as well as the mechanisms for generating and validating access tokens and grants are transparent to the OAuth 2.0 protocol.

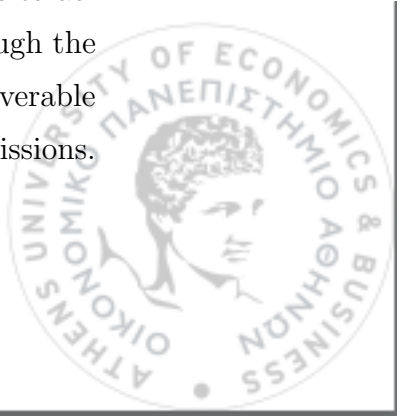
OAuth 2.0 has received widespread adoption and is considered the industry



standard protocol for authorization, as it enables delegation, interoperability, it enhances end-user security, and it facilitates access control management. Due to its intriguing properties, is is being used in environments with higher security requirements than initially considered, such as IoT systems. However, despite its adoption and popularity, OAuth 2.0 presents several shortcomings, with the most important of them being that it relies on centralized servers. The centralization of OAuth 2.0 leads to two main problems. First, a centralized service constitutes a single point for attacks. Furthermore, the administrator of this single entity, who is usually the owner, has a complete view of data and permissions.

An alternative to centralized systems, such as OAuth 2.0, is presented by Andersen et al. [43], who introduce WAVE, an authorization framework that offers decentralized trust and allows any participant in the system to autonomously delegate a portion of their permissions. As a use case, the authors consider a complex case of a smart campus, owned by a property manager, that includes multiple smart buildings, some of which are leased out to tenants. Within each campus, the property manager, considered the "owner" of the resources associated with the buildings, should delegate permissions to individual building managers, who in turn must delegate permissions to tenants, enabling them to control the resources of the buildings they lease. Building managers can also grant ephemeral permissions on subsets of the buildings to other entities, such as visitors. This use case covers a wide range of IoT applications, from small residential buildings to larger structures with many administrative points.

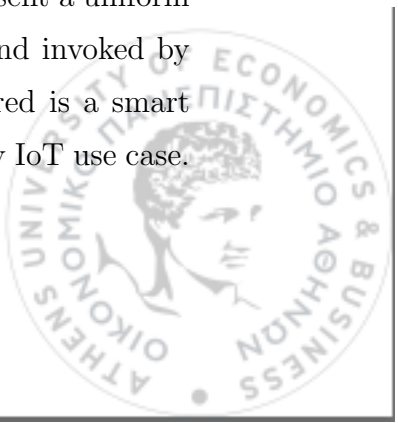
WAVE's functionality is divided into 3 layers. The first layer, the authorization layer, uses a graph-based approach to represent permissions. The global authorization graph consists of entities, which are bundles of public and private keys, and attestations, which are permission grants between entities. When a user wishes to grant permission, he constructs an attestation signed by the granting entity, containing a policy describing the permissions. When a client wants to access a service or a resource, he has to present a proof, which is a path through the graph from authority to the prover. The second layer uses Reverse-Discoverable Encryption (RDE) to encrypt attestations, ensuring the privacy of permissions.



This functionality is transparent to users. In a nutshell, in each encrypted attestation, the decryption key of the previous attestation is included. To achieve RDE, the authors use Wildcard Identity-based Encryption (WIBE). The last layer is a scalable untrusted storage. The entities (public keys) and RDE ciphertexts (attestations) are stored into the scalable untrusted storage. Again, this layer is transparent to users. Clients operate only at the level of granting permissions, creating proofs, and verifying them. The scalable untrusted storage has similar guarantees as a blockchain. Furthermore, it supports proof of non-existence, which allows revocation and efficient auditing.

One of the main issues with traditional access control solutions that authors identify and address is the reliance on a central trusted party and the problems that arise if this party is compromised. Additionally, the main requirements that are met by the aforementioned design can be summarized in the following: no reliance on a central trust, transitive fine-grained delegation and revocation, protected permissions, decentralized verification, no ordering constraints, and support for offline participants. WAVE meets all these requirements simultaneously, for this complex IoT use case, offering same performance metrics as the ones offered by traditional centralized systems. However, this work does not involve context-awareness and does not support the collaboration of many entities for the access control.

On the other hand, there are works that involve context-awareness on the access control procedure, particularly in decision and enforcement actions. Schuster et al. [44] identify that in the IoT, situation tracking is entangled with the enforcement of access control policies. Thus, they propose a new approach in IoT access control towards that direction. Specifically, they introduce Environmental Situation Oracles (ESOs) into the IoT ecosystem, to enforce situational constraints within IoT access control frameworks. ESOs can be deployed at any layer of the IoT stack, where access control is applied. They encapsulate the tracking of environmental conditions relevant to access control enforcement and present a uniform interface that can be incorporated into any access control policy and invoked by any monitor within a given IoT ecosystem. The use case considered is a smart home, however, the proposed solution can be easily integrated in any IoT use case.

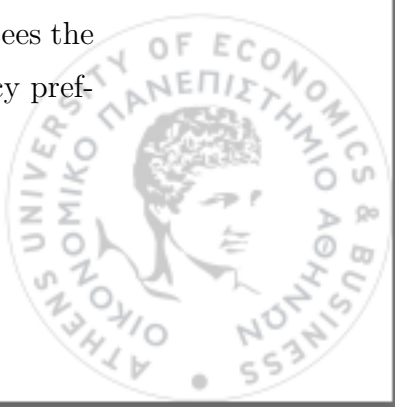


Using situational conditions in access control policies is a well-known approach and many solutions have been proposed and integrated on well-known IoT frameworks, such as Samsung's SmartThings.[1] These frameworks typically track the user's location, usually from the user's smartphone, to determine their location. However, the existing solutions have several problems and limitations. First of all, tracking the environmental situations directly from the user's smartphone can lead to over-privileging in IoT applications and privacy violations. If the user has installed many IoT frameworks, from different vendors, all of them gain the ability to track and determine his location. Moreover, the existing frameworks are unable to enforce common policies, such as "allow access only when user is at home." Furthermore, GPS applications in smartphones are not completely accurate; they cannot, for example, distinct whether the user is in the living room or in his bedroom. All these issues result in over-privileged, redundant, inconsistent, and inflexible implementations.

In contrast, ESOs enable two-way obliviousness between access control policies and situation trackers. A single ESO can serve multiple access control frameworks across the IoT ecosystem. This reduces inefficiency, supports consistent enforcement of common policies, and minimizes over-privileging. Furthermore, the access control frameworks, are oblivious of the details and implementation of the ESO. Finally, ESOs enforce the principle of the least privilege. Thus, IoT frameworks have access only to abstract data, e.g., "the user is at home," rather than to raw data, such as the user's smartphone GPS coordinates. An ESO can be maintained by a trusted party or even by one of the client IoT frameworks (e.g., SmartThings). However, in both cases, other IoT frameworks no longer need access privileges for the raw information.

Another work with a design very similar to ESOs, which offers the same benefits, is presented by Chi et al. [45]. The authors recognize that in IoT applications, large amounts of (sensitive) data are transmitted to IoT platforms. To address this issue, they propose a data minimization approach that guarantees the correctness and completeness of home automation, satisfies personal privacy pref-

[1] https://www.samsung.com/us/smartthings/



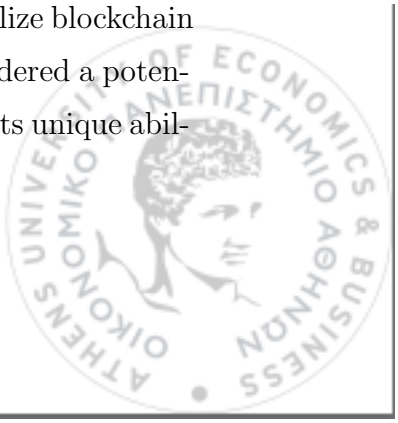
erences, and reduces the amount of data sent to IoT platforms. To achieve that, they introduce PFirewall, a customizable data flow control system that enhances the privacy of IoT users. PFirewall is particularly aimed at protecting the privacy of smart home owners. They have tested their solution in Samsung's SmartThings framework, and on openHAB.[2] However, the authors claim that their solution can also be applied in other IoT use cases, such as smart offices, hospitals, and factories.

PFirewall, similar to ESOs, acts as an intermediate trusted entity for IoT systems, positioned between IoT devices and IoT platforms (or hubs and gateways). PFirewall includes three main modules, the device connector, which communicates directly with IoT devices, the platform connector that interacts with the IoT platforms, and the policy-based data filter, located between the device and platform connectors. It filters the sensitive data produced from IoT devices and sent to IoT platforms, based on policies. The policy-based data filter itself has three components, the policy generator that generates policies either automatically or user-defined, a module that checks if a user-defined policy conflicts with existing data-minimization policies and reports results back to users, and the last component interprets and executes all policies. Their evaluation on the two aforementioned frameworks shows that PFirewall significantly reduce sensitive data leakage, without negatively impacting home automation, and generally minimizes the attack surface concerning attacks aimed at compromising user privacy. These works do not propose new access control solutions for the IoT; instead, they introduce mechanisms to incorporate context-awareness into the access control procedures, regardless of the access control solution. Therefore, they do not address any of the open challenges or IoT access control requirements other than context-awareness.

3.2.2 Blockchain-based IoT access control

In this section, we explore IoT access control solutions that utilize blockchain technology as a core component of their systems. Blockchain is considered a potential solutions for enabling access control in IoT environments, due to its unique abil-

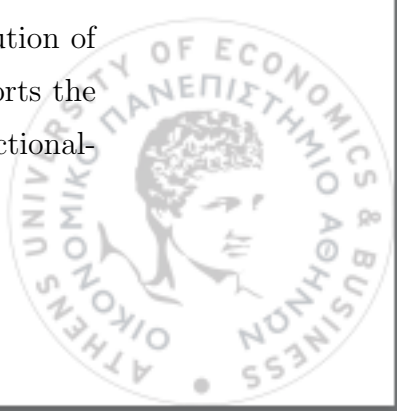
[2]https://www.openhab.org/



ity to establish trust, transparency, auditability, availability, reliability, integrity, and non-repudiation [46]. Research on blockchain-based IoT access control can be broadly categorized into three types. The first category involves the use of blockchain technology and smart contracts to develop complete IoT systems. The second leverages smart contracts to represent components of the access control architecture, such as PDPs, PEPs, etc. The third category utilizes blockchain and smart contracts to implement and represent auxiliary entities of access control, e.g., ACTs, capabilities, and related mechanisms.

Ouaddah et al. [47] present FairAccess, a fully decentralized, pseudonymous, and privacy-preserving authorization management framework that enables users to own and control their data. FairAccess introduces new types of transactions in the Bitcoin blockchain that are used to grant, get, and delegate access. Although the authors demonstrate their solution in a smart home use case, they claim that it is applicable to a variety of IoT use cases, such as transportation and healthcare. The main entities of FairAccess include resource owners and requesters. Each of these entities interacts through transactions. Furthermore, each entity owns a wallet, which holds his credentials, his addresses, and the transactions related to them. Moreover, every user and his resources have a public cryptographic identity, which in essence is an address. The authors have modified Bitcoin transactions to transfer ACTs, which are encrypted with the public key of the requesting party, instead of transferring bitcoins. The types of the modified transactions are grant access, where a resource owner defines an access policy and generates a new ACT, get access transaction, where a requester "spends" the ACT, and delegate access, where a requester delegates the access to another entity. FairAccess enables transparency and demonstrates a decentralized architecture. Furthermore, it is user-driven, allowing users to be masters of their own data with full and granular control over the shared resources.

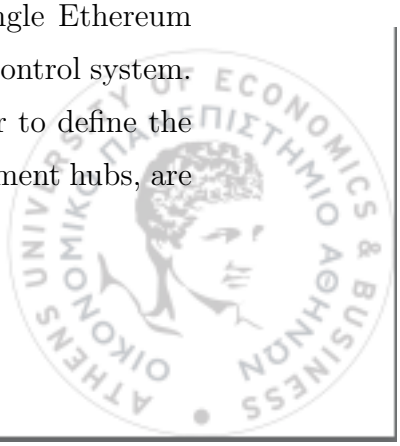
FairAccess utilizes the Bitcoin blockchain, which presents poor performance and does not support the execution of smart contracts, but only the execution of scripts with limited functionality. In contrast, Ethereum blockchain supports the execution of smart contracts, making it easier to implement complex functional-



ities, like those needed in access control. Hammi et al. [48] leverage Ethereum's capabilities for parallel execution of smart contracts to propose a decentralized system, called bubbles of trust, which ensures robust identification and authentication of IoT devices, protects data integrity, and ensures availability. To achieve all that, the authors rely on the security properties provided by the blockchain technology. Authors claim that their solution can be applied to the majority of IoT use cases, however, they evaluate their approach in four use cases; smart home, waste management, smart factory, and smart road radar.

The primary goal of the system is to create secure virtual zones of IoT devices in an IoT environment. Each IoT device communicates only with other IoT devices within its zone and considers all other devices as malicious. The authors use an Ethereum smart contract to group IoT devices in "bubbles" of trust. In a nutshell, the system works as follows. Each "bubble" is managed by a master, analogous to a CA, that decides which device can join the bubble. To join a bubble, an IoT device must present a lightweight certificate, signed by the master, to the smart contract. Once part of the bubble, an IoT device can communicate securely with the other members of the same bubble, through the smart contract, which verifies whether the sending and receiving IoT devices belong to the same bubble. This design, leveraging the blockchain technology, ensures mutual authentication and message integrity, identification, non-repudiation, and scalability. Furthermore, it ensures protection against sybil attacks, spoofing attacks, message substitution attacks, message reply attacks, and Distributed Denial of Service (DDos) attacks. However, since the solution relies on Ethereum blockchain, it introduces monetary costs, which may be not negligible.

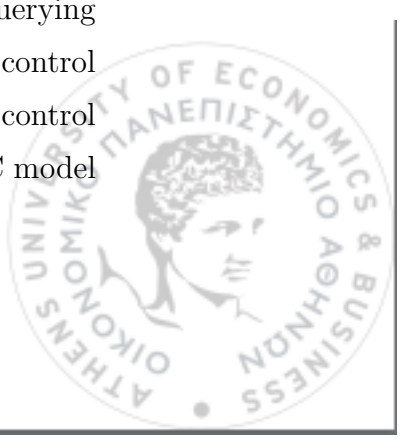
Other works utilize smart contracts to implement some of the access control actions within the blockchain. Novo [49] proposes a new decentralized access management system, where access control information is stored and distributed using blockchain technology. The proposed solution involves a single Ethereum smart contract that defines all the operations allowed in the access control system. Entities, called managers, interact with the smart contract in order to define the access control policies of the system. Gateway nodes, called management hubs, are



responsible for handling resource requests by taking into consideration the policies stored in the blockchain. To do so, they query the smart contract to fetch the access control policies and based on them decides on access. Thus, the proposed solution uses a smart contract just to act as PAP and PIP.

In contrast, other works utilize smart contracts for the decision processes. Zhang et al. [50] propose a smart contract-based access control framework that achieves distributed and trustworthy access control for IoT systems. The authors demonstrate their solution in a smart home use case, but assert that their approach is generic enough to be deployed in a variety of IoT use cases. In their construction, the actions a "subject" can perform on an "object", as well as the corresponding permissions are recorded in an "access control smart contract". A "register smart contract" is responsible for maintaining a mapping from subject-object identifier pairs to access control contract addresses. An IoT gateway handles resource requests and it is responsible for enforcing the access control policies defined in the corresponding access control smart contract. Additionally, they introduce a judge smart contract, which facilitates the dynamic validation by receiving misbehavior reports, judging the misbehavior, and returning the corresponding penalty. Finally, the register smart contract, which registers the information of the access control and judging contracts, and provides functionality for managing them. In this work, authors just implement an identity-based access control solution on the blockchain, without proposing any novel mechanism utilizing the blockchain's properties.

In this direction is also the work presented later by Liu et al. [51], who propose an access control system for the IoT, name Fabric-IoT, with the difference that Fabric-IoT is based on the Fabric blockchain, rather than on Ethereum. The system is similarly composed of three types of smart contracts, a device smart contract, a policy smart contract, and an access control smart contract. The device smart contract provides methods for storing the URL of resources and querying them. The policy smart contract provides functions for managing access control policies, while the access control smart contract implements the access control methods. Essentially, the authors split the main functionality of the ABAC model



https://doi.org/10.26219/heal.aueb.9346

and implement it across the two smart contracts related to access control policies, decisions, and enforcement. The system ensures data consistency, distributed architecture, privacy, flexibility, fine-grained and dynamic access management, and presents better performance, than solutions that use public blockchains.

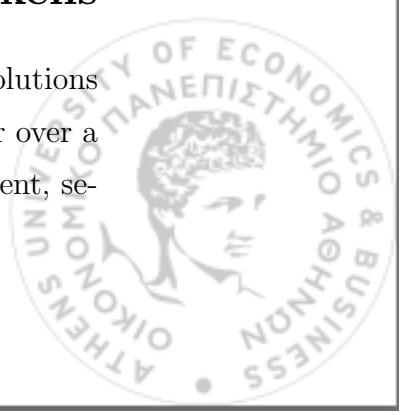
Chapter 4

New access control solutions for IoT architectures

In this chapter, we present our proposed access control solutions for IoT systems, which leverage blockchain technology to enhance security. As IoT environments become ubiquitous and increasingly complex, conventional centralized access control solutions encounter various challenges, underscoring the need for more decentralized and secure approaches. Our access control solutions are designed to harness the intrinsic properties of blockchain technology, offering robust access control mechanisms tailored to the needs and requirements of IoT environments. By utilizing smart contracts and blockchains, we manage to decentralize critical entities of the access control architecture (Figure 2.4), especially the PDP, which traditionally acts as central authority in making access control decisions. By leveraging blockchain technology, we can distribute the decision-making process, thereby enhancing transparency, trust, and resilience against attacks, while reducing bottlenecks associated with centralized systems.

4.1 Token-based access control – ERC-20 tokens

Many legacy access control mechanisms implement access control solutions using "tokens" that indicate the capabilities and the access rights of a user over a resource. However, particularly in the context of the IoT, token management, se-



curity, and semantics interpretation cannot be trivially implemented. Utilizing the inherent properties of blockchain technology, such as transparency, decentralization, and immutability, can significantly enhance the security of ACTs and reduce the risks associated with centralized control. For this reason, we leverage the capability of the Ethereum blockchain, which supports the development of custom tokens, to design and implement novel forms of ACTs. This capability allows for the creation of token-based access control solutions specifically tailored to IoT environments, ensuring more robust, efficient, and secure access control solutions, providing better token management, such as efficient revocation, delegation, and fair exchange.

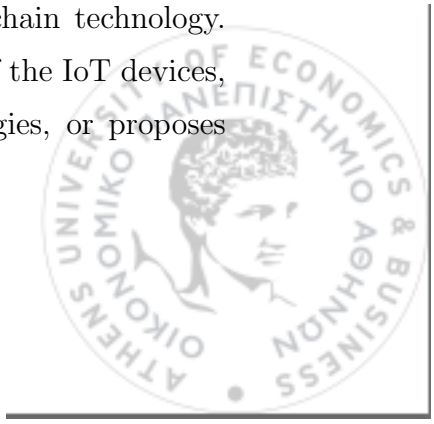
Ethereum has specified an open "token standard," called ERC-20 [14]. This standard defines some functions that a smart contract should implement in order to be able to create and manage custom fungible tokens, i.e., a new type of coin on the Ethereum blockchain. Table 4.1 describes the main functions defined in ERC-20 token standard. The two transfer functions, when invoked, each generates an event, called `Transfer`. The event has three attributes; the *from* address, the *to* address, and the *value* of transferred tokens. Furthermore, the function `approve` generates an event, called `Approval`, which has three attributes; the *owner* address, the *spender* address, and the *value* of the tokens. Many popular Ethereum wallets, such as Metamask,[1] can handle and manage ERC-20 tokens.

To validate the feasibility of ERC-20 tokens as ACTs and show their potentials and benefits, we demonstrate it into two real life IoT use cases [52, 53].

4.1.1 Large scale IoT control system

In this work, we take advantage of the distributed nature of blockchains to build a large scale IoT control system that supports novel token-based access control [52]. We argue that existing approaches lack realism and do not take full advantage of the possibilities and capabilities of the blockchain technology. Indeed, related work in this area either neglects the limitations of the IoT devices, or tries to introduce new, hard to deploy, blockchain technologies, or proposes

[1] https://metamask.io/

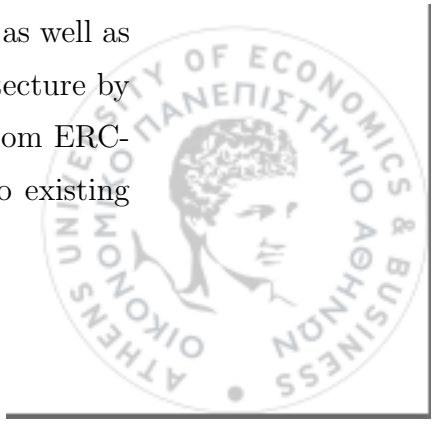


Name	**Purpose**
name()	Returns the name of the token
symbol()	Returns the symbol of the token
decimals()	Returns the number of decimals the token uses
totalSupply()	Returns the total token supply
balanceOf(address)	Returns the account balance
transfer(to, value)	Transfers tokens to an address
transferFrom(from, to, value)	Transfers tokens from one address to another
approve(spender, value)	Allows spender to withdraw from your account, multiple times, up to the value
allowance(owner, spender)	Returns the amount, which spender is still allowed to withdraw from owner

Table 4.1: ERC-20 token standard main functions.

(unrealistic) modifications to existing blockchain architectures. Similarly, it does not create new solutions using the new features provided by this novel paradigm, instead it tries to merely transfer existing techniques into the new environment. Although, the latter approach may seem to have some value, it turns out that many of the existing solutions do not consider the particularities of the blockchain technology. For example, public blockchains cannot be used for storing secret and sensitive information, nevertheless, many proposals use public blockchains for storing private user data and business roles and structures.

We are concerned with the secure operation of large IoT deployments and our work is based on the observation that many blockchain solutions can be used as event-based systems. With this in mind, we design a blockchain-based architecture that allows users to control IoT devices organized in "groups," e.g., turn on the lights of a smart city. Our architecture, which is built using the Ethereum blockchain, considers the limitations and capabilities of the IoT devices, as well as the properties of the blockchain technology. Then, we secure this architecture by adding a token-based access control solution, using the Ethereum's custom ERC-20 tokens. This approach has some significant advantages compared to existing



token-based approaches, with the most important being that it is impossible for a user to transfer his security tokens to another user.

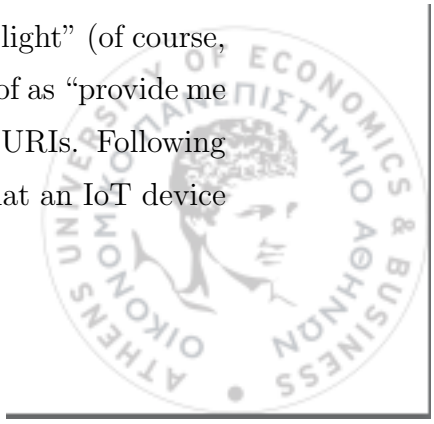
A blockchain-based IoT architecture

We now describe our Ethereum-based IoT architecture, which is composed of the following entities:

- The blockchain infrastructure
- A smart contract that generates events
- Full nodes that also act as RPC servers
- IoT devices
- IoT gateways
- Clients that want to control the IoT devices

Clients and IoT gateways are in control of an Ethereum blockchain wallet and they can be Ethereum full nodes themselves or they can be connected to the blockchain through another full node, acting as an RPC server, as we have already mentioned. In our design, we consider the latter design option. A client does not have to interact directly with an IoT gateway (or IoT device), instead all interactions take place through the blockchain. Furthermore, IoT devices are connected to IoT gateways. From a high-level perspective our architecture is designed as follows. All device operations are mapped to a function in a smart contract; every time a client invokes a function (properly) the smart contract generates the corresponding blockchain events. These events are received by interested IoT gateways and eventually result in an operation in the appropriate IoT devices.

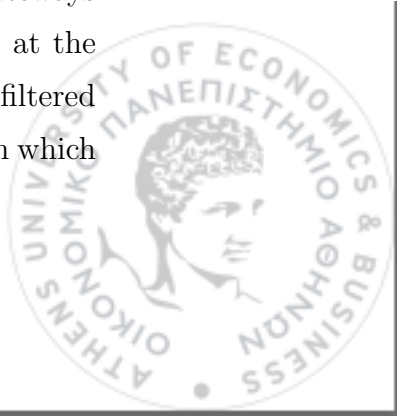
As IoT devices, we consider actuators and we assume that an actuation process can be invoked through an "operation," e.g., "turn on the light" (of course, sensors can easily be handled and their operations can be thought of as "provide me your current data"). Furthermore, IoT devices are identified by URIs. Following the semantics of CoAP group communication [54], we consider that an IoT device



may have multiple URIs and a URI may correspond to multiple devices. The semantics of a URI are application specific, for instance, they may indicate the physical location of a device, e.g., "building6/floor3/room2." An IoT gateway knows the URIs and the supported operations of the devices attached to it, e.g., by using an out-of-band configuration mechanism, or by using a service discovery protocol, such as CoRE Resource Directory [55].

The main component of our system is a smart contract, whose address is considered well-known. When invoked, this smart contract generates the appropriate events. An Ethereum event has a name and some attributes. A RPC client may request to watch the events produced by a smart contract, by specifying the event name and optionally a filter over (a maximum of three) "indexable" attributes. In our architecture, we consider a generic event name, i.e., `Operation`, and we specify for each event two attributes: an indexable called *OPCode* that encodes the desired operation and a second one, also indexable, called *URIResource* that corresponds to the URI of the device(s) in which OPCode is applied (in order to be more precise, since Ethereum does not allow strings to be indexable, the URIResource attribute holds the hash of the URI). IoT gateways register to their RPC server to watch the event Operation of our smart contract and (optionally) specify filters on the event's attributes.

Clients simply interact with the smart contract and invoke the appropriate functions. The main function of our smart contract is called `invokeOperation`. This function accepts two input parameters: an *OPCode* and a *URIResource*, and generates an `Operation` event, whose attributes have the same value as the function call parameters. Eventually, this event reaches the IoT gateways that are "watching" for it. In return, each IoT gateway invokes the corresponding operation at the IoT devices that are associated with the specified URI. An overview of our approach is illustrated in Figure 4.1. In this figure, there is a client, two IoT gateways, and two IoT devices attached to each gateway. One of the gateways starts "watching" for the **Operation** event of the smart contract located at the address "0xa3c1" (step 1). Furthermore, the gateway requests events to be filtered based on their *OPCode* and specifies that it wants to watch only for events in which



https://doi.org/10.26219/heal.aueb.9346

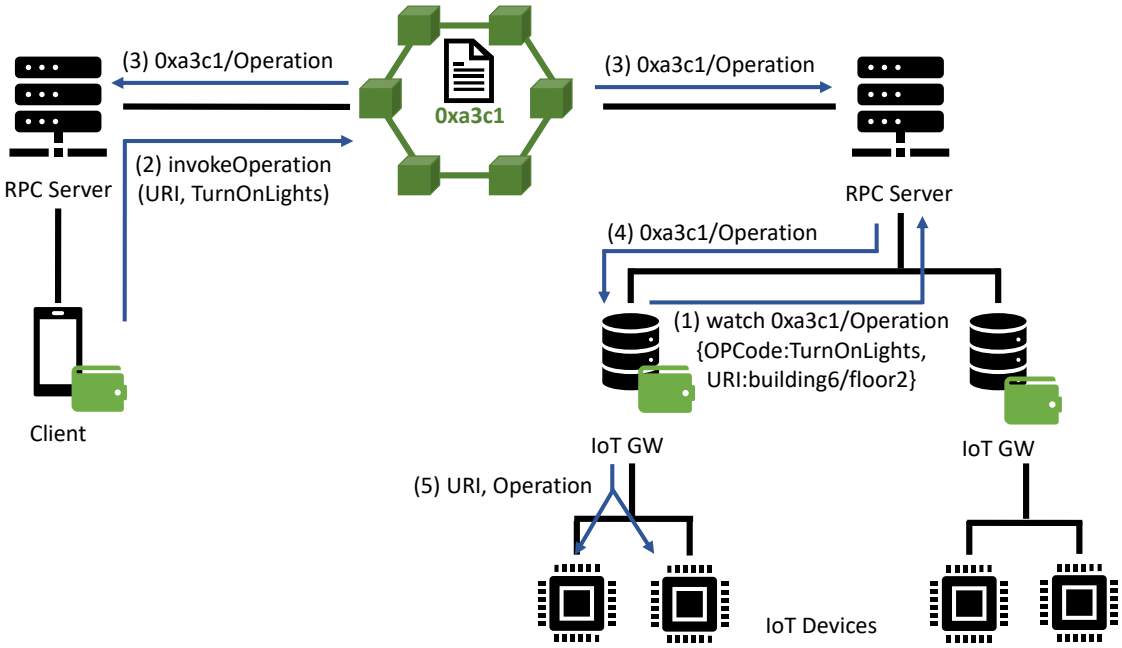
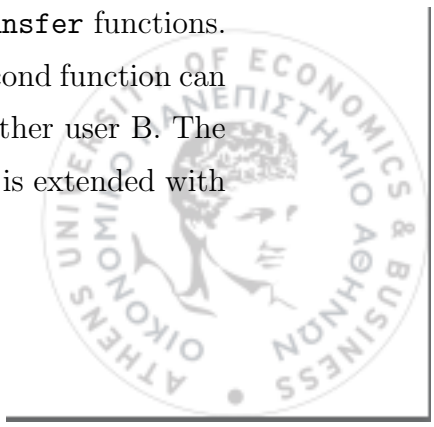


Figure 4.1: A blockchain-based large scale IoT architecture.

OPCode is "TurnOnLights". At some point, a client invokes the `invokeOperation` function of the smart contract. It uses as URIResource, a URI that matches the IoT devices of the aforementioned gateway and as OPCode "TurnOnLights" (step 2). This transaction results in the creation of an event, which is propagated to all full nodes (step 3). Furthermore, it is transmitted to the IoT gateways that are watching for such events, including our example gateway (step 4). The gateway extracts the URIResource of the event and checks if it matches any of the IoT devices attached to it. Since this is the case in our example, the IoT gateway executes the corresponding operation on the appropriate devices (step 5).

Token-based Access Control using Smart Contracts

The core of our token-based access control solution is built using two of the functions, presented in Table 4.1, namely the `balanceOf` and `transfer` functions. The first function returns the token balance of a user, while the second function can be invoked by a user A to transfer some tokens (he owns) to another user B. The smart contract of the architecture defined in the previous section is extended with

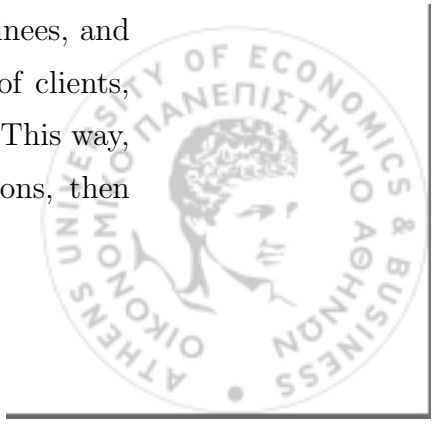


implementations of the functions defined by the ERC-20 token standard. These extensions are used for providing access control as follows.

Initially, a user that "owns" the smart contract assigns all tokens to himself. We refer to this user as the "owner." Then, the owner transfers at least one token to each authorized client. As a matter of fact, the number of tokens a client owns can be used as an indication of his role; the more tokens he owns, the more privileged his role. The smart contract owner can protect an operation by specifying the roles, i.e., the balance in custom tokens, of authorized clients. Therefore, in the simplest case, an operation can be protected simply by having the smart contract function checking if the client that invokes it owns the necessary number of tokens (this validation is trivially implemented using the `balanceOf` function). However, our approach can handle some more sophisticated and novel constructions.

Token transfer. In theory, and based on the ERC-20 semantics, any client can transfer some of his tokens to another client using the transfer function. Of course, speaking about ACTs, this constitutes a security threat, since this way a client authorizes another client – potentially malicious – to perform an operation. It should be noted here that this is an existing threat in legacy token-based access control systems. Fortunately, ERC-20 defines only an "interface" and does not dictate any particular implementation choice. Hence, in our smart contract, a client is allowed to transfer his ACTs only to the owner. This transfer is enabled in order to support functionalities, such as "shifts," where a client is authorized to perform an operation only for a specific time period (that corresponds to his shift) and then transfers through the owner his authorization to the client of the next shift. It should be noted here that off-chain ACT transfers are impossible.

Probation periods. Another interesting capability of an ERC-20 compatible smart contract is that it can modify the token balance of a user at will. In our mechanism, we leverage this feature to support clients in probation, trainees, and other similar roles. In particular, we allow the owner to define a list of clients, whose balance is decreased by one every time they invoke an operation. This way, these clients are allowed to perform only a certain number of operations, then

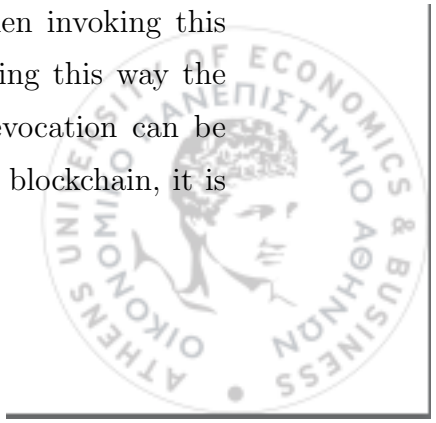


the results of these operations are inspected (out of band), and if everything is as expected, the clients regain their tokens back.

Supervised operations. Using our mechanism, it is possible to define "critical" functions, that require the "approval" of a client that holds a more privileged role. In particular, if such a function is invoked by an underprivileged client, instead of producing an operation event, a new type of event is produced called `AuthorizationRequest`. This event is handled by a privileged client, who inspects its fields and acts accordingly, i.e., he may ignore it, or he may invoke the same function again so that the `Operation` event is generated. This operation can even be asynchronous.

Two-step access control. Since the Ethereum ledger is distributed any full node (or RPC client) can learn the token balance of a user without interacting with the corresponding smart contract, by just inspecting the blockchain. This property enables the definition of additional (possibly finer grained) access control policies at the IoT gateways. This means that even if a client is authorized by the smart contract, eventually his operation may be rejected by some/all IoT gateways. The access control policies defined at the IoT gateways may take into consideration, in addition to the role of the client, other auxiliary information provided by the "real" world, such as time, location information, other IoT measurements, etc. Notice that Ethereum smart contracts do not have access to such information. This mechanism is extremely useful, as it allows our solution to be combined with traditional access control solutions at the IoT gateways.

Panic button. Our access control smart contract defines a function that can be invoked only by the owner and it resets the token balances of all users, returning in essence all ACTs back to the owner. This function can be used in case of emergency, e.g., in case of a security breach. Additionally, when invoking this function, the owner can specify the public key of a user, resetting this way the balance of that particular user. Using this approach, client revocation can be trivially implemented. Since all transactions are recorded in the blockchain, it is



Operation	**Cost measured in gas**
Contract deployment	1418480
Send Operation event	2560
Search map	1033
Map entry creation	45938
Map entry modification	6110

Table 4.2: Cost of the construction building blocks of the large-scale IoT architecture.

painless to restore user balances to their value prior the "panic button" invocation. Moreover, the clients, whose tokens are revoked have no control over this process, hence revocation is instantaneous and effective using only a single transaction.

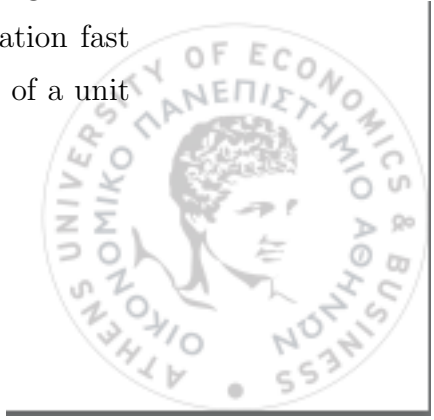
Evaluation and Discussion

Performance and cost evaluation. We have implemented and tested our proposed solution in a private Ethereum network, as well as, in a number of Ethereum test networks, including Rinkeby and Ropsten Ethereum testnets. As an IoT gateway, we have used Mozilla's Things Gateway that implements the WoT standards.[2] We implemented clients as JavaScript Web applications using web3.js Ethereum JavaScript API and the Metamask Firefox extensions.[3]

As we have already mentioned, the invocation of an Ethereum smart contract function creates some computational overhead measured in "gas" units, which depends on the operation's complexity. Each user declares the price he is willing to pay per gas unit; the bigger the amount, the faster the operation will be executed. The fastest an operation can be executed in Ethereum is ≈ 15 seconds, which is the time required by the Ethereum network to generate a new block. However, the block time can vary slightly due to various factors, such as network congestion. Hence, users compete each other since they wish to execute their operation fast but they do not want to get charged a lot. Currently, the average price of a unit

[2]https://webthings.io/

[3]https://web3js.readthedocs.io/en/v1.10.0/index.html

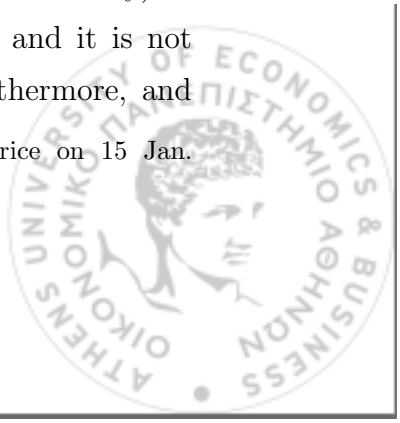


of gas is \$0.0729.[4] Our construction uses Ethereum's events and is built using a "mapping" type, i.e., a hashtable-like data structure that maps "keys" to "values." Our events have two fields, namely OPCode of type *byte* and URIResource of type *byte32*, i.e., a byte array of size 32. Furthermore, our mapping maps keys of type *address* to values of type *int* and it is used for maintaining client's balance. The primitive operations required by our smart contract are map *search*, *creation* of a new map entry, and *modification* of the value of a map entry. Table 4.2 shows the cost of these operations in terms of gas consumption.

Qualitative and security properties. Our construction leverages the inherent properties of the blockchain technology. By design, blockchain solutions offer reliability and robustness, since the ledger is replicated in multiple locations and there is no single point of failure. Furthermore, blockchain communication protocols and APIs include message integrity protection, as well as, resilience against replay attacks. Smart contract execution is deterministic and cannot be affected by malicious entities. Similarly, smart contracts cannot be modified, not even by their owners. As already discussed, invoking a smart contract function has some monetary cost; this could be an effective defense against DoS attacks.

As far as our token-based access control is concerned, it can be observed that it has some intriguing security properties. Firstly, ACTs can only be used by their owners, and ACT owners cannot transfer them to other users. Even if the blockchain keys of a user are compromised, our construction prevents ACT transfer (of course the stolen keys can be used for issuing transactions on behalf of the victim users). This is a significant advantage compared to traditional token-based access control mechanisms, where not only the corresponding ACTs have to be secured, but also an ACT recipient should be able to verify the binding between the ACT and the user who sent it (i.e., additional mechanisms for detecting stolen ACTs should be in place). In other words, the responsibility (and security) of binding of ACTs to ACT owners is performed by the blockchain and it is not the responsibility of each user (which opens security issues). Furthermore, and

[4] As measured by https://ycharts.com/indicators/ethereum_average_gas_price on 15 Jan. 2025

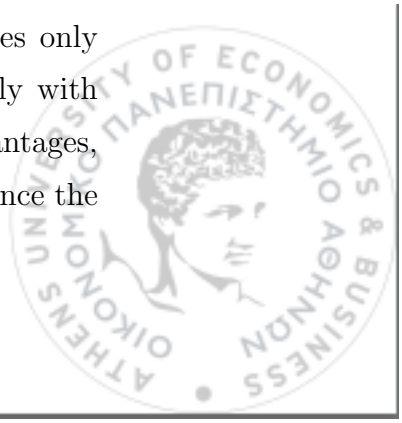


as already discussed, blockchains are an indelible, append-only, and tamper-proof logs, hence, in case of a security incident or in case of a dispute they can provide undeniable auditing information. Moreover, our construction offers secure and effective revocation. Ethereum's mechanisms guarantee that only an owner can revoke ACTs (providing of course that the owner's private key is secured), as well as, that a token revocation has immediate affect. Finally, since our construction is based on an established Ethereum standard, libraries and wallets that support it, can be used for implementing client applications.

Ethereum is composed of a P2P network, where all valid blocks are broadcast to all nodes. In reality, events are special fields encoded in those blocks, hence the number of nodes watching for events does not have any impact on the number of transmitted messages. In other words, if we take Ethereum infrastructure for granted (or any other similar architecture), it is costless to build a group communication application on top of that. Moreover, due to this property, the network location of the IoT gateways does not have to be well known, neither have gateways to be reachable through the Internet.

Discussion. Despite the advantages of the blockchain technology, it comes with some costs. As already discussed Ethereum (and most blockchain systems) involve some monetary cost, as well as some transaction delay. Unfortunately, the monetary cost of transactions fluctuates greatly. Furthermore, Ethereum operates on the premise that at least half of the network nodes are honest; having an attacker controlling more than 50% of the nodes is an unlikely but not impossible threat. Finally, Ethereum's ledger is public and anybody can inspect it. This property constitutes a privacy threat since it is possible for a third party to deduce information such as, who perform which operation and when, the "roles" of the users, the introduction of new authorized users, etc. All these shortcomings can possibly be addressed using a permissioned private blockchain, such as Fabric.

In the construction presented, clients interact with the IoT devices only through the blockchain. Of course cases, where a client interacts directly with an IoT gateway can be considered. This direct interaction has some advantages, including zero transaction fees and faster response times. Moreover, and since the

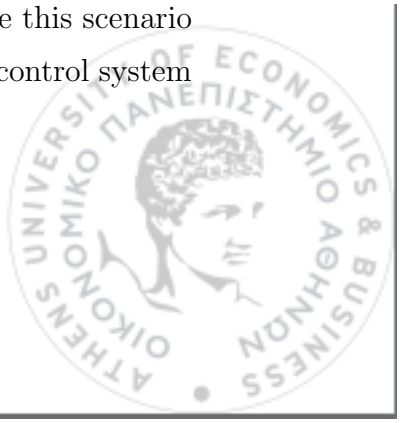


Ethereum ledger is replicated in all nodes, a gateway can still perform token-based access control. On the other hand, in this case, the gateway should be able to verify the identity of the client, i.e., his blockchain public key.

4.1.2 Multi-tenant smart city

As we have already mentioned, ERC-20 tokens are fungible, meaning that each token is identical in type and value. Therefore, in our previous solution, we cannot distinguish users who hold the same balance. So, attempting fine grained roles (e.g., sub-roles) and access control policies is not trivial and may require extra logic to the smart contract, leading to higher costs and higher risk of implementation errors. To avoid that, we demonstrate a slightly different design for a multi-tenant smart city scenario [53].

Consider a smart city, in which numerous IoT devices have been deployed in order to provide services to the citizens and to address many of the urbanization challenges. There are many types of IoT devices that can be deployed in a smart city, from air pollution and temperature sensors, to connected cameras, streetlights, and GPS trackers. Each type of IoT device provides a different service to the community. For illustration, we consider four types of services: traffic monitoring, video surveillance, connected streetlights, and weather monitoring. Some of these services, such as the traffic and weather monitoring, can be used by any citizen, while others, such as the video surveillance and connected streetlights, can only be used and managed by the appropriate authorities. In our case, the municipal government (which consists of many persons, such as the mayor, deputy mayor, et al.) is the ("owner" and) "manager" of the IoT devices deployed in the smart city. The owner should allow access to the traffic and weather services to the citizens. In addition, it should allow to the appropriate authorities, e.g., the police department, which again consists of many persons, to manage and access the services of connected streetlights and video surveillance. We will use this scenario to demonstrate the benefits of blockchain-based token-based access control system in large, complex, and multi-tenant IoT environments.



https://doi.org/10.26219/heal.aueb.9346

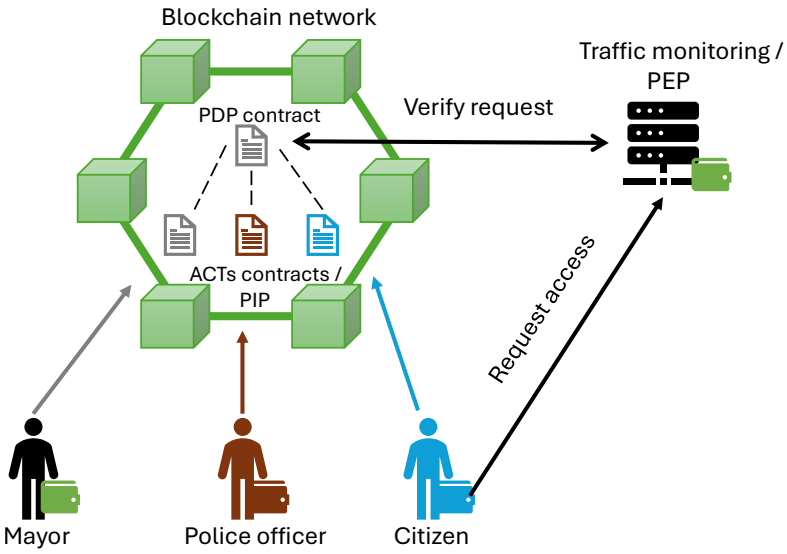


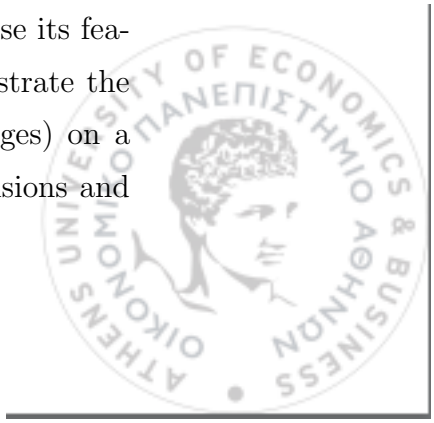
Figure 4.2: Blockchain-based token-based access control architecture for a multi-tenant smart city use case.

System design

Our architecture, illustrated in Figure 4.2, is composed of the following entities:

- The blockchain infrastructure
- A smart contract that acts as a PDP
- Many ACT smart contracts that acts also as PIPs
- The municipal government
- Clients wishing to interact with IoT devices and the corresponding services
- IoT services that act also as PEPs

As the blockchain, we use the public Ethereum blockchain because its features and properties are well known and trusted in order to better illustrate the methodology. However, our solution can also be based (with no changes) on a private instance of the Ethereum blockchain, or with appropriate extensions and

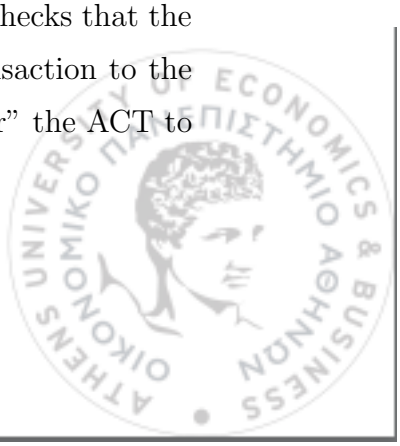


modifications on permissioned blockchains, such as Fabric, or even combinations of permissionless and permissioned blockchains through interledger technologies [56].

In our system, we introduce two types of smart contracts. The *PDP contract* is responsible for deciding if an actor can gain access to a service or not. All other smart contracts, called *ACT contracts*, are responsible for creating and managing ACTs. All the ACT contracts implement the ERC-20 token standard. We have three ACT smart contracts. One for creating and managing the ACTs that correspond to the owner and the managers of the IoT devices, the provided services, and the system as a whole (the municipal government). The second smart contract creates and handles the ACTs for the citizens, while the last ACT contract creates and manages the ACTs for the police department. In addition, we can easily add more ACT contracts that correspond to other authorities of the city by just deploying new ERC-20 contracts with the appropriate functionality.

From a high level perspective, the entities of our system interact with each other as follows. Initially, the owner develops and deploys all the smart contracts of the system on the Ethereum blockchain. Then, the owner modifies the ACT contracts and for each contract adds an appropriate operator. For example, for the ACT contract that corresponds to the police department, the operator is the police chief. For the citizens ACT contract, the operator is someone chosen by the municipal government. Subsequently, the owner modifies the PDP contract. He creates a mapping that shows which services can be accessed from which type of ACTs. For instance, the video surveillance service is mapped to the police department ACT, since only the users having such ACTs can access this service.

In order for a client to request an ACT, he has to send a transaction to the appropriate smart contract to invoke the `requestACT` function. When this function is invoked, it generates an event that contains the Ethereum address of the requester (in essence, his public key). The event is eventually "caught" by the operator, who "listens" to the blockchain for events. The operator checks that the client's address is valid (client authentication) and he sends a transaction to the blockchain, and in particular to the `transfer` function, to "transfer" the ACT to the client's blockchain wallet. Otherwise, the request is denied.

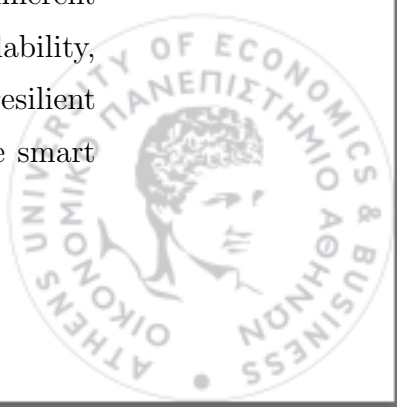


From this point on, a client is able to request access to a specific service or IoT device. To do so, the client sends an access request to the service or IoT device he wants to interact with. Then, the request is forwarded to the PDP contract. In particular, a transaction is sent to the PDP contract to invoke the `requestAccess` function. The transaction includes the service that the client wants to access and his Ethereum address. The smart contract checks the map for the requested service to find out which of the tokens are appropriate for that specific service, e.g., if a client request access to the camera surveillance service, then he has to own an ACT that corresponds to the police. Then, the smart contract verifies that the address of the client has the required token. To verify that, it interacts with the appropriate ACT contract and invokes the `balanceOf` function with the client's Ethereum address as a parameter. If all the requirements are met, the smart contract returns "allow access," otherwise "deny access."

The PDP contract implements also some other administrative actions that are responsible for managing the whole system. Namely, it includes a function that changes the operator of an ACT contract. Moreover, it has a function that changes the ownership of the whole system, if the municipal government changed. These functions can only be invoked by the owner. Furthermore, each ACT contract includes functions for revoking and delegating the ACTs. The delegation function can be invoked only by the clients that already have acquired an ACT, while the revocation function can only be invoked by the operator of the contract and the owner. In essence, these functions are implemented using the `transfer` function of the ACT contracts. However, at its current design, a client can delegate his ACT to anyone. This opens security issues, thus additional mechanisms for checking the delegation process should be in place.

Discussion

As in the previous design, our solution takes advantage of all the inherent properties of the Ethereum blockchain. It offers reliability, robustness, availability, immutability, non-repudiation, auditability, message integrity, and it is resilient to several attacks, such as DDos and replay attacks. With regards to the smart

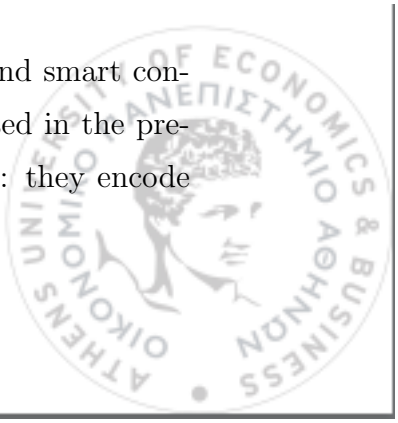


contracts, the execution of them is deterministic and since the smart contracts are part of the blockchain, they are immutable too. Thus, we are sure that the access decisions will always be correct and respected no matter what. So, only the clients having the right ACT will gain positive answer to their access request. However, the enforcement of the access decision does not happen on the PDP contract nor on the ACTs contracts, but it happens on the IoT devices or the software that manages the provided services. Thus, if a provided service is compromised, then a malicious user can might get access to it, without owning the appropriate ACT. Additionally, our system has many advantages compared to traditional access control mechanisms. First of all, it is decentralized and it is lightweight. It is much easier and simpler for a constraint IoT device to just read the blockchain for the decision instead of performing advanced cryptographic techniques to check the validity of an ACT or to decide if an access decision is valid or not. Furthermore, as previously, our solution provides a secure, effective, and instantaneous revocation mechanism. Finally, it presents the same drawbacks as the previous design. In particular, it introduces monetary costs and transactions delays. However, to avoid that our presented system, depending on the use case, can easily be deployed on private Ethereum, or even with few modifications on Hyperledger Fabric and other blockchains.

4.1.3 Related work

Early attempts to incorporate the blockchain technology into the IoT proposed new blockchain systems, such as the WAVE [43]. Another such work presented by Dorri et al. [57] designed a blockchain-based smart home management system. They proposed a custom, blockchain technology, where the home gateways hold the role of the miners. Such solutions are hard to be deployed since they require a "critical mass." Our approach is built on existing technologies and can be used with already available libraries and wallets.

More recent attempts are using the blockchain technology and smart contracts to provide security and access control for the IoT, as discussed in the previous chapter [48, 49, 50]. These solutions follow a similar pattern: they encode



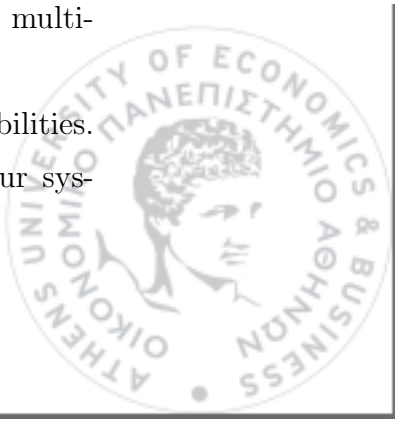
in a smart contract the actions a specific user can perform to a particular IoT device/resource. Our solution extends these approaches by considering the token balance of users in the access control policies. In other words, these solutions resemble to an access system, which is based on usernames and passwords, whereas our solution resembles to RBAC model. Furthermore, by leveraging the handling functionalities of known and popular Ethereum standards to realize ACTs, our approach enables some novel constructions.

Finally, Hanada et al. [58] explored the possibilities of smart contracts to Machine-to-Machine (M2M) communication. To this end, they developed and evaluated an IoT application for automated, M2M, gasoline purchases that uses Ethereum smart contracts to perform transactions. Our work is also in this direction. Nevertheless, in addition to merely using smart contracts to provide message transfer and payments, our solution supports group communication and access control.

4.1.4 Conclusions and future work

In this work, we designed, developed, integrated into real life scenarios, and evaluated blockchain-based ACTs. In particular, in [52], we designed, developed, and evaluated an IoT architecture, based on smart contracts and blockchains. Our solution leverages the distributed nature of the blockchain technology to build an event-based system. Furthermore, we enhanced our architecture with an access control solution based on custom blockchain-based tokens. We showed that our access control solution has some intriguing properties and presents some important advantages compared to traditional access control systems, such as improved efficiency, high availability, auditability, non-repudiation, and rich functionality, including secure and effective revocation. Finally, our Ethereum-based implementation shows that our solution is feasible with low overhead. Additionally, in [53], we presented an alternative design of our token-based access control for a multi-tenant smart city use case.

Blockchain is an exiting, evolving technology, with endless possibilities. Hence, our system can be extended in numerous ways. For instance, our sys-

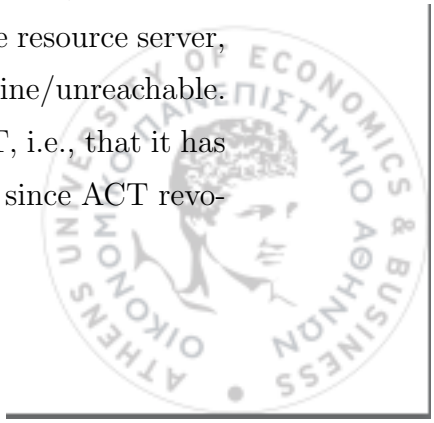


tem can be extended to support IoT-based sharing economy, or even auctions over IoT ACTs (e.g., a funny use case could be an auction for the token that can light the Christmas tree of a city). Similarly, the blockchain can be used for tracking user reputation or even "score" in a gameficated application. Finally, our architectures can be extended to support blockchain-based Decentralized Identifiers (DIDs), a new technology under standardization with exciting security and privacy properties.

4.2 OAuth 2.0 authorization using blockchain-based tokens – ERC-721 tokens

As mentioned, ERC-20 tokens are identical in both type and value, so users cannot be uniquely distinguished on-chain. Thus, ERC-20 tokens are not appropriate for identity-based access control. For that reason, we also propose a different type of ACT, which is protected using proof-of-possession keys and at the same time it supports auditing and accountability, fast revocation, and added-value services [59]. In order to achieve our goal, we build again on the blockchain technology. Our solution considers an append-only distributed ledger, where users, identified by a public key, can transact *uniquely identified tokens*. Our implementation is based on the Ethereum blockchain and the ERC-721 token standard [15]. Our proposed system leverages smart contracts to build blockchain-based token management services. The feasibility of our solution is verified through a proof of concept implementation, where we integrate our solution on OAuth 2.0 protocol for an IoT gateway access use case. The proposed solution has the following advantages.

- The entity that generates ACTs (i.e., the authorization server in the OAuth 2.0 protocol) can easily revoke them before they expire. Furthermore, ACT revocation does not involve any interaction with the client or the resource server, hence it can be implemented even if these entities are offline/unreachable. Similarly, a resource server can verify the validity of an ACT, i.e., that it has not been revoked, even if the authorization server is offline, since ACT revo-



cation is recorded on the blockchain through a transaction that can occur at any time prior to the ACT's usage.

- Clients do not have to store their ACTs locally, neither do they have to store any secret associated with their ACTs. Instead, all ACTs can be retrieved form the ledger. Hence, ACTs are portable and can be easily used by multiple client devices. Furthermore, since we are using a popular token specification, a wide range of "wallets" and libraries can be supported by our system.

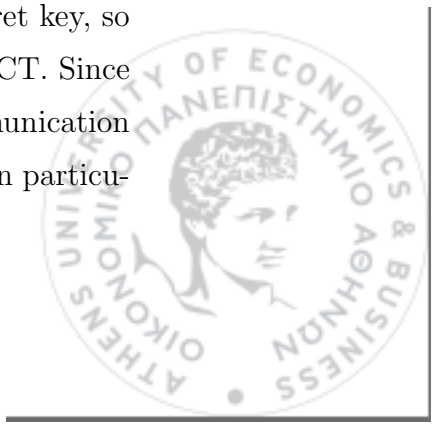
- ACT integrity and authenticity can be verified simply by performing a lookup in the ledger, and it does not involve any signature verification or any other cryptographic operation. Therefore, our solution is less prone to implementation errors, and ACTs are simpler, since they do not carry any cryptographic proof. Furthermore, token ownership can be securely modified without any interaction with the authorization server.

- All tokens, including the revoked ones, are immutably stored in the ledger, hence auditing and accountability mechanisms are facilitated.

4.2.1 Background

JSON Web Tokens

In a nutshell, OAuth 2.0 constitutes the protocol through which a client obtains an ACT from an authorization server, to access a protected resource, stored in a resource server. However, OAuth 2.0 specification does not define how an ACT is generated, validated, and destroyed; instead it leaves the management of OAuth 2.0 token's lifecycle as an open design choice. Each OAuth 2.0 deployment may choose the type of ACT, it will use. The most commonly used type of ACT is the *bearer* ACT [60], which can be used by any user who possesses it, i.e., the "bearer." For additional security, an ACT can be associated with a secret key, so that only users, who can prove that they possess this key can use the ACT. Since the latter type of ACTs provides more security, at the cost of the communication overhead required to verify ownership, it is considered by our solution. In particu-



Name	Semantics
iss	The issuer of the token
sub	The subject of the token, i.e., the entity that will use the token to gain access to a resource
aud	The audience of the token, i.e., the the recipients that the JWT is intended for
exp	The expiration time on or after which the JWT must not be accepted for processing
jti	A unique token identifier

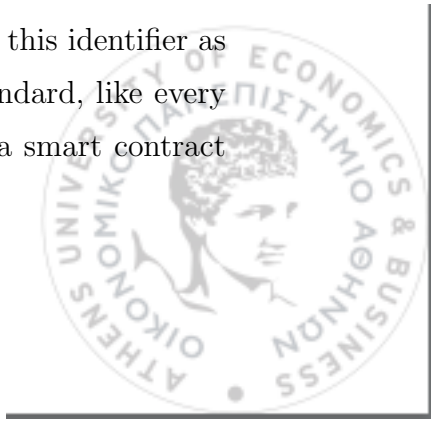
Table 4.3: JWT claims used in our OAuth 2.0 system that uses ERC-721 blockchain-based Access Control Tokens.

lar, our constructions are based on JSON Web Tokens (JWTs) [61], enhanced with blockchain-based proof-of-possession mechanisms.

A JWT is a compact, URL-safe means of representing "claims." It consists of zero or more name/value pairs, and it is transmitted encoded in base64url [62]. RFC 7519 [61] defines some "standard" claim names and their semantics. Table 4.3 contains the names and the corresponding semantics of the JWT claims used by our solution.

ERC-721 token standard

ERC-721 [15] is an open standard that describes how to build *non-fungible* or unique tokens on the Ethereum blockchain. This standard is very similar, in many ways, to ERC-20 token standard, which is probably the most popular Ethereum standard. However, in contrast to ERC-20 tokens, ERC-721 tokens are *unique* and non-interchangeable with other tokens (non-fungibility). Many Ethereum wallets, such as Metamask, can handle also these tokens. All ERC-721-based tokens are identified by a unique identifier (we will refer to this identifier as $token_{id}$), and they can be owned only by a single user. This standard, like every other token standard in Ethereum, defines some functions that a smart contract



Name	**Purpose**
ownerOf($token_{id}$)	It accepts as input a $token_{id}$ and returns the address of the token owner
transferFrom(from, to, $token_{id}$)	Transfers a $token_{id}$ from one address to another
approve(address, $token_{id}$)	It approves an address to manage a $token_{id}$ on owner's behalf
getApproved($token_{id}$)	It retrieves the address that is allowed to manage $token_{id}$
tokenURI($token_{id}$)	Returns a URI that point to $token_{id}$'s metadata

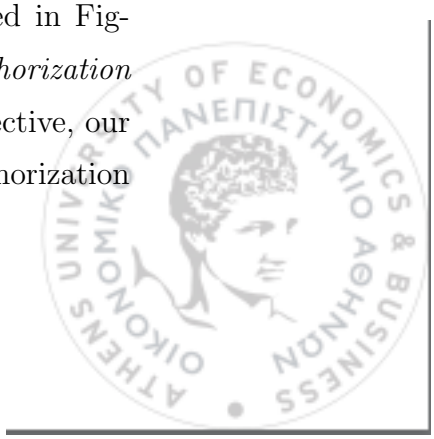
Table 4.4: ERC-721 token standard and ERC-721 metadata extension main functions.

should implement in order to be able to create and handle ERC-721 tokens. Furthermore, the ERC-721 metadata extension, defines some additional functions that can be used for associating an ERC-721 token with metadata. Table 4.4 describes the functions defined in ERC-721 and in ERC-721 metadata extension, used by our system.

Functions `transferFrom` and `approve`, when invoked each generates an event, named *Transfer* and *Approval* respectively. Both these events have three attributes; the attributes of the Transfer event are the from address, the to address, and the $token_{id}$, while the attributes of the Approval event are the owner address, the approved address, and the $token_{id}$.

4.2.2 System design

Our system considers a typical OAuth 2.0 architecture, depicted in Figure 3.2. Therefore, the main entities of our system are *clients*, *authorization servers*, *resource servers*, and *resource owners*. From a high-level perspective, our system operates as follows. The client requests an ACT from the authorization



https://doi.org/10.26219/heal.aueb.9346

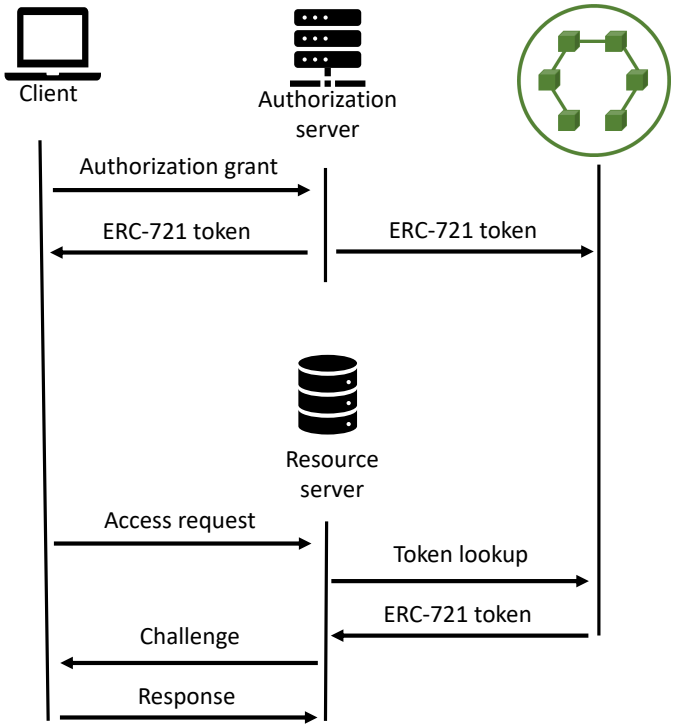
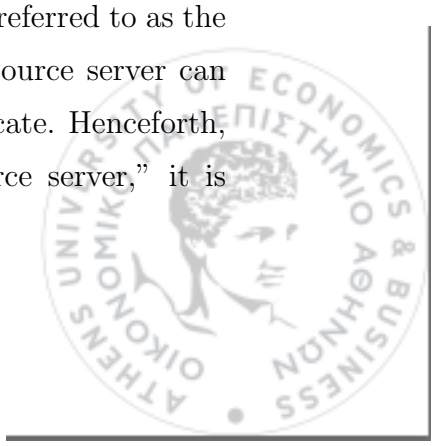


Figure 4.3: OAuth 2.0 using ERC-721 tokens architecture.

server, using an authorization grant received by the resource owner. The authorization server validates the authorization grant of the client, generates a JWT and an ERC-721 ACT, transfers the ERC-721 ACT to the Ethereum address of the client, and transmits the JWT to the client. Then, the client requests access from the resource server, providing the JWT. The resource server retrieves the corresponding ERC-721 ACT, which is used for verifying the validity and the ownership of the JWT: if all verifications are successful the resource server allows the client request. This process is illustrated in Figure 4.3.

Notation and security assumptions

In our system, resources are uniquely identified by a URI, referred to as the $URI_{resource}$. We assume a security mechanism with which a resource server can prove that it hosts a specific $URI_{resource}$, e.g., using a X.509 certificate. Henceforth, when we say that "a client requests $URI_{resource}$ from a resource server," it is



assumed that the client has already verified $URI_{resource}$ ownership. Moreover, the communication between a client and a resource server takes place over a secure communication channel.

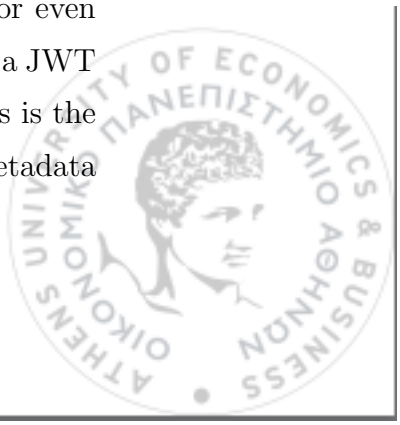
In our system each authorization server owns an ERC-721 contract, referred to as $Contract_{AS}$; we detail these smart contracts in the following section. Moreover, we assume a security mechanism with which clients and authorization servers can securely communicate; this mechanism provides confidentiality and integrity protection of the exchanged messages, as well as, authorization server authentication. Furthermore, resource owners trust authorization servers to behave according to the specified procedures.

Finally, each client owns a public key PK_{Client}, and a corresponding private key, used for transacting with the Ethereum blockchain. Again, we consider a security mechanism with which PK_{Client} ownership can be verified.

The ERC-721 smart contract

ERC-721 contracts in our system are implemented such that only the contract owner can create new ACTs. When a token is created its $token_{id}$ and *metadata* are specified: these two properties are read-only and they cannot be modified. Moreover, ACT owners in our system cannot transfer their ACTs; the only entity that can invoke the `transferFrom` method is the contract owner. Moreover, when invoking the `transferFrom` method, the contract owner is allowed to transfer any token, no matter who the token owner is.

The ERC-recommended approach for associating a token with some metadata is through the metadata extension, which provides a method that maps a $token_{id}$ to a `URI`, where metadata are stored, i.e., `tokenURI`. Nevertheless, we postulate that this approach violates both the decentralization and immutability principles of DLTs, since with this approach the metadata file is stored in a centralized location, and it can be modified without being possible to track or even detect the changes. For this reason, in our system, metadata are encoded in a JWT and the base64url representation of the JWT is stored in the contract: this is the return value of the `tokenURI` method. Therefore, in our system, the metadata



extension is used to retrieve the metadata themselves and not a URI that points to the metadata.

Setup

During the setup phase, resource owners configure their resource servers with $Contract_{AS}$. Moreover, clients "register" with the authorization server.[5] During this registration process, which is out of the scope of this paper, the authorization server learns the Ethereum public key of the client, i.e., PK_{Client}. In cases where this registration process cannot take place a priori, solutions such as the OAuth 2.0 dynamic client registration protocol [63] can be considered. Additionally, prior to any other operation, each client obtains from the resource owner an authorization grant (the format of this grant and the mechanisms for generating it are out of the scope of this paper). This grant represents a resource owner's authorization, and is used by a client to request an ACT for a specific $URI_{resource}$.

ACT request

A client, that owns PK_{Client}, requests from an authorization server an ACT for a protected resource $URI_{resource}$, including in the request the authorization grant received during the setup. The authorization server verifies PK_{Client} ownership and grant validity (using protocols out of the scope of this paper). Then, the authorization server creates a JWT, which contains the claims show in Listing 4.1.

Listing 4.1: The generated JWT.

```
{
    ''iss'': ''Contract_AS'',
    ''sub'': ''PK_Client'',
    ''aud'': ''URI_resource''
    ''jti'': ''token_id''
    ''exp'': ''expiration time''
}
```

[5] This registration step is also assumed by the OAuth 2.0 protocol.

As a next step, the authorization server creates an ERC-721 ACT. The $token_{id}$ of this new ACT matches the value specified by the `jti` claim of the JWT. Moreover, the metadata of the token is set equal to the base64url encoding of the JWT. Finally, the authorization server invokes the `transferFrom` method of the ERC-721 contract to transfer the created ACT to PK_{Client}, and sends the generated JWT back to the client. It should be noted that the client does not have to store the JWT: at any time, he can retrieve all the ACTs he owns from the ERC-721 contract, and extract the corresponding JWT from an ACT's metadata.

Resource access request

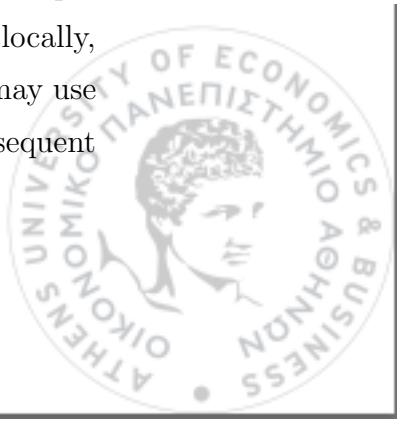
In order for a client to access a protected resource, it sends to the resource server a request that includes the received JWT. The resource server performs the following steps.

1. It examines: (i) if it "knows" the $Contract_{AS}$ included in the `iss` claim of the JWT (i.e., if it has been configured with this contract address), (ii) if the $URI_{resource}$ included in the `aud` claim of the JWT matches the URI of the requested resource, and (iii) if the token is still valid (i.e., it has not expired).

2. It executes the `ownerOf` method of $Contract_{AS}$, providing as input the $token_{id}$ included in the `jti` claim of the JWT, and examines if the returned address corresponds to PK_{Client} included in the `sub` claim of the JWT.

3. It executes the `tokenURI` method of $Contract_{AS}$, and examines if the returned string is the same as the received JWT.

At this point, the resource server is able to attest the integrity of the received JWT (since it is the same as the metadata of the token stored in the blockchain), as well as its validity. As a next step, the resource server verifies that the client is the real owner of PK_{Client}. If all verifications succeed, the resource server accepts the client's request. Additionally, a resource server may store a valid JWT locally, create a *session identifier*, and send this identifier to the client: the client may use this identifier, as long as the JWT is still valid, accelerating this way subsequent requests for the same resource.



https://doi.org/10.26219/heal.aueb.9346

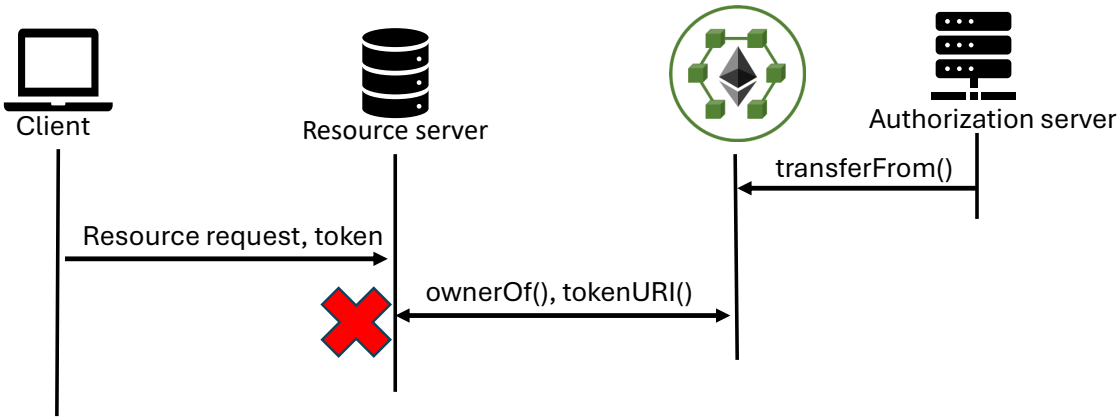


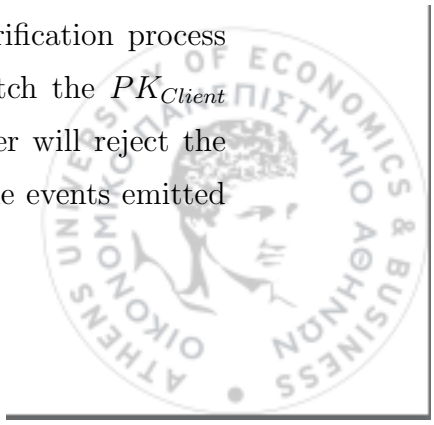
Figure 4.4: Revocation of ERC-721 tokens on OAuth 2.0.

4.2.3 Token management services

Revocation

ACTs usually carry an expiration time: OAuth 2.0 and JWTs specifications do not provide any mechanism that allows an authorization server to revoke an ACT prior to its expiration time. Even "OAuth 2.0 token revocation RFC [64]," specifies a mechanism that allows "clients to notify the authorization server that a [...] ACT is no longer needed," i.e., a mechanism that provides a "log out" functionality rather than revocation.

In our system, ACTs can be revoked by an authorization server by invoking the `transferFrom` method of the ERC-721 contract and transferring an ACT back to the authorization server (it is reminded that authorization servers are the smart contract owners and smart contract owners in our system are allowed to transfer any ACT, no matter who the ACT owner is), as shown in Figure 4.4. We consider two cases: (i) the corresponding JWT has not been used by the client by the time of the revocation, and (ii) the corresponding JWT has been used, it has been stored locally by the resource server, and it has been associated with a session identifier. In the former case, when a client tries to use the JWT the verification process will fail, since the output of the `ownerOf` method will not match the PK_{Client} included in the sub claim of the JWT, hence the resource server will reject the JWT. In the latter case, the resource server must "listen" for the events emitted



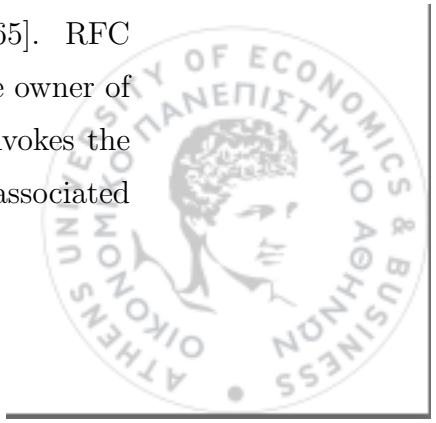
by the `transferFrom` method; if an event contains a $token_{id}$ included in an already stored JWT, the resource server must delete the JWT and the associated session identifier. It should be noted that events are immutably stored in the blockchain, hence if a resource server is offline for some time, it can easily recover all missed events.

Delegation

Listing 4.2: JWT with delegation enabled. cnf stands for "confirmation."

```
{
    ''iss'': ''Contract_AS'',
    ''sub'': ''PK_Client'',
    ''aud'': ''URI_resource''
    ''jti'': ''token_id''
    ''exp'': ''expiration time''
    ''cnf'':
    {
        ''kid'': ''PK_Delegee''
    }
}
```

There can be cases, where a client does not wish to authenticate to the resource server using PK_{Client}, e.g., PK_{Client} may be stored in a secure, offline storage place, or a user may want to temporary use another device that does not have access to PK_{Client} (for example, a user may want to use different devices while traveling). For these cases, our system allows an ACT owner to delegate an ACT to another Ethereum address. The ACT does not change ownership, and the latter address is not allowed to further delegate the token. We implement this functionality by using the `approve` method of the ERC-721 contract and the "proof-of-possession key Semantics for JWTs" defined in RFC 7800 [65]. RFC 7800 defines a "confirmation" (`cnf`) claim, which contains the key of the owner of a JWT. Delegation of a $token_{id}$ is implemented as follows: PK_{Client} invokes the `approve` method providing as input $token_{id}$ and the Ethereum address associated



https://doi.org/10.26219/heal.aueb.9346

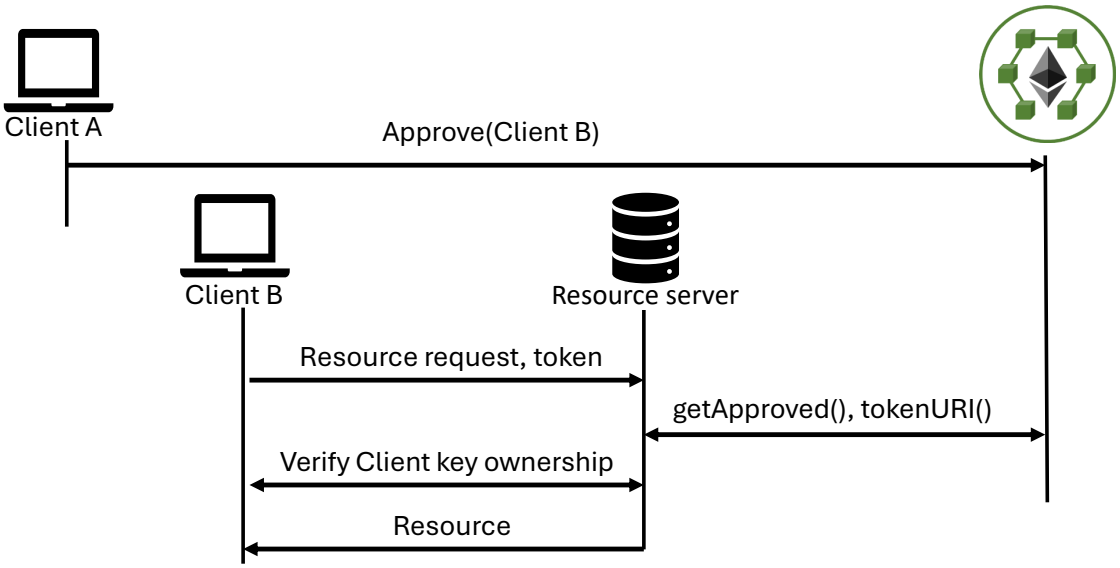
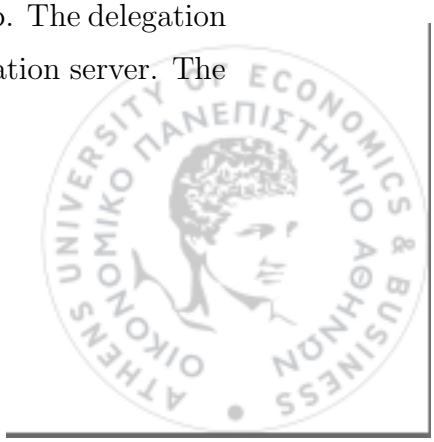


Figure 4.5: Delegation of ERC-721 tokens on OAuth 2.0.

with the public key of the delegee $PK_{Delegee}$. Now $PK_{Delegee}$ can use this token by constructing a JWT and by adding $PK_{Delegee}$ in the `cnf` claim, as illustrated in listing 4.2. The `cnf` claim is not recorded in the ERC-721 token's metadata, since metadata are read-only. For this reason, when $PK_{Delegee}$ performs a resource access request, Step 3 of token integrity verification is modified as follows. It should be noted that when a token is revoked, the delegee can not use it.

3. The resource server executes the `tokenURI` method of $Contract_{AS}$, and examines if the returned string is the same as the received JWT **excluding the cnf claim**.

Then, JWT ownership is verified as follows: the resource server executes the `getApproved` method of the ERC-721 contract, providing as input $token_{id}$, and examines if the return value equals to the Ethereum address associated with $PK_{Delegee}$; finally, it challenges client to verify $PK_{Delegee}$ ownership. The delegation process does not involve the resource owner, neither the authorization server. The delegation process is shown in Figure 4.5.



https://doi.org/10.26219/heal.aueb.9346

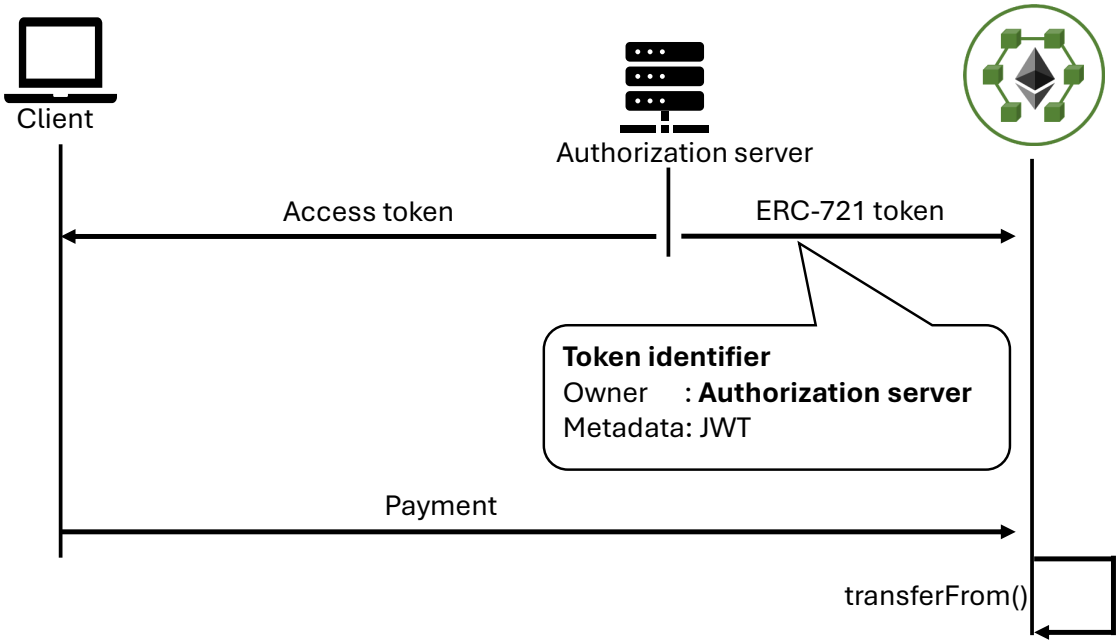


Figure 4.6: Fair exchange of ERC-721 tokens on OAuth 2.0.

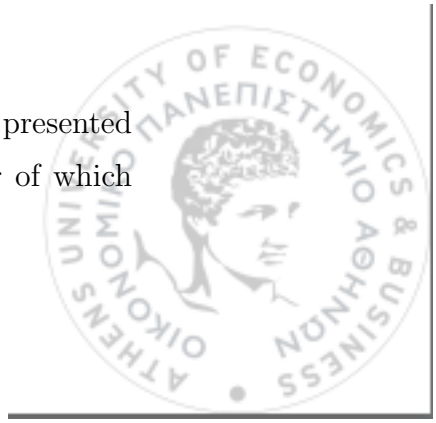
Fair exchange

Smart contracts are ideal for performing "fair exchange" of digital goods [66]. In our solution "fair exchange" of ACTs works as follows (shown also in Figure 4.6). The authorization server creates the ERC-721 token, and stores it in the ERC-721 smart contract by "indicating" a PK_{Client}; the smart contract transfers the ACT to PK_{Client} only when the latter performs an action, e.g., pay a pre-specified amount of money. The advantage of this approach is that the client can inspect the ACT before performing any action; of course the client cannot use the ACT, before performing the specified action, since up until this point, the client does not own the ACT.

4.2.4 Implementation and evaluation

Implementation

We have developed a proof of concept implementation of the presented solution. We considered the case of an IoT gateway access, the owner of which



wishes to grant access to guest users. As an IoT gateway we used the WebThings Gateway that implements the WoT standard. For our proof of concept, we chose not to modify the gateway itself, instead, we have developed an application that acts as a proxy, between the client, the blockchain, and the gateway (that holds the role of the resource server). As an Ethereum wallet, we used the Metamask Firefox extension, which can handle ERC-721-based tokens. We implemented clients as JavaScript web applications using web3.js Ethereum JavaScript API.

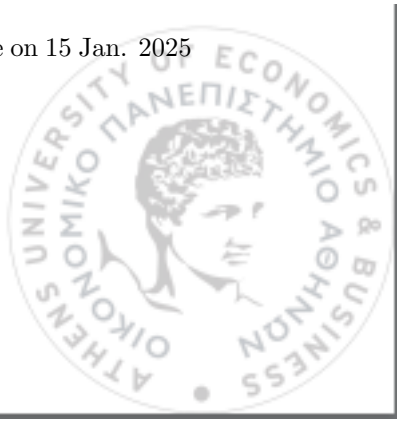
The main component of our proposed system is the smart contract that implements the functions of the ERC-721 interface. The smart contract was developed using Solidity, which is a Turing-complete programming language for implementing smart contracts. In addition to the functions included in Table 4.4, we implemented two functions that are not defined in the ERC-721 interface. The first one, named `mint`, is for creating new ACTs, and the other, named `burn`, for "burning" ACTs, i.e., for destroying them. These functions, as well as, the `transferFrom` and `approve` functions, can only be invoked by the smart contract owner.

Performance and cost evaluation

We have tested our proposed system in the Rinkeby Ethereum test network.[6] We chose a public test network rather than a private one in order to have more reliable results. Table 4.5 shows the cost of deploying the smart contract in the blockchain network, as well as the cost of operations performed by our system in terms of gas units. The average price of a unit of gas is \$0.0729.[7] Some of the aforementioned functions are declared as *view* functions. That is, they only *read* the state of the blockchain, without modifying it. Thus, they incur no cost and delay. These functions are: `tokenURI`, `getApproved`, and `ownerOf`. In addition to gas, Ethereum adds an execution time overhead, which on average is $\approx$ 15 seconds.

[6] https://www.rinkeby.io/

[7] As measured by https://ycharts.com/indicators/ethereum average gas price on 15 Jan. 2025



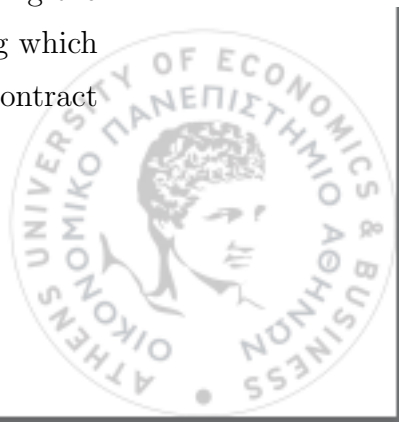
Operation	**Cost measured in gas**
Contract deployment	1585444
Create an ACT	254141
Burn an ACT	85791
Transfer an ACT	63858
Approve	45735

Table 4.5: Cost of the construction building blocks of the ERC-721 Access Control Tokens.

Security evaluation

Privacy considerations. With our solution, an authorization server does not have to be aware of the resource server, i.e., they never have to communicate directly, not even in the case of an ACT revocation. The authorization server learns only the $URI_{resource}$ that a client wants to access, but this does not have to be the real URI of the resource: any pseudonym that the resource server can understand can be used instead. On the other hand, the metadata of an ERC-721 token are immutable and visible to anybody, constituting a privacy threat. In order to enhance clients' privacy, metadata can be encrypted using a key known only to the resource owner, to the client, and to the resource server.

Authorization server key breach. In order to enhance the security of our solution, we specify that some functions of the ERC-721 contract can only be executed by the contract owner, i.e., the authorization server. Of course, if the private key of the authorization server used for deploying the contract is compromised, then the security of our system is jeopardized and a new contract has to be deployed, which requires re-configuration of the resource servers. For this reason, a realization of our system may consider two different keys: one for invoking the security critical functions of the smart contract, and another for specifying which is the former key. The latter key can be the one used for deploying the contract and can be securely stored offline.



Approving other types of identifiers. ERC-721 specifications define that with the `approve` method a token owner can specify another Ethereum address that can manage its tokens. However, in our system a delegee never interacts with the blockchain, therefore the input to `approve` does not have to be an Ethereum address. Other types of delegee identifiers can be considered, such as legacy public keys, or even contemporary forms of authentication such as Verifiable Credentials (VCs) [67].

4.2.5 Related work

As we showed in Section 4.1.3, many research efforts investigate blockchain-based access control, either by defining custom blockchains, such as [57], or by using smart contracts for recording policies and for implementing on-chain PDPs [48, 50, 49, 68, 69]. Our solution does not rely on smart contracts for implementing PDP, which may cause privacy issues, instead it uses the ledger for storing auxiliary information used for validating OAuth 2.0 ACTs.

Other works have also explored integrating blockchain technology into the OAuth 2.0 protocol. For example, Siris et al. [70] demonstrate how to enhance the OAuth 2.0 protocol with blockchain. In particular, they present four distinct applications of blockchain technology in OAuth 2.0 protocol: linking authorizations to blockchain-based payments without online interaction; immutably recording hashes of exchanged information during the OAuth 2.0 flow; encoding policies via smart contracts for transparency and immutability; and using blockchain as a deterrent against DoS attacks through the required transaction costs. Additionally, the study in [71] offers a high-level description of using smart contracts within OAuth 2.0 to enable users to freely select the server that provides authorization to their protected resource. Unlike these approaches, our work specifically utilized blockchain technology to issue and manage OAuth 2.0 ACTs.

4.2.6 Discussion

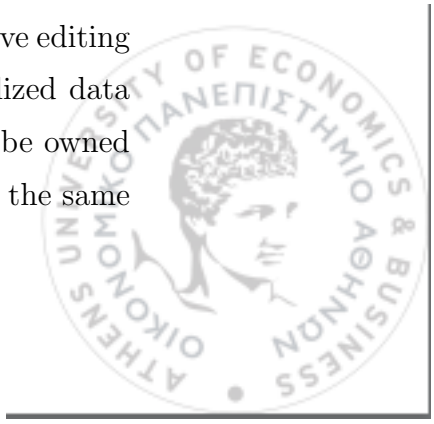
In this work, we demonstrate how to integrate blockchain-based ACTs with the OAuth 2.0 protocol. However, our proposed ERC-721 ACTs can likewise be applied to blockchain-based access control solutions. For instance, the ERC-721 ACTs can be easily incorporated into the access control solution presented in Section 4.1.1, with only minor modifications. In particular, the access control policies, included in the smart contract, would now take the form: access to a resource X is permitted to the user owning the ACT with $token_{id}$ y. Thus, the smart contract, which acts as PDP, invokes the `ownerOf` function, with the appropriate $token_{id}$, rather than checking the ERC-20 token balance as in our previous design.

4.2.7 Conclusions and future work

In this work, we presented the design and implementation of an OAuth 2.0 ACT, backed by blockchain-based smart contracts. Our solution uses Ethereum to store information that can be used for auditing purposes, as well as, for ACT integrity verification. Ethereum smart contracts facilitate revocation, decouple authorization and resource servers, enable token delegation, and create opportunities for exchanging tokens with assets. We believe that our work can be extended towards many directions, including the use of permissioned ledgers, ACT validation using conditions stored in the ledger, privacy-preserving ACT representations, and the application of similar mechanisms for managing authorization grants.

4.3 Consensus-based access control

Emerging systems and applications depart from the traditional centralized host-centric paradigm and allow multiple stakeholders to share their resources in a decentralized manner, as well as to jointly collaborate on the same resources. Examples of such systems include *IoT* data-sharing applications, collaborative editing tools, video conferencing tools, decentralized storage services, decentralized data spaces, and many others. In such collaborative systems, resources may be owned by one entity, stored by another entity, and accessed by a third one. At the same

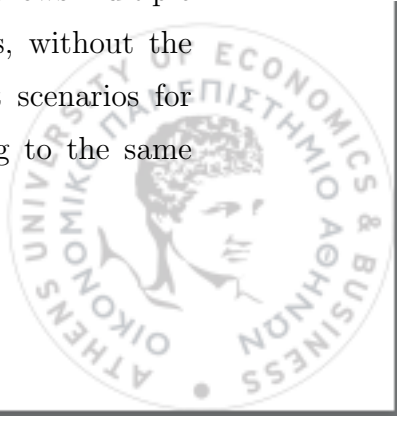


time, a protected resource may be composite, e.g., a smart building that includes multiple devices, where each individual component may have a different owner. Additionally, protected resources may have inter-dependencies, i.e., relations with each other. One such example is healthcare records, which contain personal health information, owned by the patients, and diagnoses, treatments, lab results, etc., made and owned by healthcare providers. Thus, there is not a single entity, who is solely responsible for the access control, but all the involved entities should collaborate to achieve that.

Existing legacy access control solutions, that often *rely on a single trusted entity* for defining, deciding, and enforcing access decisions, are not well suited for the aforementioned cases, since they are too *rigid* and *inflexible*, and *they do not allow and support collaborative definition of access control policies* [72, 73]. At the same time, emerging paradigms, such as the *Zero Trust* paradigm [11], which is built on the assumption that no entity is trusted by default and even if they were previously verified, require continuous authorization. This requirement can overload centralized ASes, disrupt the availability of centralized access control systems, and even result in *DoS*.

For these reasons, new solutions that enable multiple *ASes*, entities responsible for authenticating and authorizing users, and handling access control procedures, to jointly manage access control policies and make access control decisions, are required. However, this collaborative form of access control becomes challenging, in cases, where these entities are not mutually trusted, especially if there is not a single centralized service to facilitate the collaboration. In that case, consensus protocols should be securely applied, and efficient, flexible, decentralized, transparent, and auditable mechanisms are required.

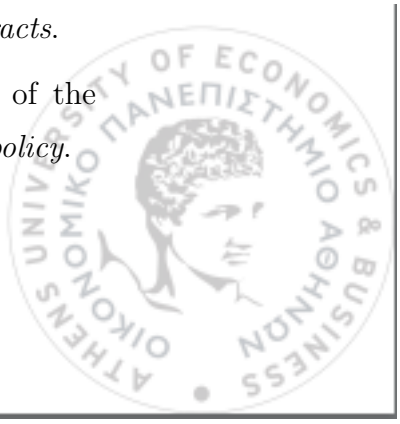
To this end, we explore how *DLTs* can be used to build secure, decentralized, and reliable access control solutions for collaborative environments. In this work, we design an application-independent access control solution that allows multiple *ASes* to cooperate and make access control policies and decisions, without the need for a single centralized authority. We study three different scenarios for collaboration. Initially, we study the case, where all ASes belong to the same



administrative domain (e.g., a company wishing to apply the Zero Trust paradigm), but does not trust a single AS. Then, we extend our design to consider cases, where each AS may belong to a different administrative domain, but all ASes evaluate the same access control policy (e.g., collaboration platforms, where data access requests are approved by all collaborating entities using the same policy). Finally, we further extend the latter design to consider cases, where each AS makes access control decisions based on a "local" policy and the final decision is made by applying a *consensus* protocol among the ASes (e.g., a collaboration platform, where users must prove that they have "approval" to perform an action by at least n other collaborating entities and each entity evaluates a different property of the user).

Our solution addresses the main challenges of the collaborative access control [72, 73], it eliminates the need for reliance on a trusted centralized service and it facilitates co-operation among entities that are not mutually trusted. Furthermore, the ASes in our solution can implement any access control model, from simple *ACLs* to more complex models, such as *OrBAC* and *ReBAC*. For the collaboratively defined access control policies, our solution is expressive enough and flexible, allowing it to adapt to a variety of use cases. In order to achieve our goals, we leverage Hyperledger Fabric [13]. Our Fabric-based implementation, takes measures to decrease the communication overhead towards ASes, as well as to "isolate" them, so that they can be accessed only by the Fabric *peers* of the corresponding organization. Overall, with this work, we are making the following contributions:

- We propose a decentralized consensus-based access control solution tailored to collaborative systems based on Hyperledger Fabric.
- We design, implement, and compare two distinct approaches for implementing consensus-based access control decision.
  - An approach, where consensus rules are encoded in *smart contracts*.
  - An approach, where consensus rules are integrated in a part of the consensus mechanism of Hyperledger Fabric, the *endorsement policy*.



https://doi.org/10.26219/heal.aueb.9346

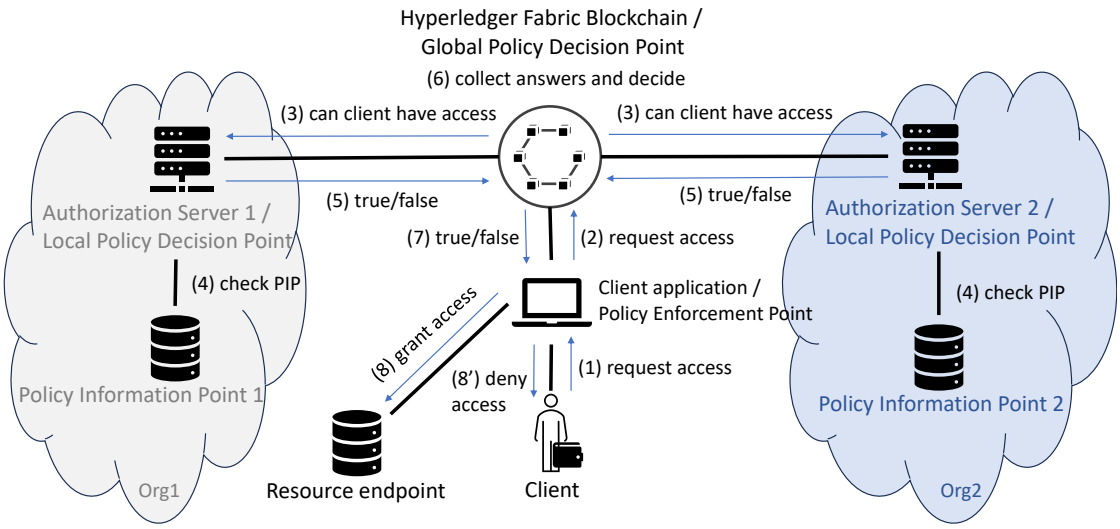


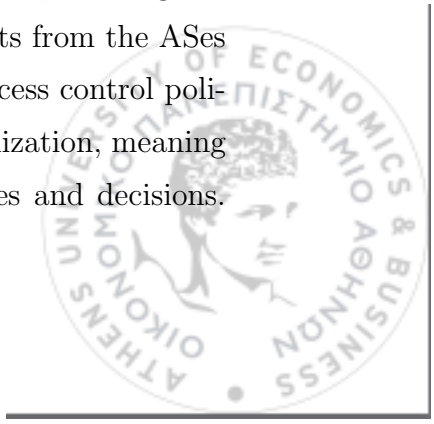
Figure 4.7: Overview of the consensus-based access control solution's architecture.

- We develop proof-of-concept implementations and we evaluate the performance and security properties of both approaches using the Hyperledger Caliper blockchain benchmark tool.

4.3.1 System design

System entities and notation

The main entities of our system, illustrated in Figure 4.7, are *clients*, *ASes*, *PIPs*, a *client application* that acts also as *PEP*, and the *Fabric blockchain infrastructure*. The proposed system shares the main components, protocols, and access control flow common to any access control system, as illustrated in Figure. 2.4 but introduces key enhancements for collaborative systems. In particular, we incorporate two distinct decentralized PDPs; one, involving the ASes, is responsible for decisions based on "local" access control policies, and the other, utilizing the blockchain and smart contracts, collects and aggregates the results from the ASes and decides based on "global" access control policies. "Local" access control policies are defined and validated solely within the scope of one organization, meaning that each organization has its own "local" access control policies and decisions.

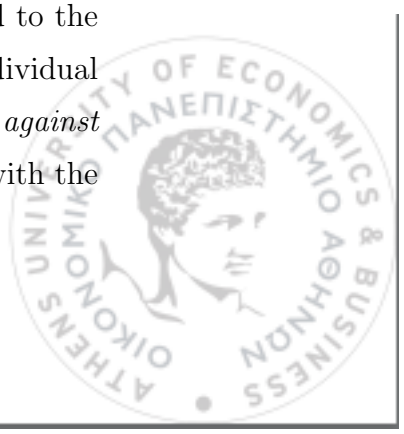


Furthermore, "local" access control policies can be evaluated with any access control model, as long as, the access decision is yes or no. On the other hand, "global" access control policies are defined "collaboratively" by all the involving entities. Thus, the ASes act as "local" PDPs, while the blockchain infrastructure acts as the "global" PDP. The most critical part in our system is the smart contract that receives the clients' access requests and acts accordingly (see Section 4.3.2).

Clients interact with the system through a client application that is configured with the x.509 client's certificate, as it interacts with the blockchain network and the deployed smart contracts on the behalf of the user. The protected resource is hosted in a particular endpoint, which is uniquely identified by a URL, referred to as the $URL_{resource}$ (the endpoint is oblivious to the proposed solution). The ASes are also identified by URLs, referred to as URL_{AS}, and having a structure like, https://as1.company1, depending on the administrative domain that they are owned by. We model administrative domains as Fabric *organizations* and we allow each organization to maintain its own external ASes, which are accessed over HTTPS.

Usage scenarios

We consider three distinct scenarios that vary in complexity. These usage scenarios have been carefully chosen to encompass a spectrum from simpler, more straightforward scenarios to progressively more sophisticated ones. By exploring these diverse scenarios, we aim to comprehensively analyze the applicability, security, and performance of our proposed designs across a wide range of real-world contexts. In the first baseline scenario, we consider the case, where a protected resource is owned and managed only by a single organization. Namely, only one entity is responsible for defining, deciding, and enforcing the access control policies and requests. However, we assume that the organization owns multiple ASes that perform the same validations. All of the ASes should individually respond to the access requests and then the final response is determined based on the individual responses. *This approach is used to enhance the resilience of the system against AS compromises.* The second scenario is similar to the baseline scenario, with the

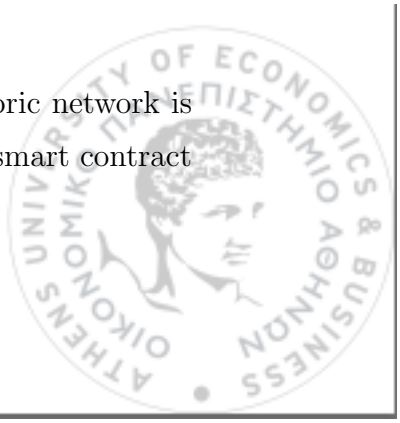


difference that instead of having only one organization owning multiple ASes, we have multiple organizations owning multiple ASes. However, as in the baseline scenario, all the ASes from all organizations perform the exact same validations. *This approach can be used for use cases, where the organizations do not fully trust each other.* The final scenario that we consider includes multiple organizations owning ASes, but instead of performing the same checks each AS group validates different client's properties. For instance, the ASes of one organization verify if the requester is a student and the ASes of another organization if her age is over 20. *One use case for this scenario is collaborative attribute-based access control, where each organization provides and verifies different attributes.*

System overview

In a nutshell, our system (illustrated in Figure 4.7) works as follows. Initially, clients, through the client application, request access to a protected resource, by sending a transaction to the system's smart contract. Then, the smart contract forwards the access request to the appropriate ASes, which fetch the client's attributes from their PIPs and decide on access based on the "local" access control policies. Subsequently, the ASes send back to the blockchain their "local" access control decision. The blockchain aggregates the "local" access responses from all the involved ASes and based on the collaboratively defined "global" access control policy decides for the access and responds back to the client application with the final access control decision. Finally, the client application/PEP enforces the "global" access control decision. Therefore, to approve a resource access request, many ASes, *owned by one or more administrative domains*, depending on the scenario, must cooperate to decide whether clients can be granted access or not, based on *collaboratively* defined access control policies. For the cooperation of the ASes, we propose two different approaches/solutions (see Section 4.3.2). Our system involves the following phases.

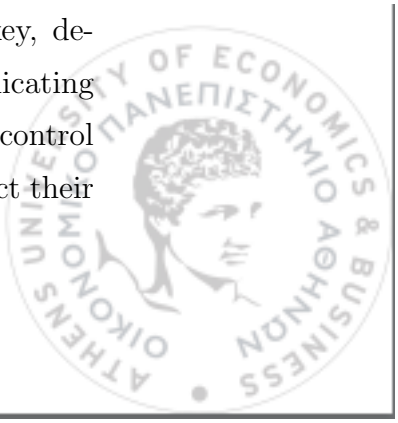
Set up. Our system assumes a setup phase during which the Fabric network is created, the blockchain and its parameters are configured, and the smart contract



is developed and deployed. Moreover, in this phase, the certificates and the keys for the members of the network (peers, orderers, etc.) are created. Furthermore, each organization should configure its ASes with the "local" access control policies. Our proposed system is oblivious to the access control model used in the ASes, and thus, how the access control policies are defined. As we discussed above, depending on the scenario, all ASes may be configured with the same access control policy or each organization may configure its ASes with different access control policies, e.g., AS_1 has a "local" policy that dictates that the client $pub\_key_{client_x}$ can access *data*, but AS_2 is configured with a policy dictating that $pub\_key_{client_x}$ cannot access *data*. Furthermore, during this phase, the organizations should come together and decide the "global" access control policies. After deciding the "global" access control policy, every organization signs it through the blockchain with the form of transactions. Finally, in the setup phase, the client obtains a X.509 certificate, issued by a CA that participates to the Fabric network, in order to be able to interact with the system.

Access request. The client request, which is essentially a transaction to the blockchain, is initiated via the client application. The transaction includes the client's certificate along with the client's digital signature, which is generated using the client's private key, $priv\_key_{client_x}$. The request ends up on the smart contract deployed on the Fabric blockchain. Following, the smart contract verifies the client's signature using the client's public key, $pub\_key_{client_x}$, extracted from her certificate. Then, the smart contract includes the client's public key to the request and forwards it to the appropriate ASes, requesting access for the client with that particular public key.

Access decision. Each AS checks the client's public key and responds to the request, based on its "local" access control policies (two different organization/ASes might have different "local" access control policies for the same public key, depending on the scenario). To evaluate the requests, the ASes are communicating with their PIPs to fetch user's attributes and evaluate the "local" access control policy based on these attributes. Then, they send back to the smart contract their



access decision. Following, the blockchain collects the replies from all ASes and based on a "global" access control policy, defined collaboratively by all involved organizations, responds with the final decision (the exact flow of this process is presented in the following section).

Access enforcement. In this final phase, the client application, which also acts as the PEP, receives the "global" collaborative access decision from the blockchain. Then, based on that decision, it either forwards the request to the resource endpoint, if the client has been granted access to the resource, otherwise it sends back to the client an appropriate error message.

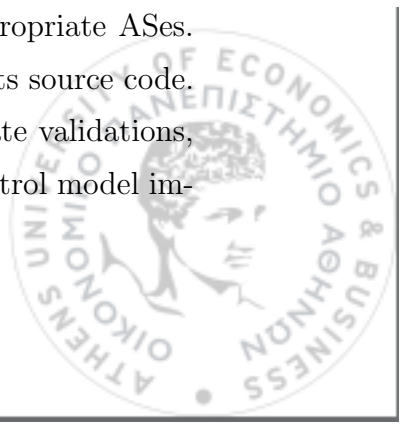
4.3.2 Consensus-based access control

In this section, we present our two different solutions for collaborative access control, namely the cooperation of many mutually non-trusting entities to decide on access control.

Smart contract-based approach

Our first approach, called *smart contract-based approach*, to enable collaborative access control, i.e., allow the cooperation of many ASes having no trust relationships with each other, utilizes smart contracts. In our design, we consider a smart contract, called *Authorization Smart Contract* (ASC) that receives the access request from the client application, which also acts as a PEP (following the reference architecture in Figure 2.4), forwards the request to the appropriate ASes, calculates the "global" access control decision, and responds back to the client the final access decision. The design of our solution is presented in Figure 4.8.

The solution works as follows. Initially, the client, after acquiring her certificate, sends an access request, through the client application, to the deployed ASC. The ASC receives the request and forwards it to all the appropriate ASes. To do so, the ASC should include all the URLs of all ASes within its source code. When an AS receives the access request, it performs the appropriate validations, depending on the implemented access control model (the access control model im-



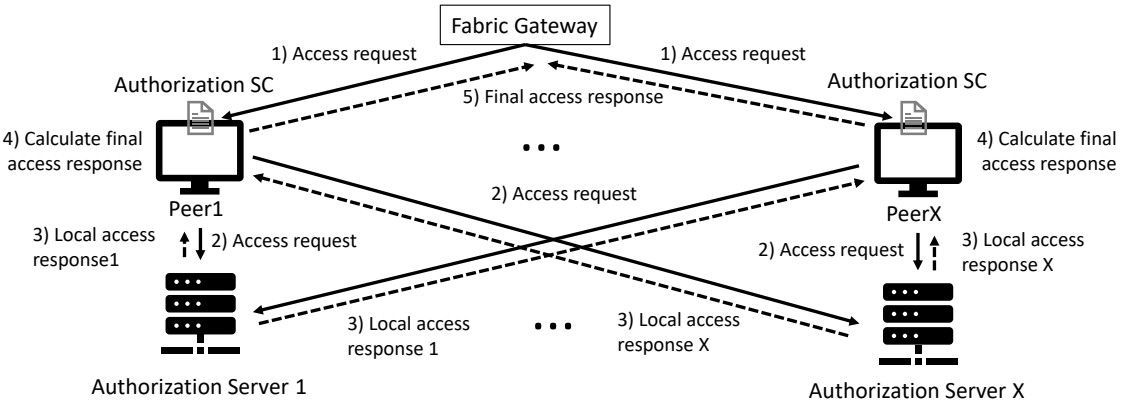
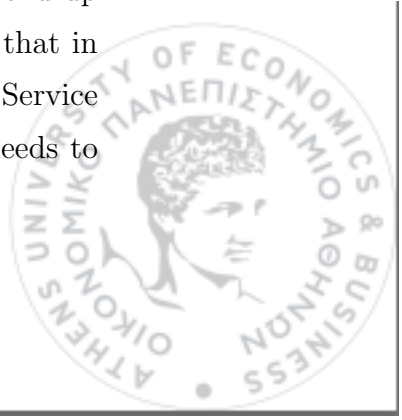


Figure 4.8: Design of the smart contract-based approach for the consensus-based access control solution.

plemented in the ASes is oblivious to our system) and responds back to the ASC accordingly. Following, the ASC receives the responses from all the ASes, aggregates them, and determines the final response based on the collaboratively defined "global" access control policies. Various strategies can be employed for calculating the final response, depending on the use case and the scenario. Examples of such strategies include, deny-overrides, majority voting, or a threshold scheme. In the third scenario, where each group of ASes validates different attributes, the collaboratively defined "global" access control policy can be of type *m out of n*. Finally, the client application receives the response from the blockchain network and its subsequent actions are contingent upon the response (as we show above in the access enforcement process).

We should note here that the ASC is deployed on all peers participating in the Fabric network, meaning that every peer executes the transactions. Thus, in our design, a client request will end up to all the peers that have deployed the ASC and all these peers will send requests to all ASes, as shown in Figure 4.8. For instance, if there are 1000 peers and everyone has to execute the transaction, then all of them will send a request to all ASes. Each AS will eventually end up receiving 1000 requests for the same access request. Therefore, it is clear that in some use cases, where many peers are needed, this can lead to a Denial of Service (DoS) for the ASes. Furthermore, with a design like this one, the ASC needs to



https://doi.org/10.26219/heal.aueb.9346

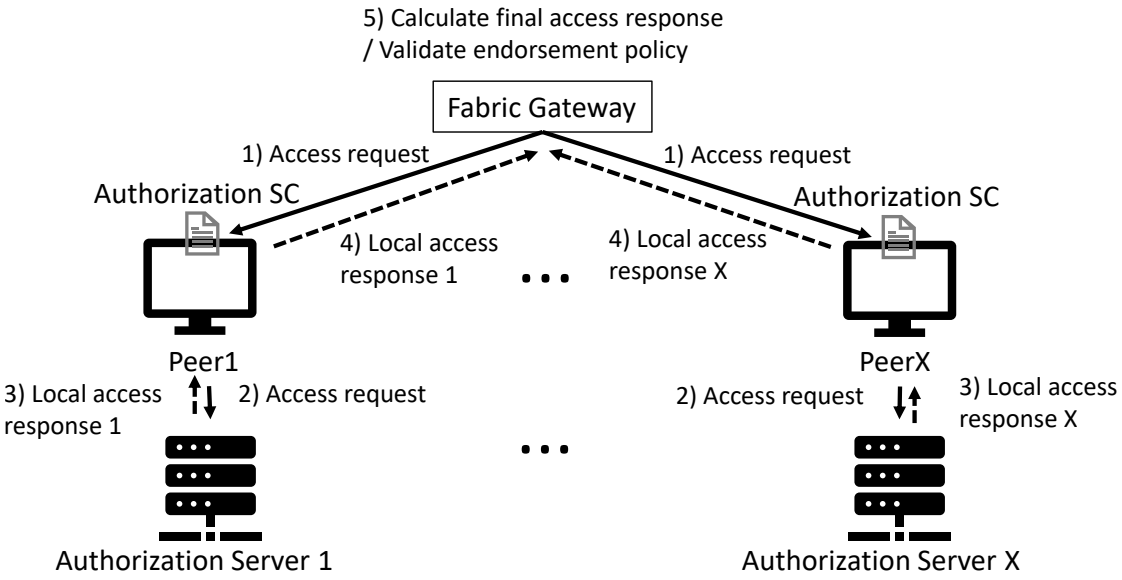
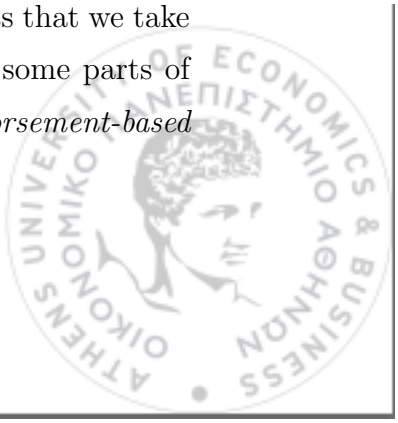


Figure 4.9: Design of the endorsement-based approach for the consensus-based access control solution.

include the URLs of all the ASes. This may be inappropriate in some use cases, as every organization can learn the URLs of other organizations' ASes.

Endorsement-based approach

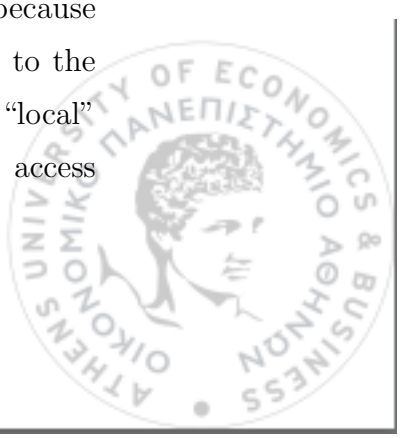
With the smart contract-based approach, we achieve to decentralize the "global" PDP, responsible for evaluating the collaboratively defined "global" access control policies. Furthermore, we enable transparency and auditability regarding the access decision and how it is achieved, by making a smart contract responsible for this process, since the smart contracts are deployed and run in all the involved peers and their execution output is immutably written in the ledger. However, as we showed above, this design may be inappropriate for some use cases. For instance, in the second and third usage scenario, where there are many and possibly untrusted organizations, an organization might not want other organizations to know the URLs of its ASes or its "local" access responses. To address that we take advantage of some of the features of Fabric blockchain, as well as some parts of its consensus mechanism to create a more elaborate design, the *endorsement-based approach*, presented in Figure 4.9.



As shown in Figure 4.9, a peer, instead of sending requests to all ASes, it only sends a request to the AS(es) owned by its organization. To achieve that we exploit the newly introduced feature of Fabric that allows the same smart contract not to be identical across the network members that have deployed it. In particular, we slightly modify the source code of the smart contract, and depending on the organization where it is deployed, we configure it to include only the URL of the AS(es) of its organization and only that (not the URLs of the other ASes). For instance, a smart contract that is deployed on the peers of organization1, includes only the URLs of the ASes owned by that organization, while the same smart contract that is deployed on the peers of organization2 will include the URL of ASes owned by that particular organization, and so on. In this way, we achieve that the smart contract performs different validations depending on the peer where it is deployed. Furthermore, with this design, we enhance the security of the ASes, protecting them from DoS, as they do not receive requests from all the participating peers. Also, we manage to hide their URLs to other organizations' members.

However, even with the aforementioned design, the URLs of the ASes are included in the smart contracts, but only the peers of the same organization can see them. To tackle this challenge, an alternative solution is to include only a generic URL (e.g., https://as.company) within the smart contract and modify the DNS configuration of each individual participating peer, by modifying its hosts file, and associate the generic URL with the IP address of a distinct AS. Due to this customization process, there is no need for including all the URLs of all the ASes within the smart contracts. In this manner, we also manage to hide the location information of the ASes, by excluding completely their URLs from the smart contracts.

In the endorsement-based approach, the smart contract receives responses only from the AS of the peer where it is deployed, rather than from every AS. As a result, the smart contract cannot play the role of PDP anymore, because it does have access to all "local" access responses. Therefore, in contrast to the smart contract-based approach, where the smart contract collects all the "local" access responses, aggregates them, and decides on the final collaborative access



https://doi.org/10.26219/heal.aueb.9346

response, this design exhibits a different flow. Here, the *endorsement policy*, which has been agreed among the involved organizations, serves as the collaboratively defined "global" access control policy and the entity responsible for gathering all the responses from the smart contract and validates the endorsement policy (the "global" collaborative access control policy) is the *Fabric gateway*. For instance, in a scenario with four organizations, where access to the protected resource requires approval from three out of the four organizations, the endorsement policy would be *OutOf(3, 'Org1MSP.member', 'Org2MSP.member', 'Org3MSP.member', 'Org4MSP.member')*, following the Fabric terminology. This means that access is granted if any three organizations approve the request, regardless of which specific organization approve it. As we have already mentioned, for an endorsement policy to be validated, the peers that are dictated by the endorsement policy should execute the transaction and produce identical results. For example, in a scenario with two organizations, each owning one peer, both peers must give a positive response for access to be granted. In this case, the peers of both organizations would execute the transaction, send requests to their respective ASes, and expect both the respond with $true$. If one of them responds with $false$ and the other with $true$, the endorsement validation fails. As a result, the transaction will not be appended to the ledger and the Fabric gateway will return an error message, denying access to the client. If both ASes respond with either $true$ or $false$, the transaction will be validate and written to the ledger. If the result is $true$ the access is permitted, otherwise access is denied.

As we mentioned in Section 2.1.2, the endorsement policy is validated twice, before the transaction is committed on the ledger. First, it is validated by the Fabric gateway, which is a single entity. Then, it is validated by all participating peers. Therefore, the PDP, which is responsible for validating the collaboratively defined "global" access control policy, is not a single entity, i.e., the Fabric gateway, but all the participating peers play this role.

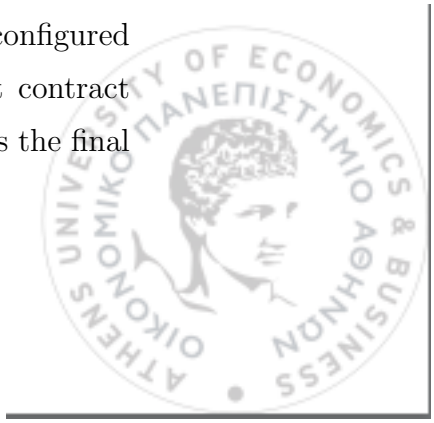
4.3.3 Implementation and evaluation

Implementation

We developed proof-of-concept implementations of the proposed system, as well as, the two distinct approaches for collaborative access control, namely the smart contract-based approach and the endorsement-based approach, to evaluate their feasibility and performance. In both implementations, the client application is developed using the Fabric Client SDK, enabling interaction with the blockchain network. For the evaluation, we utilize a baseline network, which is structured as follows:

- One orderer organization (org0) with three orderer nodes.
- Three peer organizations (org1, org2, and org3) having two peers each.
- Three ASes, each independently validating access requests, based on "local" access control policies.

In the smart contract-based approach, a single smart contract handles the access control process. Since no single AS or organization is inherently trusted, each organization deploys the same smart contract that forwards every access request to all three ASes. Thus, every AS receives six identical requests, as there are six peers on the network. Each AS evaluates the request and returns its decision ('allow' or 'deny') based on its "local" policy. The smart contract aggregates these responses and applies, in this case, a majority voting policy; if two out of the three ASes approve the request, access is granted. The endorsement-based approach leverages Fabric's endorsement policy to determine the final access decision. In the endorsement-based approach, each organization only communicates with a single AS, exploiting Fabric's capability to run slightly different smart contracts across organizations. Specifically, in our setup, org1 only communicates with AS1, org2 with AS2, and org3 with AS3. Each smart contract instance is configured to contact only its respective AS, and we do not have a single smart contract aggregating access decisions. Instead, the endorsement policy determines the final



outcome, which in this case is the majority endorsement policy, requiring at least two out of three ASes to approve a request for access to be granted.

In regards to the access control model implemented in the ASes' side, we implemented a basic decision model, where each AS generates a random decision ('allow' or 'deny') based on 0.5 probability threshold, thus our implementation does not include communication with the corresponding PIPs or PAPs. This model abstracts "local" policy and access control model specifics, such as how the PAP and the PIP are implemented, how policies are stored, and so on. We emphasize that the proposed solution is agnostic to specific PAP and PIP implementations, making it flexible in this regard. Additionally, we deliberately exclude the overheads of PAP and PIP from our evaluation to focus solely on the proposed solution.

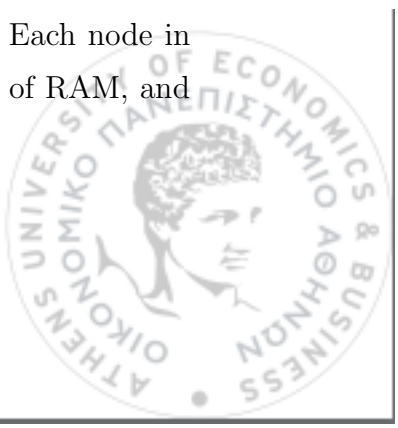
Performance evaluation

To assess the system's performance, we utilize the following metrics and properties:

- *Latency* (s): We measure the time required to respond to an access request, which includes the execution of an issued transaction and the communication with the ASes.
- *Scalability*: We assess the scalability, by gradually increasing the number of peers to the peer organizations and evaluate the system's performance, i.e., latency and throughput, by conducting a set of similar experiments.

The baseline network was implemented using a Kubernetes cluster consisting of four nodes; one master node and three worker nodes. The master node orchestrates the Kubernetes environment, serving as the control plane, and hosting the three ASes. The blockchain network is deployed on the other three nodes. To ensure more reliable results in next-generation networks, we deployed the cluster within a 5G testbed, provided by the 6G-SANDBOX project.[8] Each node in the testbed is equipped with a CPU having eight processors, 16GB of RAM, and 100GB memory.

[8]https://6g-sandbox.eu/



To evaluate the aforementioned metrics, we utilize Hyperledger Caliper, a blockchain benchmark tool.[9] Hyperledger Caliper allows us to control the rate at which transactions are sent to the system, thus enabling us to analyze how the system handles increasing workloads. For our experiments, we utilize the fixed-rate controller, which sends transactions at a constant, predefined rate, until the test's duration is reached. To optimize performance, we modify a few key configurations across Fabric, Hyperledger Caliper, and Kubernetes. A few of these include increasing the peer's cache size from the default 64MB to 128MB, and extending the endorsement timeout (how long the gateway waits for a response from endorsing peers) from 30 to 120 seconds.[10] Additionally, we modify the concurrency limit related parameters to increase the number of concurrently running requests to a service on each peer.

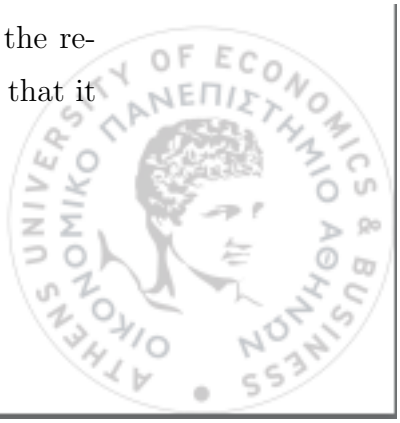
Each experiment consists of ten test cycles, with each cycle gradually increasing the send rate, measured in *TPS*. We start at 10 TPS and increase it by 10 TPS per test cycle, reaching a maximum of 100 TPS in the final test cycle. Each test cycle runs for a duration of 100 seconds. The purpose of this experiment is to evaluate how the system handles increasing transaction rates, eventually identifying the system's limits. We measure the number of successful and failed transactions processed within each test cycle and analyze the corresponding latency. The final three test cycles, and especially the last one at 100 TPS, serve as a stress test, revealing potential performance bottlenecks.

Smart contract-based approach. The results of the experiments conducted for the smart contract-based approach are depicted in Figure 4.10, where we present latency with and without communication with the ASes. In the second case, where there is no communication with the ASes, we only measure the time required for a transaction to be processed by the blockchain network, so as to estimate the overhead added by our proposed solution.

In the first seven test cycles, the system processes transactions at the requested rate without failures with 0.11 seconds average latency, indicating that it

[9]https://www.lfdecentralizedtrust.org/projects/caliper

[10]https://hyperledger-fabric.readthedocs.io/en/release-2.5/performance.html



https://doi.org/10.26219/heal.aueb.9346

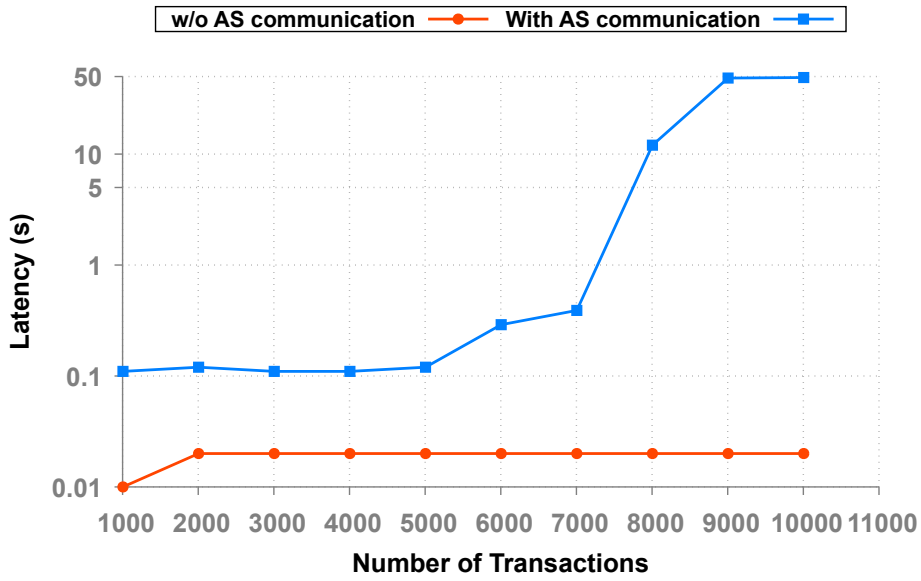
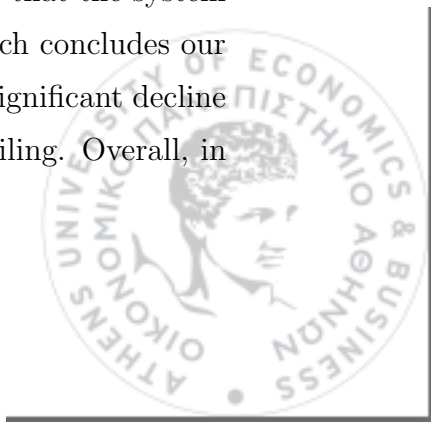


Figure 4.10: Experiment results for the smart contract-based approach in the consensus-based access control solution: with and without communication with the Authorization Servers.

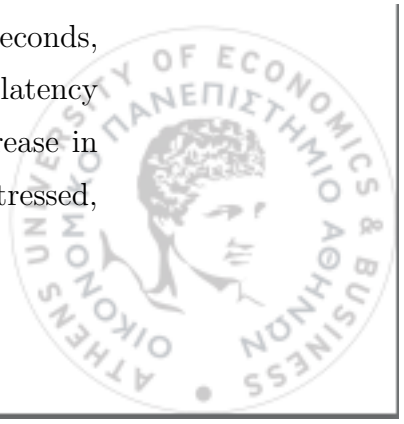
is operating within its capacity. However, latency begins to increase in the sixth and seventh test cycles, with an average value of 0.34 seconds. Although the increase may appear minor at first, latency in the sixth test cycle is nearly three times higher than in the preceding cycles, and in the seventh test cycle, it reaches four times the values recorded in the first five cycles. These observations suggest that the system is approaching its performance limits, with the seventh test cycle serving as an early indication of the performance degradation that follows. In the eighth test cycle, latency rises considerably to 12 seconds, which compared to 0.12 seconds in the earlier test cycles is a 100 times more. Additionally, 873 out of 8005 transactions (10.91%) fail, indicating that the system is stressed. In the ninth test cycle, latency also increases significantly to 48.47 seconds, and 8589 out of 9005 transactions (95.38%) fail, demonstrating even more clearly that the system is struggling to process the workload. In the final test cycle, which concludes our stress test, the system is pushed beyond its limits, resulting in a significant decline in performance, with 9897 out of 10005 transactions (98.92%) failing. Overall, in



the final three test cycles, latency remains excessively high and the majority of transactions fail. These results show that while the system maintains stable performance under moderate loads, beyond a certain point, latency rises considerably, and transaction failures increase.

Note that the system does not reach its peak due to the blockchain itself. To confirm this, we conducted an additional experiment in which the system operated without communication with the ASes. The results demonstrate that under the same conditions, the blockchain alone does not reach its processing limits during the experiment. Across all test cycles, no transaction failures occur and the blockchain introduces minimal overhead, contributing at most 0.02 seconds of additional latency, which also remains stable across the test cycles. The minimal additional latency confirms that blockchain not only can be integrated into access control solutions without becoming a performance bottleneck, but it also provides the ability to record and verify access decisions, ensures traceability, transparency and auditability, and thus accountability and trust. Therefore, we observe that in the complete experimental setup the bottleneck arises from the communication with the ASes, as each AS must respond to numerous separate requests per transaction and processes these requests sequentially. However, this limitation is implementation-specific and it can be addressed by leveraging multi-threading for parallel request handling or by implementing load balancing mechanisms. This could improve performance. However, we should note that the communication with the ASes is not specific to our design and also occurs in legacy systems.

Endorsement-based approach. The results of the experiments conducted for the endorsement-based approach are depicted in Figure 4.11. In these experiments, as the send rate gradually increases, the system initially maintains stable latency, with an average of 0.02 seconds, and successfully processes transactions at the requested rate without any failures during the first four test cycles. Between the fifth and seventh test cycles, although latency remains below 0.15 seconds, the average value has increased to 0.08 seconds, which is four times the latency recorded in previous test cycles, serving as an early indicator of the decrease in performance that follows later. In the final three test cycles, the system is stressed,



https://doi.org/10.26219/heal.aueb.9346

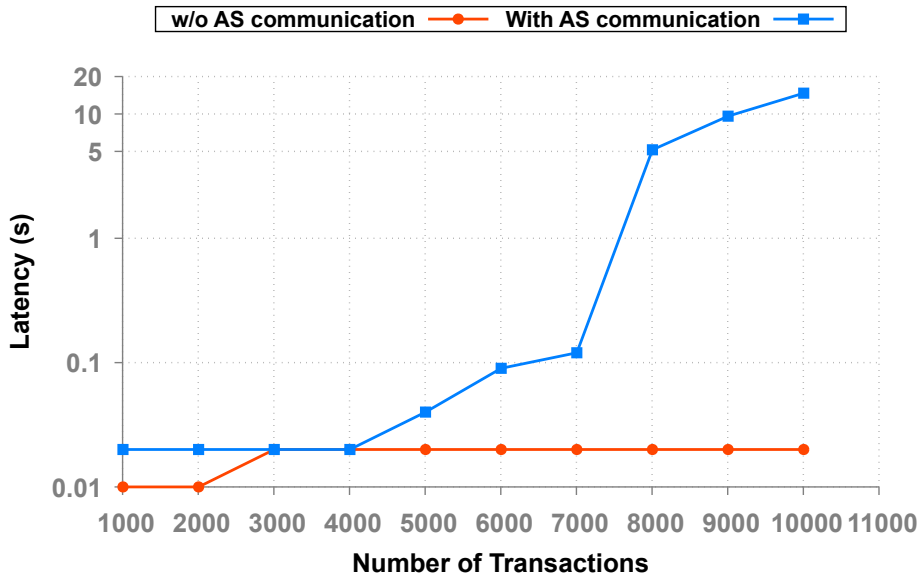
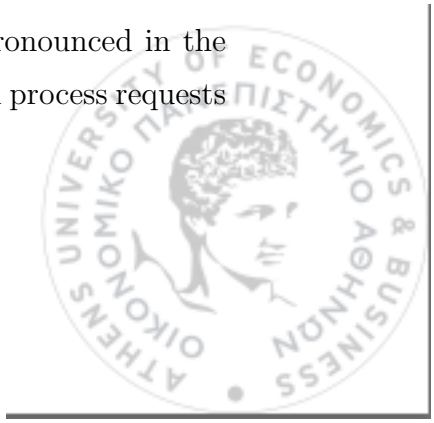


Figure 4.11: Experiment results for the endorsement-based approach in the consensus-based access control solution: with and without communication with the Authorization Servers.

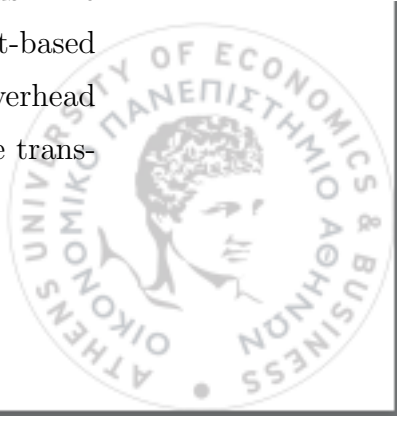
underperforming resulting in increased latency, ranging from 5.15 to 14.7 seconds. Moreover, in the final three test cycles, we begin to observe failed transactions. Specifically, in the eighth test cycle 4.66% of the total 8005 transactions have failed, in the ninth test cycle, we have 18.89% of the total 9005 transactions failing, and in the tenth test cycle 29.68% of the total 10005 transactions failed.

We repeat a similar experiment for the endorsement-based approach without communication with the ASes to record the blockchain induced overhead. The results closely resemble those observed in the similar experiment for the smart contract-based approach, confirming again that the system's performance limitations are not due to the blockchain itself. Throughout all test cycles , no transaction failures occur, and the blockchain introduces only minimal overhead. Specifically, the additional latency remains stable across all cycles and does not exceed 0.02 seconds. While the AS communication bottleneck is less pronounced in the endorsement-based approach, the same issue remains, as ASes still process requests sequentially.



Scalability. To assess the scalability of the proposed solutions, we extend the baseline network by gradually increasing the number of peers to each peer organization. The experiments consist of seven test cycles, with a 100-second duration each and 80 TPS send rate. We assess the scalability of the system on just one test cycle of the previous experiments, the one with a 80 TPS send rate, since it marks the point, where the system begins to collapse for both approaches. The scalability experiments allowed us to evaluate whether adding more peers would improve performance, reduce latency, and enable more efficient workload handling, as well as to determine which approach scales better. Note that our implementation utilizes the majority endorsement policy, where the majority of the organizations have to endorse the transaction and any member from each of these organizations has to sign it. This means that the scalability conclusions presented in this section are specific to this configuration (the results may vary under different endorsement policies, particularly those requiring approval from all peers in the network). It is also important to note that in this experiment, we did not add more orderers. Additional orderers are typically introduced in larger and more complex topologies to enhance fault tolerance and improve overall network resilience. The results are depicted in Figure 4.12.

The smart contract-based approach shows a gradual improvement as more peers are added. Specifically, with 6 peers (two peers in each organization), we have 12.03 seconds latency, with 9 peers, latency drops to 9.2 seconds and with 12 peers in the final test cycle, we achieve 3.19 seconds. Although in the final test cycle latency remains relatively high, it still shows significant improvement, as we increase the number of peers. The endorsement-based approach achieves significantly better latency. In the first test cycle, having 6 peers (two peers per organization), latency is 5.15 seconds, in the fourth, with 9 peers, we achieve 0.11 seconds, which is an acceptable value for a test cycle during which the system was previously stressed, and in the final test cycle, latency drops to 0.03 seconds. The differences in latency values further reflect the advantage the endorsement-based approach has over the smart-contract-based approach in regards to the overhead introduced by AS communication. This is also confirmed by observing the trans-



https://doi.org/10.26219/heal.aueb.9346

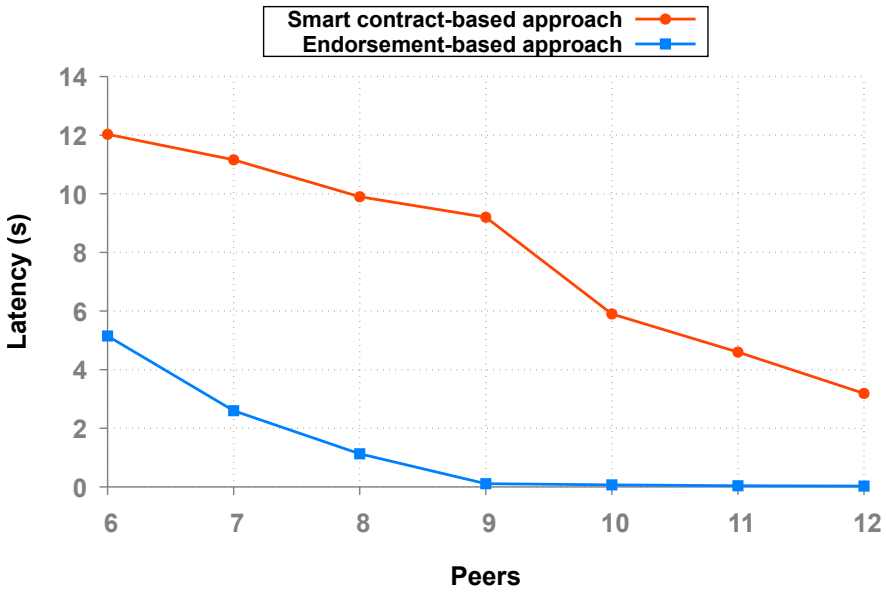
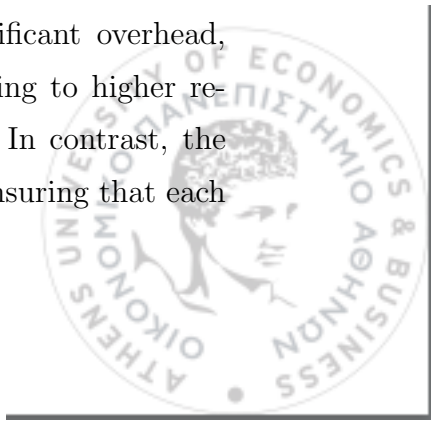


Figure 4.12: Latency with varying numbers of peers for both approaches of the consensus-based access control solution.

action failures for each approach. The endorsement-based approach started with 373 failed transactions, while the smart contract-based approach with 873 failed transactions, in a test cycle where they would need to execute 8005 transactions. The smart contract-based approach managed to improve this value to 658 by the fourth test cycle, where an additional peer was introduced to every organization in the network, and 227 in the final test cycle, where each peer organization had three peers. In contrast, the endorsement-based solution had no failed transactions after the fourth test cycle, where one additional peer was introduced across all peers in the network.

Comparison. A major differentiating factor between the two approaches is the frequency of the communication associated with the ASes. In the smart contract-based approach, each transaction results in multiple identical requests being sent to the ASes; one per peer organization. This introduces significant overhead, as each peer and each AS must process multiple requests, leading to higher resource consumption and potential DoS risks under high loads. In contrast, the endorsement-based approach eliminates redundant requests by ensuring that each

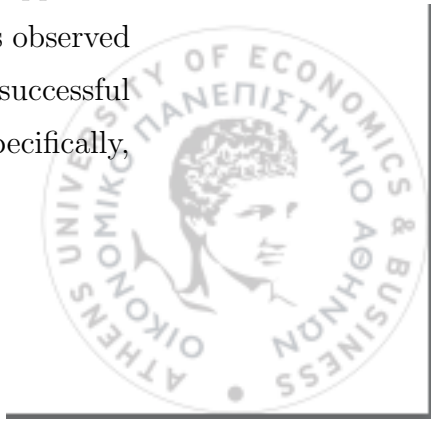


Approach	**Test cycle**	**Latency**	**# Successful**	**# Failed**
Smart contract-based	8	12.03	7132	873
	9	48.47	416	8589
	10	48.98	108	9897
Endorsement-based	8	5.15	7632	373
	9	9.60	7304	1701
	10	14.7	7036	2969

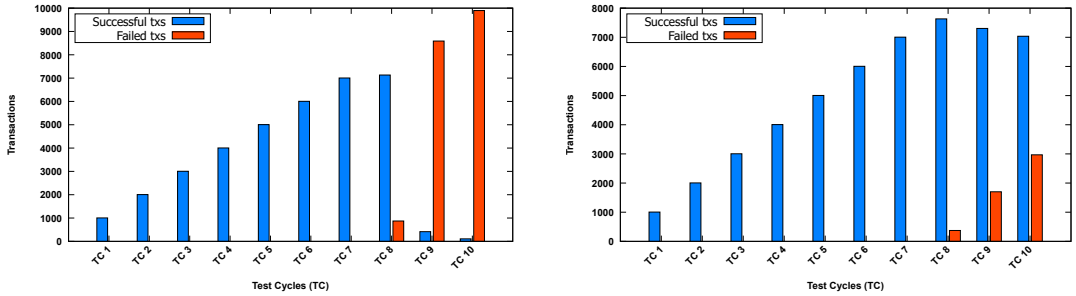
Table 4.6: Experiment stress test comparison table for smart contract-based and endorsement-based approach of the consensus-based access control solution.

organization only communicates with its own AS, significantly reducing the load on ASes and by extension the peers (this is also shown in Figures 4.8 and 4.9).

We compare the final three test cycle results for the fixed-rate experiment for both approaches in Table 4.6. Both approaches initially perform similarly, processing transactions at the requested rate with stable latency. Notably, the endorsement-based approach presents a lot less latency than the smart contract-based approach. As the send rate increases, in the smart contract-based approach, signs of stress begin to appear in test cycle 6 and 7, with a sharp increase in latency. The final three test cycles show significant performance decline, where the failed transactions increase. At 100 TPS, 98.92% of transactions failing and latency recorded at 48.98 seconds. The endorsement-based approach follows a similar trend but exhibits better resilience under stress, with significantly less failed transactions. While latency also increases in the final three test cycles, it remains consistently lower than in the smart contract-based approach. For instance, at 100 TPS, failed transactions account for 29.68% instead of nearly 99% in the smart contract-based approach. Additionally, latency in the endorsement-based approach peaks at 14.7 seconds, which is significantly lower than the 48.98 seconds observed in the smart contract-based approach. The difference between failed and successful transactions across the test cycles is also depicted in Figure 4.13. Specifically,



https://doi.org/10.26219/heal.aueb.9346



(a) Failed and successful transactions for the smart contract-based approach.

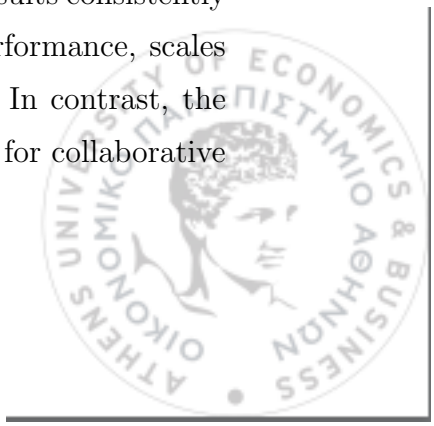
(b) Failed and successful transactions for the endorsement-based approach.

Figure 4.13: Failed and successful transactions for the smart contract-based and the endorsement-based approach of the consensus-based access control solution.

Figure 4.13a shows the transactions for the smart contract-based approach and Figure 4.13b shows the transactions for the endorsement-based approach across test cycles. These results suggest that the endorsement-based approach maintains more stable performance as the workload increases, largely due to its reduced communication overhead with the ASes.

The aforementioned conclusion is further confirmed in the scalability experiments, where we observe that the endorsement-based approach manages to achieve the requested TPS and maintain low latency after the fourth test cycle, where one additional peer was introduced to each organization.

In a real world deployment, performance would likely be better than in our experimental setup, where the ASes share resources with each other and the Kubernetes control plane, potentially impacting response times. Additionally, in a production environment, we would have dedicated nodes with higher computational power, we would utilize multi-threading and employ load balancing techniques, as well as mechanisms to provide high availability, such as autoscaling to meet demand, etc. However, even with the limitations of our setup, our results consistently show that the endorsement-based approach maintains stable performance, scales more efficiently, and handles higher workloads more effectively. In contrast, the smart contract-based approach, although still a suitable solution for collaborative

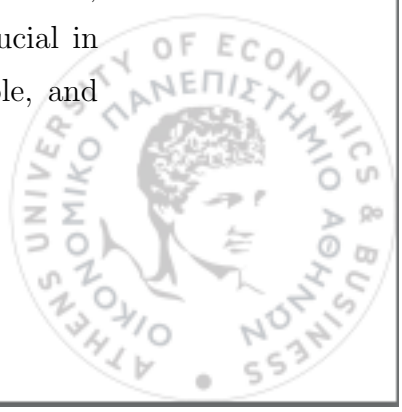


access control, is impacted by the higher communication overhead with the ASes, which limits performance under stress and can become a bottleneck if they are not implemented efficiently.

Security Evaluation

Assumptions. For our security analysis, we make the following assumptions. During the setup phase, each client, through the client application, acquires a X.509 certificate issued by a CA of Fabric network. We assume that clients manage their certificates in a secure manner, utilizing a secure storage mechanism, such as a secure wallet. Moreover, we are not concerned with malicious services that can change the flow of the protocols. Specifically, we assume that the client application/PEP, which is a centralized service in our proposed system, is always available and operates reliably. For example, if the final collaborative access decision is to grant permission, then it cannot deny it. Additionally, we assume that the communication between the client application and all other entities in the system occurs over a secure communication channel. Thus, network-based attacks, such as traffic analysis, etc., are out of the scope of this paper.

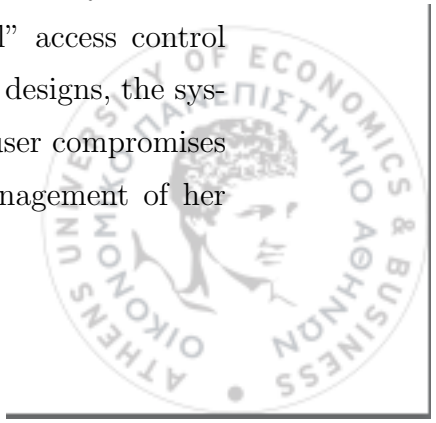
Security requirements. To enable an efficient and secure access control solution, our presented system should fulfill specific security requirements. These include confidentiality, integrity, availability, transparency, non-repudiation, and privacy. *Confidentiality* (or safety property) is a fundamental requirement for a secure access control system, as it ensures that only legitimate clients can access the protected resources. *Integrity* is also essential to ensure that access control policies and logs cannot be tampered with. This is critical, especially in collaborative environments, to maintain trust in the system. Furthermore, the *availability* property ensures that the system remains constantly operational. Regarding privacy, the access decisions or policies should not leak information about clients, ASes, interactions, etc. Lastly, *transparency* and *non-repudiation* are crucial in collaborative environments, so the access decisions can be audited, visible, and easily verifiable.



Threat model. Initially, for the members of the Fabric network, the organization's peers, we consider the *honest-but-curious* attacker model. This model does not want to create a malicious activity in the system, but in addition, the attacker is passive following honestly the protocols steps and at the same time aims to obtain information beyond their own outputs. On the other hand, we consider two types of attacks, collaborative attacks and individual attacks. Collaborative attacks involve multiple malicious users working together, for example, conducting DDoS attacks to disrupt service availability, while individual attacks include only one malicious user trying to compromise entities in the system, such as privileged insider threats. In both scenarios, the malicious users have two purposes, either deny access to legitimate users or lead the system to allow access to unauthorized users.

Informal analysis. In our designs, an honest-but-curious peer can only acquire knowledge of the final access decision. Specifically, in the second design, a peer cannot even learn the endpoints (URLs) of other organizations' ASes, nor the individual responses from those ASes. In both designs, the system limits information exposure by concealing URLs and endpoint details, thereby protecting sensitive information even from honest-but-curious peers within the network. Therefore, in addition to preventing leakage of private information, such as the URLs of an AS, our system manages to protect crucial components of the system, by not disclosing information to mutually non-trusting entities, e.g., other collaborating organizations.

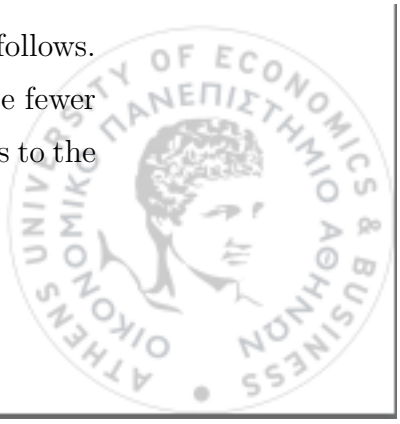
Furthermore, in our designs, the compromise of an AS or a peer does not compromise the client's access. In particular, if an adversary compromises an AS, she cannot grant or deny access, prevent clients from gaining access, or modify the access decisions (given the appropriate collaboratively defined "global" access control policy). The only action the adversary can take is to alter only her own organization's access response or inspect and modify the "local" access control policies, without however impacting the broader system. In both designs, the system is resilient even in privileged insider threats. If a malicious user compromises an admin peer, i.e., a peer with elevated privileges for the management of her



organization, she cannot alter the smart contract, modify the ledger, or override the "global access control" policy. Any such changes would require the approval of all participating organizations (or a defined threshold, depending on the use case). To achieve the aforementioned, the adversary would need to compromise more than the defined threshold (depending on the use case). For instance, if the "global" access control policy (or endorsement policy in the second design) is based on majority voting, then the adversary would need control of more than 50% of the peers or ASes. In such a case, the adversary could gain full control of the blockchain, alter it, and potentially deny access to legitimate users or grant access to malicious ones.

Additionally, our system demonstrates increased availability and robust resilience against DDoS attacks. The PIPs are oblivious to the system and only the ASes from the same administrative domain know their URLs, securing them from malicious users. Moreover, the URLs of the ASes are not stored in the smart contract or in the ledger in the second solution. Even if an adversary compromises a peer, she will only learn the URL of one AS. Similarly, if she compromises an AS, she will learn the URL of one PIP, but not the URLs of the PIPs or ASes of other organizations. Thus, the ASes and the PIPs are secured from clients or members of other organizations. Even if some ASes or PIPs are compromised or unavailable, the system will remain operational and available, as long as the number of unavailable servers is below the defined threshold in the use case. Furthermore, the PDP is fully decentralized and does not present a single point of failure, as it is implemented in a smart contract. Even if some peers are disconnected or compromised, the PDP will continue to be available. Our system does not involve a single point of failure, except for the PEP, which is a single point of failure, as in any other legacy access control system. However, the PEP is assumed to be secure and trusted. Through these mechanisms, our system effectively mitigates the impact of malicious attempts to disrupt service availability.

Our proposed system preserves the desired security properties as follows. Regarding confidentiality, as discussed above, if malicious users compromise fewer ASes or peers than the defined threshold, only legitimate users will get access to the

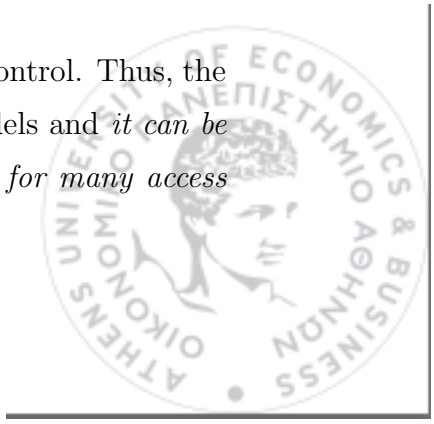


protected resource, assuming the client application is secure. Integrity is ensured through blockchain's immutability, since every transaction is immutably written on the ledger. Immutability guarantees that the "global" access control policy cannot be altered or deleted by malicious users, ensuring integrity across all peers. Transparency and non-repudiation are achieved through the inherent properties of the blockchain technology. The transparent nature of the blockchain allows everything recorded on the ledger to be audited at any time, by any member of the blockchain network, ensuring non-repudiation. However, too much transparency can lead to leakage of private information. In our second design, we mitigate this risk by excluding sensitive data, e.g., URLs of all ASes, from the smart contract, thereby ensuring privacy. Finally, our proposed system ensures policy enforcement consistency, since access control decisions are made through smart contracts running on multiple peers and are immutably recorded on the ledger (in the first design).

4.3.4 Discussion

In the previous sections, we presented two approaches for consensus-based access control tailored to collaborative systems of independent entities. With our solutions, we allow semi-trusting and untrusting entities to collaborate for providing access to shared data or resources. With the smart contract-based approach, we utilized smart contracts to completely decentralize the PDP, while with the endorsement-based approach, we achieve embedding the access decision on the actual consensus mechanism of the Fabric blockchain, which is already trusted by all the involved entities and does not depend on one of the parties The proposed solutions enable efficient and effective access enforcement by just querying the ledger. The time required for reading a record from the ledger is negligible and it is analogous to a query to an AS. Thus, the blockchain does not add overhead in this process.

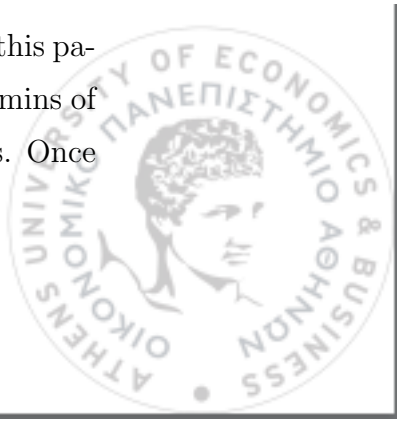
In this paper, we are focusing on the collaborative access control. Thus, the proposed system is transparent to the "local" access control models and *it can be used as an underlying framework for collaborative environments for many access*



control models. This flexibility is essential for enabling effective collaboration, as it offers *interoperability* among different access control models, allowing different organizations to implement different access control models. Each organization can retain its preferred access control model without enforcing a single, uniform system across all organizations. For instance, one organization can implement ABAC for its "local" access control and another organization can use RBAC. Such adaptability is especially beneficial given the rapid development of advanced access control systems, such as CapBAC, UCON, and ReBAC. Furthermore, the participating organizations do not need to modify their existing access control systems. Instead, they can leverage their existing implementations as their "local" access control procedures. Therefore, our approaches offer a flexible, non-intrusive framework that facilitates seamless interoperability among organizations with varying security preferences and requirements.

Additionally, our consensus-based access control solution can serve many use cases and scenarios, since the different approaches address a wide range of requirements. However, in the endorsement-based approach, the endorsement policy is used as the access control policy. That means that for all users and for all shared resources, we have one collaboratively defined "global" access control policy, since there is one endorsement policy per smart contract. This might be inappropriate for some use cases and scenarios, but it can addressed by deploying different smart contracts for different resources. Another solution that allows us to define more fine-grained policies per users or resources, is to take advantage of a new feature of Fabric [24] that allows us to set endorsement policies at the level of state variables of a smart contract. Furthermore, our first solution can easily be implemented with minor modifications in any other blockchain that supports the execution of smart contracts, such as the public Ethereum blockchain [12]. This can serve other use cases beyond those considered here, since in Ethereum openness is a critical property, while performance and costs are secondary concerns.

Regarding the administration of the "global" access control policies, this paper does not focus on how they are defined. Instead, we assume that the admins of each organization cooperate to define these "global" access control policies. Once



https://doi.org/10.26219/heal.aueb.9346

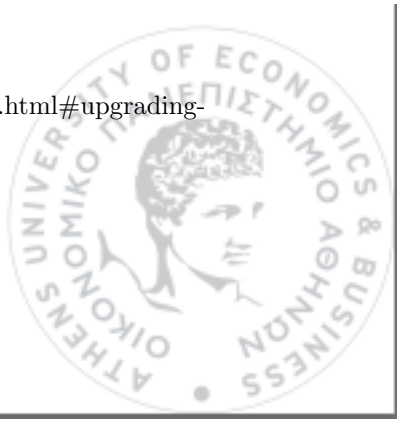
the "global" access control policies are established, they are encoded either within the smart contract or as the endorsement policy in Fabric. This ensures that the "global" access control policies will be consistently respected, as the underlying blockchain mechanisms guarantee their enforcement. In addition, after the definition of the "global" access control policies, no organization can modify them independently, as they are encoded within the smart contract or as endorsement policies. Any modification requires cooperation from all participating organizations. This establishes trust among the participating organizations, even in cases, where they do not mutually trust each other.

Except for the aforementioned, our system has many desirable usability properties. New organizations, new ASes, new PIPs, or new access control policies can seamlessly be added. Also, existing access control policies can easily be modified or removed, by exploiting a feature of Fabric, which allows us to upgrade deployed smart contracts at runtime.[11] In addition, all these changes are transparent to the client application and the PEP. Nothing has to be changed on this entity.

Paci et al. [72] and Tolone et al. [74] in their surveys on access control for collaborative systems, summarize the main requirements that access control systems should fulfill for such environments. Paci et al. [72] highlight key requirements, including policy specification, governance, usability, and transparency, while Tolone et al. [74] present higher-level requirements, as follows.

- Access control must be applied at a distributed platform.
- Access control models should be generic enough to accommodate various environments.
- Scalability, as the number of operations is much richer in collaborative environments.
- Privacy at varying levels of granularity.

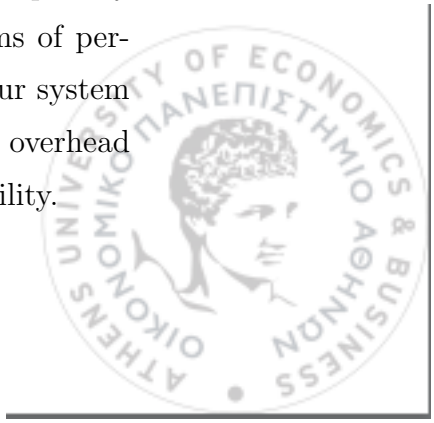
[11] https://hyperledger-fabric.readthedocs.io/en/release-2.2/deploy_chaincode.html#upgrading-a-smart-contract



- Transparency.
- Access control models must allow high-level specification of access rights, managing the complexity that collaboration introduces.
- Access control models must be dynamic, to be able to specify and change policies at runtime.
- Performance and costs should be kept within acceptable bounds.

In our work, we do not focus on policy definitions, assuming instead that each organization defines its own policies, while the collaboratively defined "global" access control policy is feasible and tailored to the specific use case. For instance, in a biomedical data sharing scenario, a patient's individual access preferences may weigh more than a hospital's access decision. Leveraging smart contracts or the endorsement policy to define "global" access control policies provides sufficient expressiveness to serve such scenarios and many others. However, our proposed solution meets all the other identified requirements. In terms of governance, our system facilitates the collaborative administration of shared resources, allowing joint management and decision making. Regarding transparency, our system allows all the organizations to understand collaborative decisions and their effect, as all organizations can audit the blockchain. As shown above, our solutions have several usability properties too.

Our system is also in line with Tolone et al. [74]'s summarized requirements. Built on a blockchain infrastructure and having smart contract or the consensus protocol acting as PDP, our solution inherently supports decentralization. Our distributed design is essential in collaborative contexts. Furthermore, as we showed above, our solution is flexible and generic enough, supporting various access control models that an organization may wish to use. Moreover, as we demonstrated in the previous section, our solution is dynamic, usable, transparent, and ensures privacy for sensitive information (e.g., URLs of endpoints, ASes, etc.). In terms of performance and scalability, our performance evaluation has showed that our system introduces minimal overhead in the decision phase, with no additional overhead during the enforcement phase, demonstrating both efficiency and scalability.

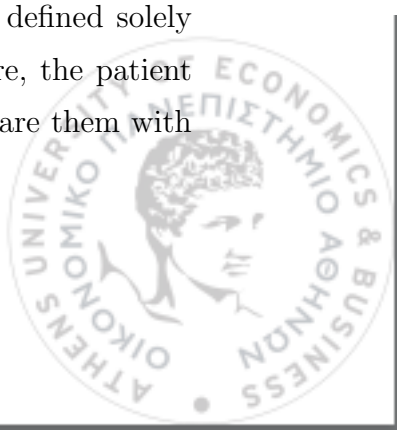


It is important to note that in the endorsement-based approach, we leverage a unique feature of Fabric that allows smart contracts not to be identical across the peers on which it is deployed. In particular, in our design, this feature enables smart contracts to request different ASes, depending on their organization. This results in the execution output potentially differing across the peers. This offers much flexibility and many advantages for our design. However, it also requires careful definition of the endorsement policy (or "global" access control policy) to ensure that the overall consensus process remains consistent and secure despite the differences in individual executions.

Real life use cases

A striking example that highlights the need for collaboration in access control is the case of healthcare data sharing. In such cases, sensitive patient records are managed by multiple healthcare providers. Thus, mutual agreement on access control by the patient and the healthcare providers is necessary to maintain privacy and regulatory compliance. One real life application that facilitates healthcare data sharing is eHealth Exchange.[12] The eHealth Exchange is a nationwide network in the United States that enables sharing of medical records across different healthcare organizations, such as hospitals, pharmacies, insurance companies, and governmental agencies. At a high level, the eHealth Exchange network operates as follows. Healthcare providers collect data directly from patients, as part of their medical care. Once collected, healthcare providers must obtain patient consent to share these data. Then, they can share the collected data with other entities participating to the network. Although the data exchanged among the participating entities are patients' records, patients do not participate directly in the network, as they only give their consent once, during the collection of the data. When another entity wants to access these data, it requests them from the provider that collected them and based on access control policies, defined solely by that provider, decides whether to allow access or not. Therefore, the patient consents once for her data and then the healthcare provider can share them with

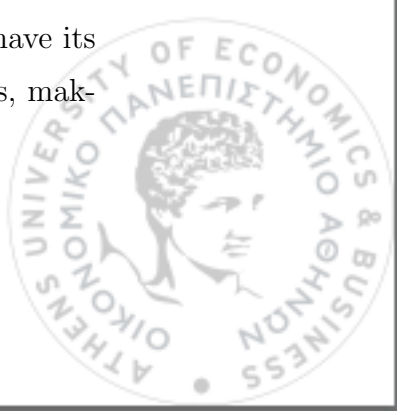
[12]https://ehealthexchange.org/



any entity it wants, without asking the patient again. With our access control solution this can be addressed. In particular, the patient can be a part of the access decision for each access request. This is critical, as there may be use cases, where the patient may want to share her data with one entity, e.g. a government agency, but not with another, e.g., an insurance company. Our solution achieves this. Aside from that, in the eHealth Exchange, access control occurs on the provider's side, on its own centralized servers. Therefore, if a patient does not fully trust the healthcare provider, she cannot be sure that her wishes are respected. Our solution, utilizing blockchain technology and smart contracts, can address that, as everything is immutably recorded on the ledger.

Another example, where data is managed by multiple mutually non-trusting entities and thus collaborative access control is critical, as it prevents any single party from exposing or altering the data, is the case of supply chains. In supply chains multiple parties, such as manufacturers, distributors, and retailers, share and manage sensitive data, such as product origins and quality certifications. There are many efforts that integrate blockchain technology to enable traceability, transparency, and auditability, among others, in supply chains. One real life application that utilizes blockchain technology is the IBM Food Trust service.[13] IBM Food Trust is a blockchain-based application, designed to bring transparency, traceability, and efficiency to the food supply chain. It is built on Fabric, which means that participants, from farmers to distributors and retailers, are known and verified. Data, such as production details, shipping information, etc., are recorded on the blockchain. The access procedures for these data are managed by the Fabric blockchain and are based on the Fabric's access control mechanism, i.e., the Fabric's MSPs, meaning that anyone possessing a certificate from any participating organization can access the data. For data, shared only among a portion of organizations, the system utilizes private channels in Fabric, i.e., different ledgers. Although this solution enables transparency and auditability with regard to access control, it is somewhat coarse-grained, as it does not allow each entity to have its own access control policies. Additionally, it does not involve external ASes, mak-

[13]https://www.ibm.com/blockchain/resources/food-trust/food-logistics/

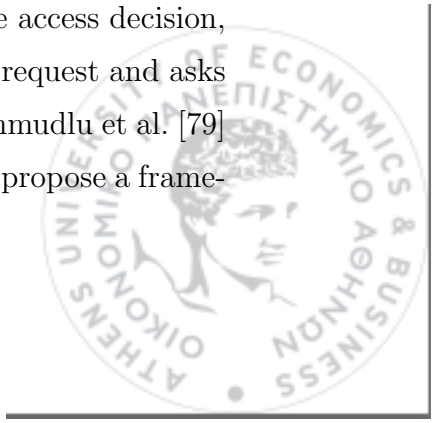


ing the system extremely vulnerable if just a single CA is compromised. Finally, it is complex, as it creates different channels and ledgers. Our solution addresses these issues, by allowing entities to define their own "local" access control policies and by incorporating external ASes.

4.3.5 Related work

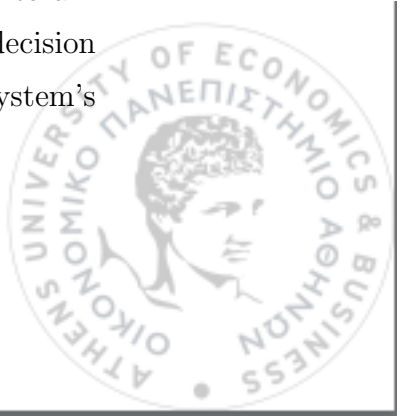
In recent years, the number of collaborative systems has been increasing. Balancing collaboration and security is a difficult problem. The first step towards securing such system is through advanced and novel access control. In particular, many research works [72, 73, 74] highlight the need for new access control models for collaborative systems, presenting the requirements and the inefficiencies of conventional access control when applied on collaborative environments. Due to these inefficiencies of traditional access control, many access control solutions tailored to collaborative multi-party systems have been proposed.

Shen et al. [75] were pioneers in introducing access control for collaborative environments. Their work, one of the earliest in this domain, propose the extension of the classic access matrix model, proposed by Lampson [76], to include collaboration rights, negative rights, automation, and inheritance-based specification. Damen et al [77] propose a collaborative access control system based on RBAC and a policy language based on a hybrid logic. For every role, the entities that need to collaborate define authorization requirements. When an access request is made, the authorization requirements are evaluated individually and the results are combined. For the final decision, the authors consider two different strategies, permit-takes-precedence and deny-takes-precedence. Their evaluation shows that the introduced overhead is from $7ms$ to $263ms$. Carminati et al. [78] enhance topology-based access control to create a collaborative access control for Online Social Networks (OSNs). They propose a policy language that allows defining collaborative access policies. Then, regarding the enforcement of the access decision, authors introduce a centralized trusted entity, which receives the request and asks the involved entities if they allow access or not. In particular, Mahmudlu et al. [79] propose a data governance model for collaborative systems. They propose a frame-



work that can express the relationships between stakeholders and data objects. To implement that, they use the access control policy language XACML [40]. Additionally, the work of Sheikhalishahi et al. [80] aims at securing and protecting the collaborative policies to avoid leak of sensitive information, e.g., the relationships of co-owners with other parties. To achieve that, they use homomorphic encryption and Secure Function Evaluation (SFE). The same problem is addressed by George et al. [81] for the use case of third-party Unmanned Aerial Vehicle (UAV) services. In particular, they present a privacy preserving multi-party access control for constrained devices. They build on top of the SFE paradigm to enable multi-party policy composition and evaluation. The UAVs can evaluate complex multi-party policies without relying on a persistent Internet connection. The main focus of this work is on how to protect the access policies. Gouglidis et al. [82] propose an enhanced RBAC, called domRBAC, for collaborative applications. Their solution differentiates the access policies that need to be enforced in each domain and supports collaboration during role assignment. Roles are enriched with the notion of domains and are expressed in pairs of domains and roles. As a PDP, they introduce an Access Control Decision Function, which parse the graph that contains all the roles.

All these works present access control solutions for collaborative systems. However, they introduce complex policy languages, and in many use cases add significant computational overhead (graph-based solutions, intensive cryptographic techniques, etc.). This can be impractical for some use cases, such as IoT, where devices are resource-constrained. Furthermore, all these solutions rely on centralized services, which may limit the resilience, scalability, and the collaboration ability. In our work, we leverage DLTs to create secure, usable, transparent, and decentralized consensus-based access control tailored to collaborative systems. Our system enables collaboration among multiple entities, without introducing complex policy languages and without revealing sensitive information of one entity to another. Notably, our system is decentralized and distributes the access decision process by using a smart contract and Fabric blockchain consensus as the system's PDP.



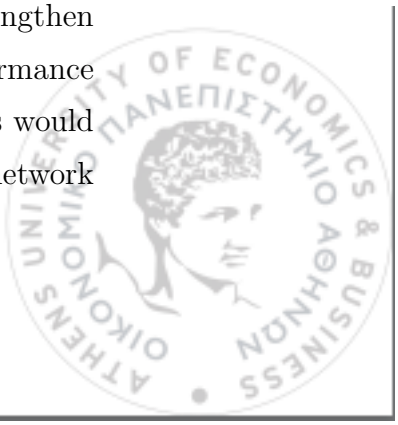
Blockchain technology has often been considered for access control system and solutions in the last few years [46, 83] and especially for access control in the IoT domain [84, 85]. There are many research efforts that utilize DLTs for access control, mainly Ethereum and Fabric that support the execution of smart contracts. In particular, some efforts [86, 52] utilized DLT to create and manage blockchain-based ACTs. Other works implement various access control entities as smart contracts [50, 51, 87], e.g., to store access control policies within a smart contract, i.e., a PIP, or to implement the PDP as smart contract. Furthermore, there are works that modify a blockchain to serve their need, e.g., Ouaddah et al. [47] introduce new types of transactions in Bitcoin that are used to get, grant, and delegate access, while others create their own blockchain to use it for access control [43]. All these efforts highlight the advantages of using blockchain and smart contracts to provide secure access control. However, they primarily focus on using blockchain technology for storing the access control policies and decisions, with smart contracts being used to process access control requests, without allowing collaboration among multiple entities. However, there is one work, by Siris et al. [88] that in one of the proposed models, authors try to utilize the blockchain technology to allow many authorizations servers to collaborate for access. However, in this work, all the ASes have to be known and their endpoints should be encoded in the smart contract. Also, this work does not support the collaborative definition of the access control policy.

In our work, we take this a step further. We do not only utilize smart contracts, but in our second design, the endorsement-based approach, we leverage a part of the actual consensus mechanism of the Fabric blockchain for deciding on and enforcing access. To the best of our knowledge, there is not any other prior work that uses the consensus mechanism of blockchain for access control processes. Furthermore, these works do not support collaboration between many entities for the access control.

4.3.6 Conclusions and future work

We proposed a decentralized, secure, flexible, consensus-based access control solution tailored to collaborative systems. Our solution utilizes the blockchain technology, and in particular the Hyperledger Fabric permissioned blockchain. We presented two different approaches, one that utilizes smart contracts to allow the collaboration of many entities regarding access control and to decide the final access decision. The smart contract-based approach can be implemented in any blockchain, public or private that allows the execution of smart contracts. The second approach takes advantage of the consensus mechanism of Hyperledger Fabric, i.e., access is decided based on a part of the consensus mechanism of the blockchain itself, the *endorsement policy*. For both approaches, we proposed mechanisms for integrating external ASes in an efficient, private, and secure way. In addition, a strong property of our solution is that it can incorporate many access control models, as the access control model utilized is transparent to our system, which is used as an underlying framework for collaborative environments. Our evaluation demonstrates that our solutions is feasible, introducing a bearable blockchain-induced overhead of 0.02 seconds. Furthermore, under normal conditions, our two proposed approaches exhibit latencies of 0.11 seconds and 0.02 seconds, respectively. This shows that although the smart contract-based approach provides increased transparency, it also results in higher communication overhead, presenting worse performance compared to the endorsement-based approach. Finally, our security analysis demonstrates that our solutions are secure and meet the defined requirements – confidentiality, integrity, availability, transparency, auditability, non-repudiation, and privacy – even in the face of various attacks.

The presented access control solution can be extended in many ways. Depending on the use case, revocation and/or delegation mechanisms may be required. Thus, experimenting with effective and efficient such mechanisms and integrating them in the presented access control solution would extend and strengthen the system as a whole. Another next step, would be to evaluate the performance of the proposed access control solution in a larger-scale environment. This would involve testing the system under various configurations, including different network



https://doi.org/10.26219/heal.aueb.9346

sizes, peer organizations, and more complex setups. Additionally, further investigation into the impact of different endorsement policies and their interactions with scalability and performance would provide valuable insights.

https://doi.org/10.26219/heal.aueb.9346

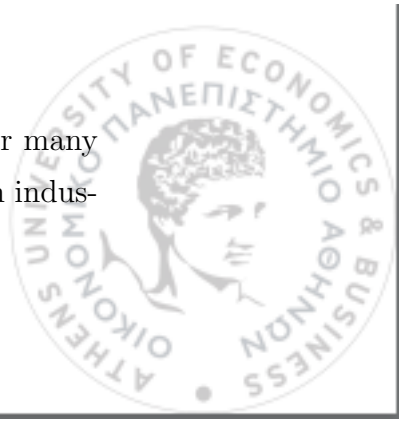
Chapter 5

Applications of blockchains in IoT – Security and interoperability

In this chapter, we explore the innovative application of blockchain technology within the IoT, focusing on two distinct use cases: IoT mobile gaming and digital twin technologies. By integrating blockchain into these areas, we aim to enhance security and interoperability among diverse IoT systems and devices, which is crucial for seamless and secure interactions. In the realm of mobile gaming, blockchains and smart contracts offer a decentralized platform enabling ecosystem expansion and diversification, customer attraction and retention, exploitation of context sensitive and personalized advertisements, and improved monetization of in-game assets. On the other hand, they can be used to realize secure, available, and reliable digital twins of IoT devices that offer interoperability, decentralization, and auditability. Throughout this chapter, we demonstrate how blockchain technology can address some of the core challenges of IoT, including security and interoperability, thus unlocking new potentials for IoT deployment across diverse sectors.

5.1 The case of IoT mobile gaming

Blockchain constitutes a significant baseline platform technology for many applications in various domains, as it offers intriguing properties. One such indus-



https://doi.org/10.26219/heal.aueb.9346

try that can potentially significantly benefit from the use of blockchains is the mobile gaming industry.[1] As the blockchain technology matures, more and more companies from the gaming industry experiment with integrating blockchains into their games. The first blockchain-based game that became a success was CryptoKitties,[2] launched in 2017. Since then, many other and of different types blockchain-based games have been developed, including gambling games,[3] online casinos,[4] and trading games,[5] among others. Each one of these types of games exploits different attributes and advantages of blockchains.

Blockchains provide decentralization, immutability and improve robustness, availability, transparency and finally trust among participants, enabling mutually non-trusting parties to interact and engage with each other, without the need for a trusted third party. When it comes to the mobile gaming industry, blockchains can help in the creation of large gaming ecosystems, involving many and diverse stakeholders that may have conflicting interests, e.g., advertising companies and independent programmers, among others. Nevertheless, these properties come with a cost. The CryptoKitties game revealed some of the inefficiencies of blockchain technology. CryptoKitties attracted a lot of attention, more than 40000 active users per day, leading to increased traffic on the Ethereum network, resulting in a sixfold increase in pending transactions on the Ethereum blockchain.[6] Furthermore, developers of such games have to face a variety of other issues regarding the performance of their games, such as increased delays, scalability issues, and last but not least, high transaction costs. Therefore, designing a blockchain-based game is not a trivial task and sometimes it can be very challenging, leading to non-scalable and non user-friendly systems.

In this work [89], we explore different system architectures, combining various types of ledgers, in order to determine a suitable architecture for blockchain-based games, and investigate the relative trade-offs in combining public and private

[1] https://www.fortunebusinessinsights.com/blockchain-gaming-market-108683
[2] https://www.cryptokitties.co/
[3] https://satoshidice.com/
[4] https://7bitcasino.com
[5] https://godsunchained.com/
[6] https://www.bbc.com/news/technology-42237162

ledger technologies. Key advances in the blockchain technology exploited in this work are: private and public instances of the Ethereum blockchain, the Fabric blockchain, which allows efficient and high performance operations in consortia, and finally interledger mechanisms [56] that allow hybrid ledger ecosystems, benefiting from the best of both worlds.

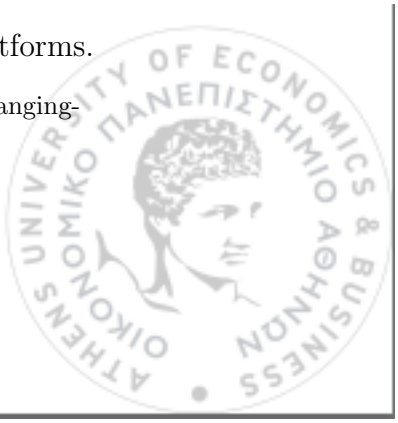
In addition, we consider the inclusion of IoT devices and technologies in mobile gaming, which can help in the creation of novel and fun mobile games, while enlarging the set of involved parties and creating new business opportunities at the same time. Gaming companies, and in particular mobile gaming companies, try to take advantage of the many benefits and opportunities that IoT provides in mobile gaming.[7] They try to combine the real with the virtual worlds, in order to develop fun augmented or mixed reality games that make players feel that they are personally involved in the game environment, part of which is in the real world. Furthermore, by creating games that combine the physical with the digital world, new business opportunities arise. Cafes, malls, and similar establishments and companies, can participate in such applications, creating or enlarging the gaming ecosystems, by deploying IoT devices in their premises in order to contribute and expand the location-based, context-aware, augmented or mixed reality gaming experience, as we will discuss below.

To sum up, we investigate and evaluate the combination of IoT and blockchains in mobile gaming. The contributions of this work are:

- We investigate exploiting the IoT and DLTs in designing an expanding mobile gaming platform that can support an open mobile gaming ecosystem with improved in-game features, such as in-game assets, transparent transactions among the various entities, e.g., players, game developers, and advertisers, cross-game and game company assets and value transfers, and player interactions with the real world.

- We define KPIs for IoT-based and DLT-supported mobile gaming platforms.

[7] https://www.headstuff.org/entertainment/gaming/how-the-internet-of-things-is-changing-the-gaming-industry/

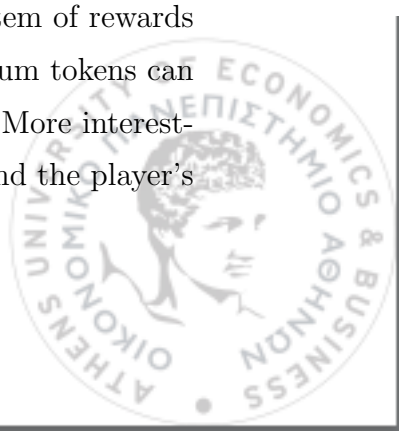


- We illustrate the advantages of permissioned and permissionless blockchains in the mobile gaming context and demonstrate the benefits of interledger technologies in combining both types of blockchains, creating hybrid environments.

- We evaluate a specific IoT and hybrid DLT mobile gaming platform designed through a combination of emulation of gaming functions and actual implementations of blockchain functionality with different combinations of an Ethereum public blockchain, a private instance of Ethereum, and Fabric blockchain.

5.1.1 A scavenger hunt location-based mobile game ecosystem emulation

The mobile gaming industry has been considering how to expand the gaming ecosystem. Various ideas have been contemplated and directions proposed. First, improving player trust in the exclusivity, rareness, or even uniqueness of in-game assets provided by gaming companies and traded by players. This is a privileged area for blockchains. Then, if that is secured, the (long-term) value of assets increases and it was felt that players that own such assets should be able to transfer them between games, even across ecosystems of different gaming companies. Again blockchains can play a role here; if the ecosystems use different blockchains, interledger technologies provides solutions. Another direction exploited to engage players and programmers and expand games and keep them fresh and adapt them to various communities and interests is the incorporation of the capability of third parties to design in-game "challenges." These are specific tasks designed and programmed possibly by independent, to the main gaming company, parties that are attached to the game and expand it. These challenges can bring their own rewards, in particular if there is a general, accessible system of rewards and assets endorsed by the game, such as a blockchain–e.g., Ethereum tokens can play both roles. Finally, advertising is also present in this domain. More interesting is perhaps context-aware advertisements, related to the game and the player's

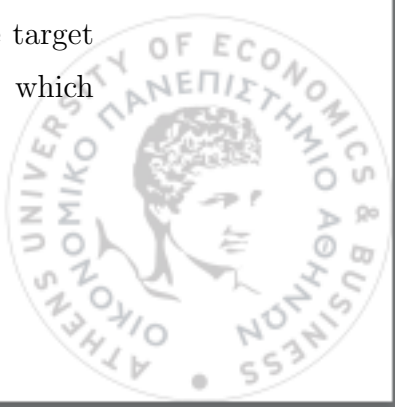


performance, interests, and choices. Enabling advertising companies to conduct campaigns outside the complete control, and perhaps without a-priory knowledge, of the gaming company is an intriguing possibility, potentially supported by open blockchains.

A different direction has been very successfully brought to the forefront by Pokémon Go,[8] one of the most used and profitable mobile apps in 2016. In addition to Augmented Reality (AR), it popularized location-based gaming technology, creating opportunities for brick and mortar stores and other businesses to exploit the digital revolution and mobile gaming in particular to improve their bottom line due to increased foot traffic. On the other hand, Pokémon Go also exhibited many other features of mobile gaming we are considering and trying to address through technology in this work: cheating and hacks by players, the power and presumed arbitrariness of the gaming company, the performance problems of centralization, in particular in the case of targeted DoS attacks, the low resolution of the location technology and others. But the success and publicity created by Pokémon Go has intensified interest in location-based and context-aware gaming more generally and in particular in combination with advanced IoT technology availability and affordability.

In order to evaluate how blockchains and IoT technologies perform in demanding mobile gaming applications and their potential impact on expanding gaming ecosystems, we emulated a generalized version of the scavenger hunt location-based mobile game introduced and implemented by Rovio in the EU H2020 project Secure Open Federation for Internet Everywhere (SOFIE).[9] The game is designed around locations with players incentivized to move from location to location with puzzles presented only when present in the right location. Players are rewarded for solving the puzzles. The game utilizes IoT beacons or specific Wi-Fi access points or localization, includes "challenges," and exploits blockchains. A comprehensive description of the mobile game and its architecture is presented in [90]. In a nutshell, the players have to solve some riddles using clues to reveal the target locations. By solving these riddles, the players are rewarded with points, which

[8]https://pokemongolive.com/
[9]https://www.sofie-iot.eu/en

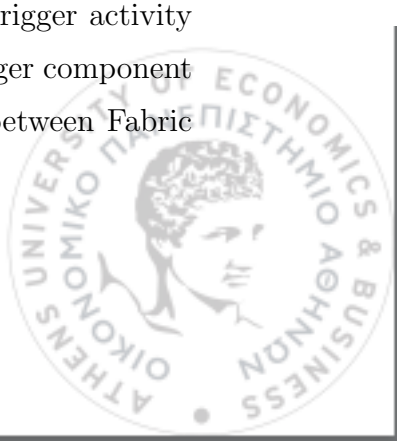


they can exchange with rewards and assets. From a high level perspective, the flow of the game is as follows. Initially, the player sees the available nearby riddles (challenges), based on her GPS location. Then, she selects one of them, and receives some clues that will lead her to the first location that she has to visit. When she arrives at the location of a deployed IoT device, a riddle alongside with some clues are shown on her mobile device. Solving the riddle or performing the appropriate task, e.g., take a picture of a statue located in square X, where X is four blocks away from square Y, will reveal the location of the next IoT device, in order for the player to go there, collect her points, and download the next riddle. This procedure continues until the last IoT device is reached.

In addition to the core functionality of the game described above, the specific mobile game provides some additional features. A user is able to skip any challenge, at any time of the game, by viewing an advertisement offered by an independent advertising company, by paying in in-app tokens or in fiat currency. Furthermore, a player can get a reward, which can be an asset for the game, by the advertising company, if she watches an advertisement. Moreover, the player can at any point of the game redeem her points in order to get rewards given by the gaming company. The rewards are assets, e.g., a sword or a shield, that can be used in the game or in any other game that uses the same blockchain platform.

The blockchains considered in the design of the game are Ethereum and Fabric, for different roles and reasons. A private instance of Ethereum, without the energy-hungry proof-of-work can be used as an alternative to Fabric. These blockchains are chosen for external and internal gaming functions. To interconnect these two blockchains, there is a need for an Interledger Gateway (ILG) that handles the communication between the two ledgers. An ILG is a component that enables interoperability among different DLTs, by transferring information and value between them. In the SOFIE project, an interledger component has been developed that enables activity on one ledger, called *Initiator*, to trigger activity on one or more ledgers, called *Responder(s)*.[10] This specific interledger component is used in the mobile game in order to allow the communication between Fabric

[10]https://github.com/SOFIE-project/Interledger



and public or private Ethereum networks.

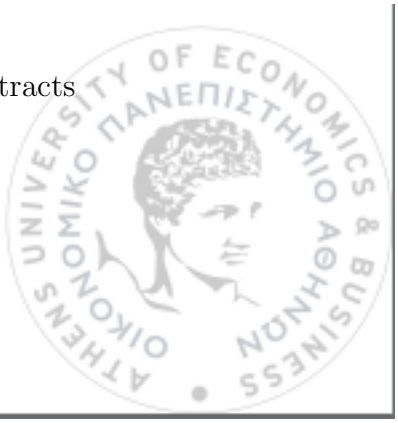
For the IoT part of the game, IoT beacons are used as proximity sensors to determine the location of a player, when she visits the corresponding Point of Interest (PoI). The beacons could be sensed by the user's smartphone using the Bluetooth Low Energy (BLE) standard, or even Wi-Fi, with different accuracies.

Modeling and emulation environment

To evaluate the performance and other aspects of the scavenger hunt location-based mobile game, we developed an emulation environment, as it allows us to easily compare different blockchain setups that utilize public (permissionless) and private (permissioned) blockchains, which would be complicated and difficult with the actual mobile game implementation. Our emulation involves and investigates public ledgers, using a public Ethereum test network, and private ledgers, using a private Ethereum network or the Fabric blockchain, allowing us to compare different configurations with the two aforementioned types of blockchains that have different features and trade-offs in terms of transaction cost, latency, transparency, and privacy. Our evaluation through the emulation environment focuses on some system performance metrics (KPIs) and does not consider business oriented metrics such as player satisfaction, gaming experience, and revenue opportunities.

The implemented emulation environment is composed of the following entities and actors (see also Figure 5.1):

- A Web application that emulates the mobile gaming client
- The players
- The game administrator
- The (public or private) Ethereum blockchain and the corresponding smart contracts
- The Hyperledger Fabric blockchain and the corresponding smart contracts
- An ILG



https://doi.org/10.26219/heal.aueb.9346

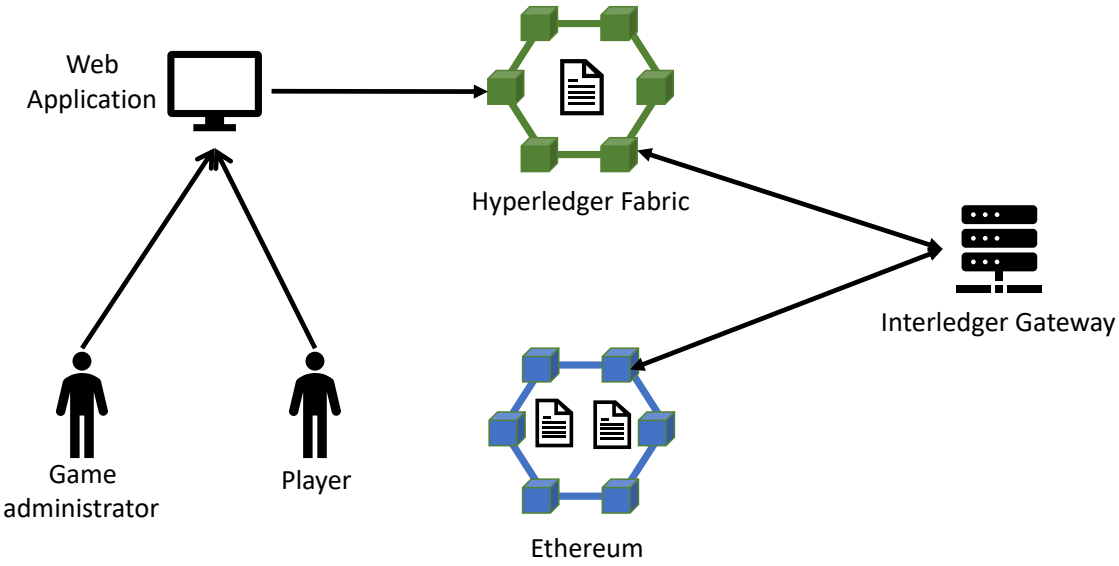
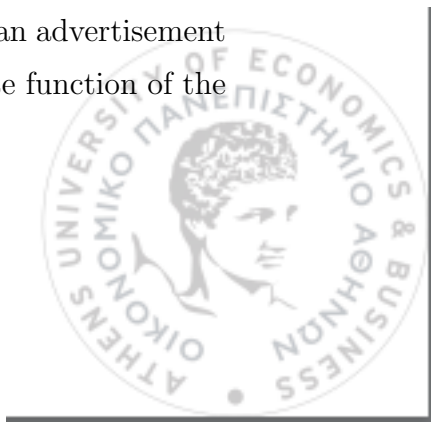


Figure 5.1: Architecture diagram for the emulation environment of a scavenger hunt location-based game, involving two blockchains that are interconnected through an Interledger Gateway.

The first component of our system is the mobile gaming client, which is emulated as a Web application implemented using React.js.[11] Both players and administrators use this Web application to interact with the game, players to play the game and administrators to modify the game. In order for actors to interact with the application, they need to own an account in the gaming system, as well as a blockchain wallet account. Each challenge/riddle is identified by a unique identifier. In order for a player to select and "play" a challenge, she has to choose the appropriate challenge identifier. The solution of the challenge's riddle is emulated in the application through a "complete" button. The player can skip a challenge by paying some predefined amount in cryptocurrency, or in in-app tokens, or by viewing an advertisement. The in-app tokens are developed using the Ethereum ERC-20 token standard. Finally, advertisements are emulated as functions of a smart contract. Thus, if a player wants to "watch" an advertisement and get the corresponding rewards, she has to call the appropriate function of the smart contract from the application.

[11] https://reactjs.org/

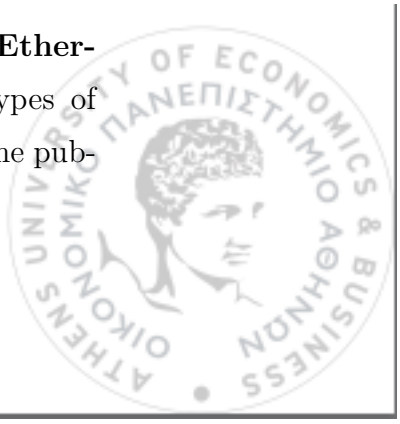


The main functionality of the scavenger hunt location-based mobile game is emulated by three smart contracts. The first smart contract, named *game smart contract* implements all the functionality related to the challenges and rewards. In particular, it records the challenges on the blockchain. It also records a mapping of players and challenges and whether a particular player has completed the tasks of a challenge or not. Furthermore, the smart contract automatically calculates the points that each player obtains, when he completes the challenge. Moreover, it implements the functions for skipping a challenge. Finally, it implements a function for redeeming the rewards. The second smart contract, called *ads smart contract* contains the functions that emulate the advertisements. It is responsible for checking whether a player "watched" the advertisement or not and provides the corresponding rewards. The third smart contract is the *token smart contract*. The token smart contract mints, burns, and manages the in-app tokens. In our emulation environment, we have implemented the ILG as one entity that "watches" all the blockchains for emitted events and acts accordingly when it receives one such event. Adding one entity that acts as the ILG is the easiest and typical solution. The interledger functions are implemented using Node.js and the web3 JavaScript library in order to interact with the smart contracts.

The architecture shown in Figure 5.1 is one of the four architecture scenarios that we have developed, and in particular is the fourth emulation scenario (see below). Figure 5.2 presents the actors' interactions with the emulation environment. The diagram shows all the involved actors of the mobile gaming ecosystem and the functions of the emulated mobile game, they can perform or interact with. The four emulation scenarios that involve different DLTs setup are presented below.

Scenario 1 – One public blockchain - Ethereum. The first scenario considers a single public Ethereum blockchain. All three smart contracts, which implement the gaming functionality, are deployed on the Rinkeby Ethereum test network.

Scenario 2 – Two types of blockchains - public and private Ethereum. The second scenario investigates the gains from utilizing two types of blockchains, a public blockchain and a private/permissioned blockchain. The pub-



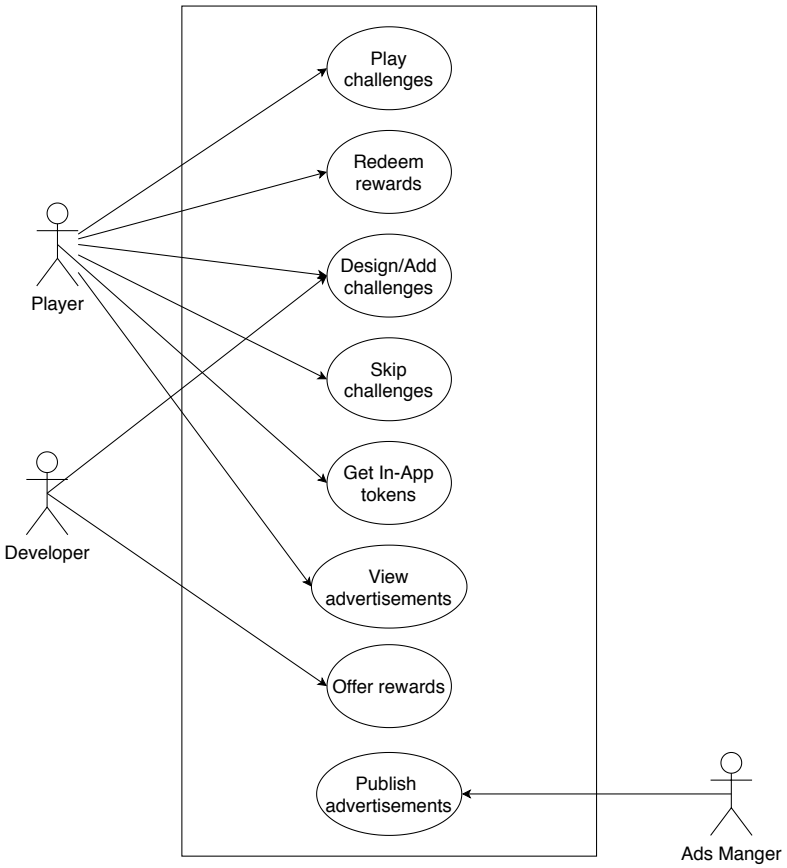
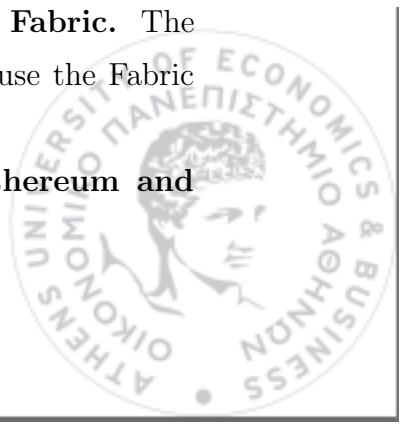


Figure 5.2: Actors' interaction with the scavenger hunt location-based mobile gaming emulated system.

lic blockchain is an instance of Ethereum on Rinkeby Ethereum test network, while the permissioned blockchain is a private instance of Ethereum. The game smart contract is deployed on the private ledger, while the other two smart contracts are deployed on the public ledger. In order for the two ledgers to communicate with each other, there is an ILG, which is responsible for the interconnection of them. ILG "listens" for events on both instances of Ethereum and handles the corresponding functions.

Scenario 3 – One private blockchain - Hyperledger Fabric. The third scenario considers a single private ledger. In particular, we use the Fabric blockchain. All smart contracts are deployed on it.

Scenario 4 – Two types of blockchains - public Ethereum and

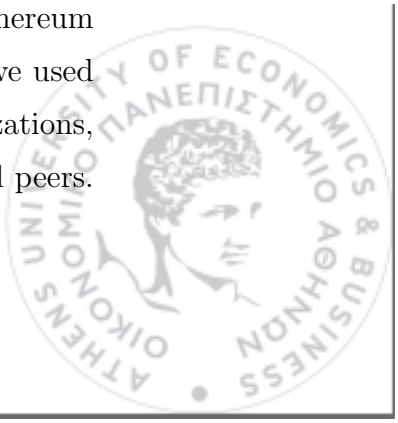


Hyperledger Fabric. Last but not least, the fourth emulation scenario utilizes the two types of blockchains, as in scenario 2, but differs from that scenario in that the private blockchain used is the Fabric blockchain, instead of a private instance of Ethereum.

5.1.2 Performance evaluation

To evaluate the performance of the presented mobile game using the emulation environment, we first define the performance metrics that play crucial role in the performance of the system. The first performance metric we consider is the response time, namely the time required to execute various requests. In this case, where the gaming system utilizes blockchain and IoT devices, the response time is affected by the time that the system performs read and write transactions, and the time that an IoT device needs to detect the players arriving in a particular location. Furthermore, since we utilize public Ethereum in the mobile game, we should consider the execution cost, measured in gas, as a performance metric. Finally, the last two performance metrics we consider are the throughput and the scalability, since they characterize the volume of transactions, as well as the number of players that the system can support in a given time window. The aforementioned metrics constitute the KPIs and are shown in Table 5.1. We do not consider business oriented metrics and that is why in the table there are not popular KPIs for mobile gaming, such as Daily Active Users (DAU), Average Revenue Per User (ARPU), and Monthly Active Users (MAU) among others, which are important but addressable only through a real implementation.

We now describe the performed experiments needed to measure the defined KPIs and the corresponding results for all the emulation scenarios described in the previous Section. The smart contracts running on the Ethereum blockchain were written in Solidity, while the smart contracts running on Fabric were written in Node.js. Ethereum smart contracts were tested in the Rinkeby Ethereum test network. On the other hand, for the scenarios that leverage Fabric, we used Fabric v1.4. The blockchain network in Fabric was consisted of two organizations, having two endorsing peers each. The smart contracts were deployed in all peers.

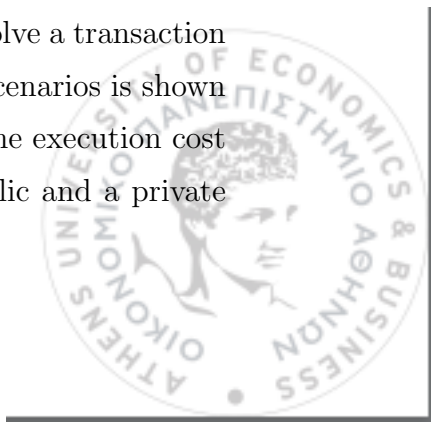


KPI	Name	Description
KPI_1	Public ledger execution cost	Cost for executing operation on a public ledger
KPI_2	Response time for write requests	Time for the system to respond to write transactions
KPI_3	Response time for read requests	Time for the system to respond to read transactions
KPI_4	BLE beacon detection time	Time that player has to wait between arriving at location and receiving the task
KPI_5	Throughput	Maximum number of transactions per time unit
KPI_6	Cost scalability	Increase of cost as number of challenges increases
KPI_7	Time scalability	Increase of response time as number of challenges increases

Table 5.1: System Key Performance Indicators for the scavenger hunt location-based mobile game.

The results presented below are the average value of 10 executions. Furthermore, we calculated the 95% confidence interval for each of the experiments, using the Student's t-distribution (because the sample size is small, < 30).

KPI_1 refers to public ledger execution cost and is measured in gas units. The third scenario, which utilizes only the Fabric blockchain does not entail an execution cost, since it does not perform any actions on a public ledger. For the same reason, some of the actions in the second and fourth scenario incur zero cost. Furthermore, some actions in different scenarios have the same implementation, hence they also have the same transaction cost. All the actions that involve a transaction on a public ledger and the corresponding execution cost, for all scenarios is shown in Table 5.2. We observe that for all the actions except of one, the execution cost is smaller in the second and fourth scenarios that involve a public and a private

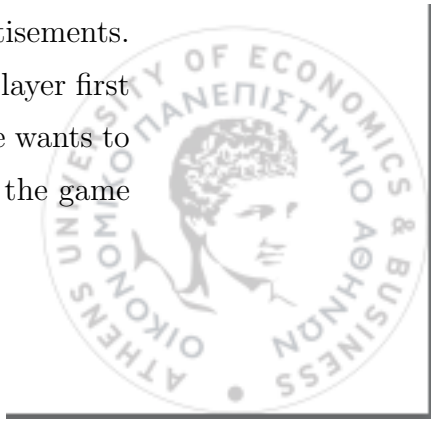


Actions	**Scenario 1**	**Scenario 2**	**Scenario3**	**Scenario 4**
Add challenge	47050	N/A	N/A	N/A
Begin challenge	52423	N/A	N/A	N/A
Complete challenge	53529	N/A	N/A	N/A
Skip challenge by paying	61867	N/A	N/A	N/A
Skip challenge by in-app tokens	63877	33438	N/A	33438
Skip challenge by viewing ads	53926	21462	N/A	21462
Get in-app tokens by paying	44199	35274	N/A	35274
Get in-app tokens by viewing ads	37981	56736	N/A	56736
Redeem rewards	36618	35274	N/A	35274

Table 5.2: Ethereum Virtual Machine execution cost (gas) for the actions of the scavenger hunt location-based mobile game.

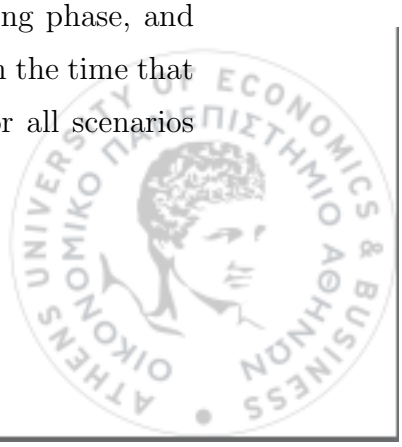
ledger. This happens, because these actions involve the interaction of the game smart contract with the ads or the token smart contract. In these scenarios, the game smart contract is deployed on the private ledger, so the actions performed by that smart contract do not incur cost. For this reason, the total amount of cost required for these specific actions is the gas consumed only by the public ledger. We should note here that the execution of a transaction in the private Ethereum consumes private EVM resources. So, we assume that the cost is zero, and that is why we have set the results for these actions as Not Applicable (N/A).

The function with the highest execution cost on the second and fourth scenarios is the function invoked for getting in-app tokens by viewing advertisements. This function involves the invocation of all three smart contracts. The player first calls the function of the game smart contract to alert the system that she wants to "watch" an advertisement in order to acquire some in-app tokens, then the game



smart contract invokes the ads smart contract, and finally when the ads smart contract finishes the execution, the game smart contract invokes the token smart contract to transfer the in-app tokens to the player's account. In the first scenario, all these invocations can be done only by one transaction for invoking the game smart contract. Then, the game smart contract will invoke the other two smart contracts internally. In the other two scenarios, we cannot perform this specific action by sending just one transaction, since the smart contracts are deployed on different blockchains, thus we need two more transactions, initiated by the ILG. One transaction to invoke the game smart contract (zero cost), which will trigger an event that will be "caught" by the ILG. Then, the ILG will send a transaction to the ads smart contract. When the execution will finish, it will send a transaction back to the game smart contract to inform it that the user "watched" the advertisement. After that, it can proceed with the payment. The same process is followed with the token smart contract. Thus, for this particular action, the second and the fourth scenario adds an overhead of two more transactions on the public ledger. The above results confirm that using a single public ledger is very costly. Furthermore, the results quantify the gains, in terms of cost that can be obtained by combining a private and a public ledger.

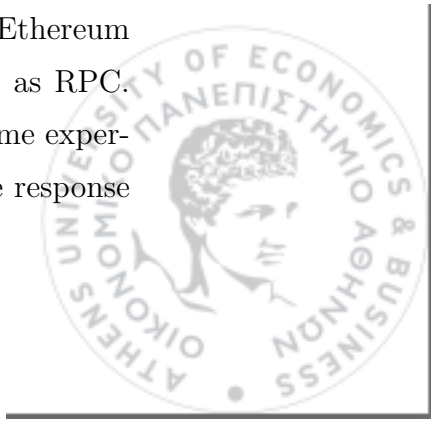
The second KPI refers to response time for write requests. As we have already mentioned, in Ethereum, the transaction delay depends on the block mining time and it is ≈ 15 seconds. Therefore, for scenarios that utilize the Ethereum blockchain (public or private) the response time for write transactions is ≈ 15 seconds. This time is also affected by other external factors, such as the load of the network, the size of the transaction, and how many transactions can fit in a block. We performed some experiments in the Rinkeby Ethereum test network, to validate that the response time is indeed around 15 seconds. On the other hand, in Fabric, in order for a transaction to be added on the ledger, it must get through by three phases. These phases are the execution phase, the ordering phase, and the validation phase. So, in Fabric, the transaction delay depends on the time that each phase requires. The results, with the confidence intervals, for all scenarios are presented in the Table 5.3.



Actions	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
Add challenge	14.55 (± 2.48)	14.55 (± 2.48)	2.209 (± 0.03)	2.209 (± 0.03)
Begin challenge	15.38 (± 3.14)	15.38 (± 3.14)	2.195 (± 0.01)	2.195 (± 0.01)
Complete challenge	14.14 (± 3.57)	14.14 (± 3.57)	2.187 (± 0.01)	2.187 (± 0.01)
Skip challenge by tokens	12.96 (± 2.90)	12.96 (± 2.90)	2.176 (± 0.01)	12.96 (± 2.90)
Skip challenge by ads	15.14 (± 3.39)	15.14 (± 3.39)	2.182 (± 0.02)	15.14 (± 3.39)
Get tokens	11.12 (± 2.05)	11.12 (± 2.05)	2.187 (± 0.01)	11.12 (± 2.05)

Table 5.3: Mean response time units for write requests (confidence intervals) for the actions of the scavenger hunt location-based mobile game.

The next KPI refers to the time needed for the system to respond to non-altering transactions, namely read requests. Read requests in Ethereum do not broadcast or publish anything on the blockchain, and the response is returned instantaneously, since the requests are local. Furthermore, read requests in Fabric follow the same flow as the one described above for the write requests, with the difference that for the read requests the flow ends when the proposal response is returned to the client (i.e., there is not an ordering and validation phase). The results are shown in Table 5.4. We observe that the response time in scenarios that utilize Ethereum is approximately 1 second, which is not negligible as expected. That is happening because in our emulation environment, we do not own and run an Ethereum (full or light) node. So, in order to interact with the Ethereum blockchain, we communicate with another node of the network, acting as RPC. Thus, an additional (network) overhead is introduced. We performed some experiments with local Ethereum, in order to find out the exact value for the response



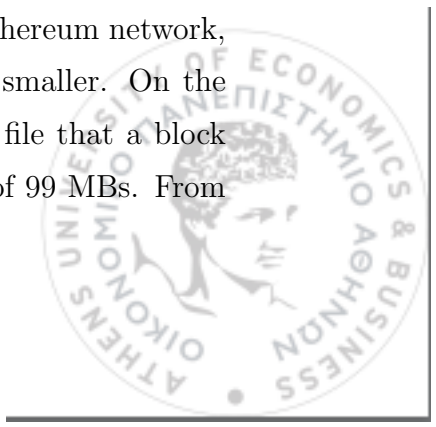
Actions	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
Query points	1.106 (± 0.05)	1.106 (± 0.05)	0.024 (± 0.002)	0.024 (± 0.002)
Query challenges	1.085 (± 0.04)	1.085 (± 0.04)	0.026 (± 0.006)	0.026 (± 0.006)
Query tokens	1.124 (± 0.06)	1.124 (± 0.06)	0.021 (± 0.001)	1.124 (± 0.06)

Table 5.4: Mean response time units for read requests (confidence intervals) for the actions of the scavenger hunt location-based mobile game.

time in non-altering transactions, which is ≈ 0.045 seconds.

The next KPI is the BLE beacon detection time. Functionality related to beacon detection time is not implemented in the smart contracts, hence this time is independent of the ledger transaction time. Furthermore, since this action is independent of the blockchains, our emulation environment does not consider IoT related actions, thus we do not have produced results about this specific KPI. However, in [90], there is an exhaustive evaluation about the beacon performance for this specific mobile game. Authors calculated that the average delay is *5.8* seconds.

Next, we have the KPI related to throughput, which is defined as the number of transactions per time unit that the system can support. To measure this KPI, we conducted a number of experiments sending write requests to the system, since read requests are local, and they do not record anything on the ledger. Initially, we measured the throughput of private Ethereum. In these experiments, we observed that a block can store up to 74 transactions, related to our game. Also, we know that a block is mined in 15 seconds, so the throughput of the system is 4 TPS. We should note that the throughput is affected by the number of transactions that can fit in a block. In particular, we measure the transactions that fit in a block by calculating the amount of gas that is packed in the block. So, the throughput might be increased for some other actions that costs less gas. In public Ethereum, a block will contain transactions from the whole Ethereum network, and not only from our game, thus the throughput will be even smaller. On the other hand, in Fabric, we have observed from the configuration file that a block can fit up to 20 transactions, with a maximum transaction size of 99 MBs. From



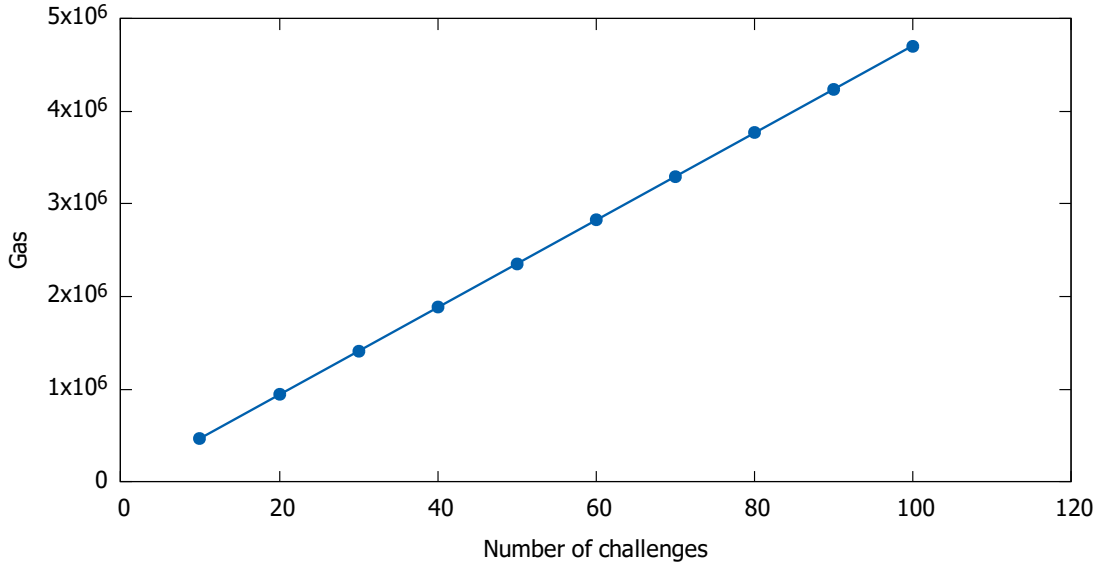
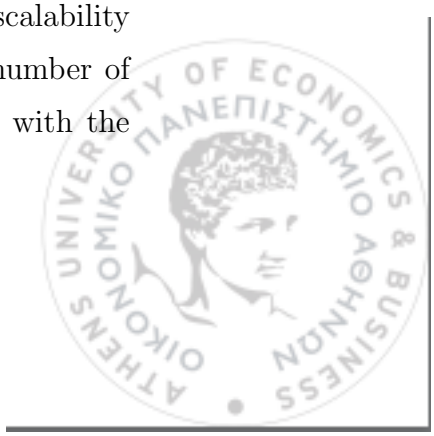


Figure 5.3: Ethereum gas consumption as a function of the number of game challenges for the scavenger hunt location-based mobile game.

the experiments we conducted, we measured that a block is added on the ledger in about 100 milliseconds and that all the transactions related to our use case had transaction size smaller than 99 MBs. Therefore, the throughput of the system is 200 TPS. The above results are not affected by the ILG component, since we measure the throughput of each blockchain separately.

The last two KPIs are related to scalability, and in particular they show how the execution cost (KPI_6) and the response time (KPI_7) are affected by the number of challenges or users. As the throughput, scalability is not affected by the ILG or any other component of the system. The cost scalability concerns only scenarios that use the public Ethereum blockchain, which introduces a transaction cost. We defined the cost scalability as the ratio of the cost over the number of challenges. The cost scalability is shown in Figure 5.3. We observe that the cost scalability is linear to the number of challenges. Similarly, the cost scalability remains linear, not only to the number of challenges, but also to the number of in-app tokens, points, and any other metric that involves transactions with the Ethereum blockchain.



https://doi.org/10.26219/heal.aueb.9346

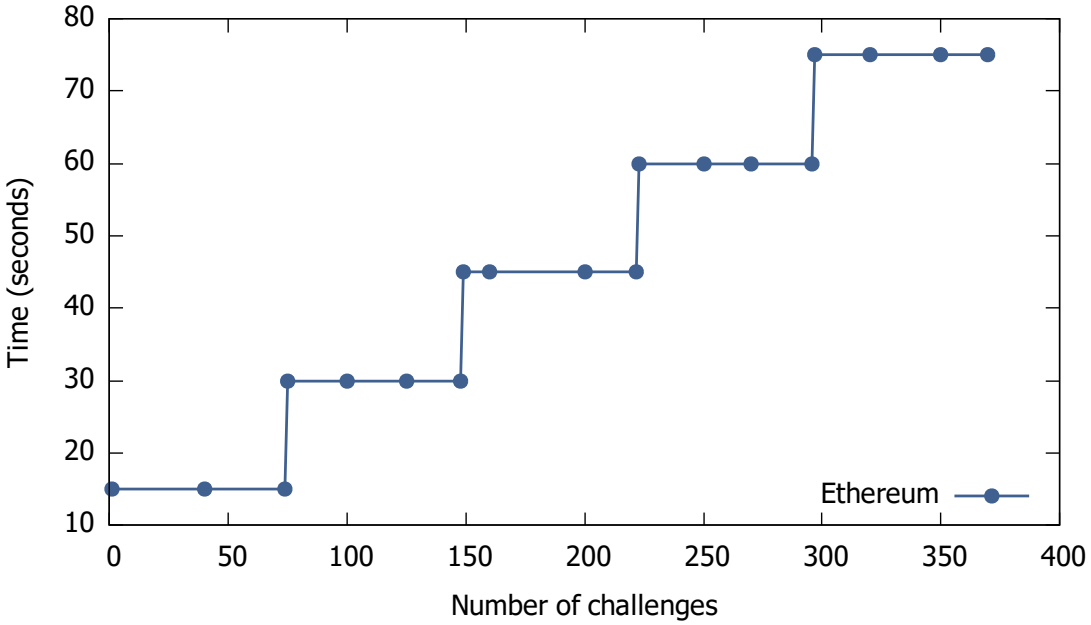
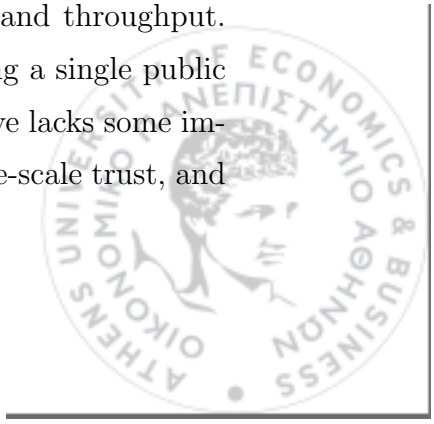


Figure 5.4: Time scalability for Ethereum-based scenarios in the scavenger hunt location-based mobile game.

Finally, the last KPI refers to time scalability, which is shown in Figure 5.4 for the public Ethereum, and in Figure 5.5 for Fabric, respectively. We define time scalability as the ratio of time over the number of challenges. From the figures, we observe that the time scalability for all types of blockchain is a stepwise function, which macroscopically becomes linear. Again, the time scalability is similar for points, in-app tokens, etc.

All the results for all the KPIs and scenarios are summarized in Table 5.5. The results in the table are the average of all the corresponding actions of all experiments. Furthermore, for the fourth emulation scenario, which utilizes both public Ethereum and Fabric, the results presented in the table are for both blockchains. It is clear that the use of public ledger (Scenario 1) is expensive and demonstrates poor performance results, especially in terms of response times and throughput. Using a single private ledger (Scenario 3) is way better than using a single public ledger, in terms of cost and performance. However, this alternative lacks some important properties of the public ledger, such as transparency, wide-scale trust, and



https://doi.org/10.26219/heal.aueb.9346

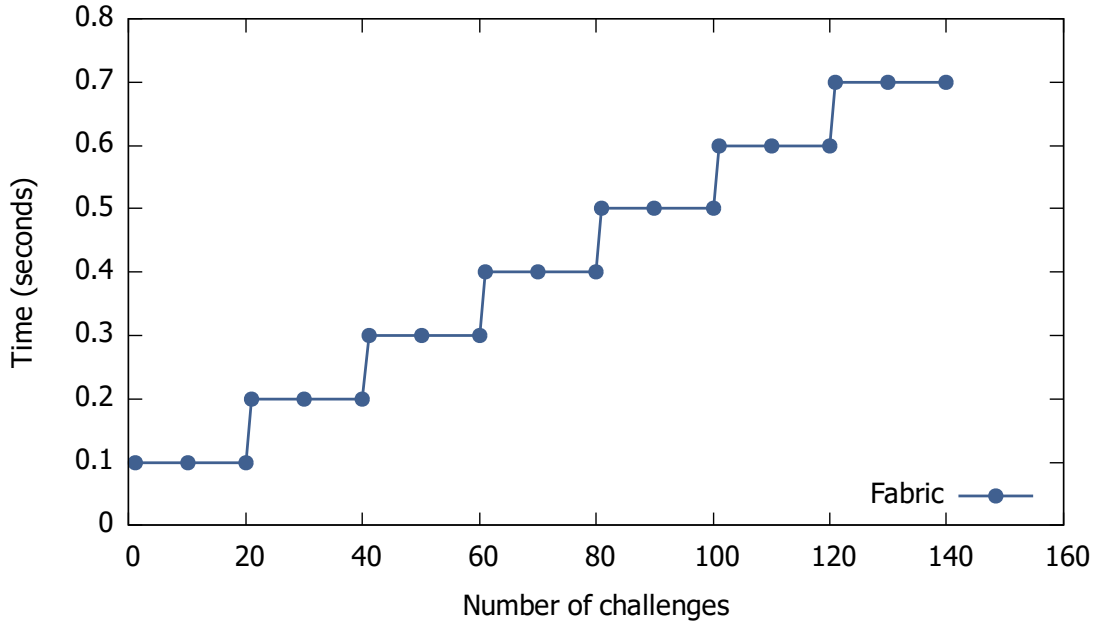
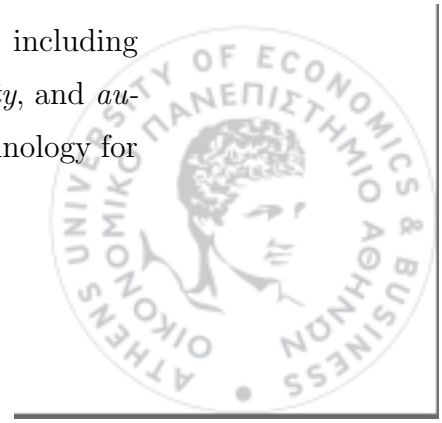


Figure 5.5: Time scalability for Hyperledger Fabric-based scenarios in the scavenger hunt location-based mobile game.

openness. Thus, the best solution is to combine these two types of blockchains, the private ledger in order to implement all the gaming functionality for which time is a crucial factor, while the public blockchain in order to implement other actions, such as advertisements and assets specific actions that require higher levels of trust, transparency, and openness in order for the entities to be able to join the ecosystem easily. So, since Fabric demonstrates better performance results than a private instance of Ethereum, we end up that the private ledger should be the Fabric blockchain, thus, the fourth emulation scenario is more suitable than the second that utilizes public and private Ethereum.

5.1.3 Discussion

We have showed that blockchains have many useful properties, including *decentralization, replication, transparency, non-repudiation, immutability*, and *auditability*. All these properties and features make DLTs a promising technology for



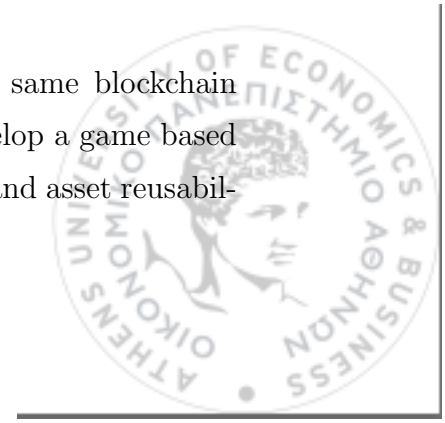
KPIs	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
KPI_1	50164	36437	0	0 & 36437
KPI_2	13.89 sec.	13.89 sec.	2.189 sec.	2.197 & 13.89 sec.
KPI_3	1.105 sec.	1.105 sec.	0.024 sec.	0.025 & 1.124 sec.
KPI_4	N/A	N/A	N/A	N/A
KPI_5	4 write TPS	4 write TPS	200 write TPS	Scenario 1 & 2
KPI_6	Linear	Linear	N/A	Linear
KPI_7	Step-wise	Step-wise	Step-wise	Step-wise

Table 5.5: System performance evaluation results for all the emulation scenarios for the scavenger hunt location-based mobile game.

the mobile gaming industry. Blockchains can enhance mobile games with *game and rules transparency, assured asset ownership, secured asset trade, easy and secure asset reusability*, and support for secure and robust *User Generated Content (UGC)*.

Everyone participating in a blockchain network can read all the records stored in the blockchain, as well as the source code of the smart contracts. Thus, if the functionality of a game or the rules of a game are written in a smart contract deployed on the blockchain, everyone, including players can access and read these data. This is a great improvement compared to legacy systems, where usually these data are stored in private servers and are provided to the players by the game company only indirectly, often not even explicitly. So, in a blockchain-based game the players can be sure that the rules of the game will be respected and that the game is played fairly by everyone. One such example is a dice game that generates a random number from one to six, in order to choose a winner. If the functionality that implements the generation of the number is in the smart contract, then the players can audit this process and be sure that no one will cheat, not even the gaming company that provides the game.

Furthermore, different game organizations can join the same blockchain platform and cooperate, in the sense that one company can develop a game based on the content of the game of another company (logic extension and asset reusabil-



https://doi.org/10.26219/heal.aueb.9346

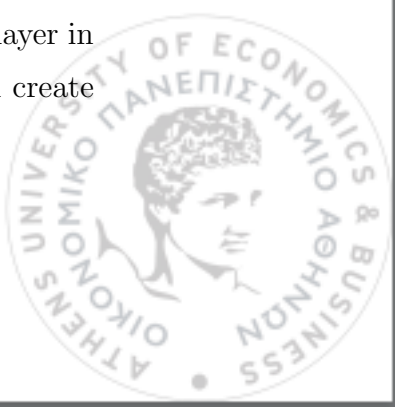
ity), like the KotoWars game.[12] If the assets of a game, e.g., shields and guns are represented as tokens in a public blockchain, which is open to everyone, then these tokens will "live" in the blockchain network, and not in the server of the specific game. Thus, everyone can also (re)use them outside the game, e.g., in another game that supports the same assets, without the intervention of the first gaming company. Furthermore, this guarantees asset ownership with transparency and consistency of asset rules, and other similar useful properties, leading to trusted and simpler trading transactions of assets among players. More importantly, this can also be done across games. Note that distrust about rare and expensive assets has been a limiting factor in some cases in the past and a key motivating factor for the introduction of DLTs.

Blockchain technology can also improve in-game features and solve some of the problems faced by the (mobile) gaming industry. A game that leverages blockchain technology can expand and improve in-app payments using cryptocurrencies. Until recently, mobile games used mostly conventional payment methods that require a third trusted party in order to be reliable. With the use of blockchains and smart contracts, and in particular due to the transparency and openness they provide, in-app payments become transparent without needing a trusted third party.

With regards to advertisements, using the blockchain technology to track in-app advertisements can strengthen this process. Companies from the advertising sector or individual advertisers can provide personalized and more importantly context-aware advertisements. In an IoT mobile game (as the one described above), advertising companies can provide advertisements based on the users' location, in addition to other game context-related information, both macroscopic, e.g., the game or type of game, or the specific state in the game or past history and choices of the player if these become observable to all or specific parties (e.g., the advertisers).

Perhaps the most intriguing benefit of blockchains in the (mobile) gaming industry is the support for UGC. In traditional gaming all the assets of a player in the game belong to the game company, even in games, where a player can create

[12] https://kotowars.com/

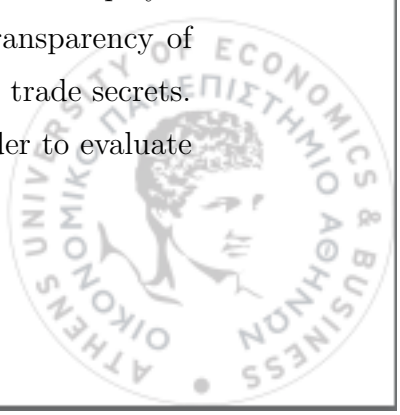


on her own her items. With the help of blockchains, this can change and lead to an open community, where any player will be able to create, own, and trade her own items. Furthermore, the items that are generated by the users will be owned by the users themselves, even if they stop playing the game. Also, DLTs can secure and strengthen this community through the aforementioned properties, e.g., avoiding incidents of stolen assets. All these properties can help the gaming industry to attract many more players and to engage them more by allowing them to create and trade assets and more generally UGC.

Therefore, it is clear that blockchains have the potential to help in the creation of new and open mobile gaming ecosystems, especially when they are combined with IoT technologies, where many different organizations (e.g., advertising companies, game studios, PoI companies, and others) can cooperate with each other to develop novel, fun, and context-aware mobile games. In our use case, it is easy for a cafeteria or a mall or any other company interested in a particular location (PoI), to deploy IoT beacons in its location, design challenges, and add them in the smart contract, without the intervention of a middleman, since there are no barriers to entry, and anyone with an Ethereum address can add a challenge. Of course, the initiator of the game smart contract, typically the gaming company, can introduce conditions to be checked automatically by the smart contract. With a case like that, new business opportunities are created, since the players will be attracted to the PoI in order to complete a challenge and may buy something from a particular store. The same also applies for the advertising sector. It is easy to create open ecosystems for the advertisers, in order to add their advertisements in the game, by just adding the URL of the advertisement within the smart contract.

Of course these benefits come with costs. Firstly, and as we show in the previous section, public blockchains involve monetary and communication overhead, which in many cases can be prohibitive. Secondly, the immutability property of the blockchain makes hard for developers to fix bugs in their code, since deployed smart contracts cannot be modified. Thirdly, the openness and transparency of blockchain makes hard to protect intellectual property or any other trade secrets.

In this paper we developed an emulation environment in order to evaluate



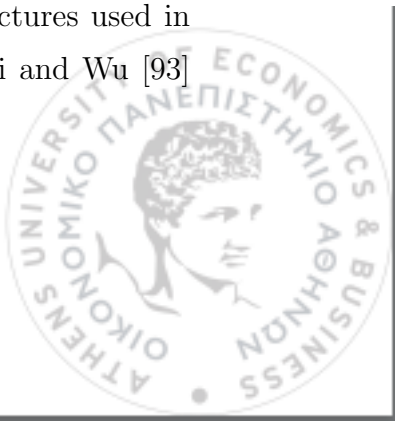
the performance and various other aspects (e.g., open ecosystems) of a location-based mobile game. Through emulation, it is easier and simpler to develop and implement different scenarios, as the ones presented above, which utilize different blockchain technologies. However, our emulation environment has some limitations. First of all, in our emulation environment, we did not consider IoT related actions. So, we did not measure the fourth KPI (see Table 5.1), which is the BLE beacon detection time. However, the detection time of beacons is one of the most important performance metrics for the specific mobile game, as if a beacon takes too long to sense a player's smartphone and send the appropriate tasks, then the player might be confused and think that she is not in the appropriate location. However, authors in [90] have calculated that the average delay is 5.8 seconds. Furthermore, in our evaluation, we did not consider business related metrics and KPIs, such as DAU and ARPU, since these metrics are addressable only through a real implementation. On the other hand, in our emulation we focused on the technologies used in the presented mobile game and how these technologies affect the performance of the system. However, we briefly describe the new business opportunities that arise with a game like the one presented above, (due to the use of IoT and blockchain technology) for the PoI companies (restaurants, cafes, and malls among others), advertisers, game developers, and even for the players. Finally, in our emulation environment, we have one application (Web app) that all actors, players, game administrators, and advertisers use to interact with the system. Also, all the functionality of the game is implemented in the three smart contracts and the ILG. In a real implementation of the game, this is not the case. The game company should have an application that will manage the game (e.g., managing the users accounts), another one to manage the challenges and so on. However, these (architectural) modifications will not affect the performance results presented above, and the results will remain at the same order of magnitude, as the bottleneck of the system is the blockchain technology.

5.1.4 Related work

Companies in the gaming industry have already integrated blockchains into their products. One of the first blockchain-based games, introduced in 2014, is Huntercoin.[13] Huntercoin is a game, where players control a "hunter", who explores a 2D virtual universe residing within the blockchain. The goal of this game is to collect coins, which can be exchanged with fiat currency. Thus, the blockchain is used mainly for mining, storing, and trading these in-game coins. As we have mentioned earlier, the first blockchain-based game that became a success is CryptoKitties. It is a game that players can buy and "breed" kitties, which are represented by blockchain-based tokens. In particular, the game uses the ERC 721 token standard to create and manage tokens that represent the in-game assets. Another game is Decentraland,[14] which is a distributed platform for a shared virtual world that enables players to build on top of it. It uses the Ethereum blockchain and utilizes the ERC-20 token standard in order to allow players to trade goods and services provided by themselves. Finally, another recent game that shows the benefits of blockchains in gaming is KotoWars. In this game, players are able to duel each other, using their assets ("kitties") earned from the CryptoKitties game. This game is of particular interest, as it demonstrates how blockchain technology can enable interoperability among different games. It is clear that many games that utilize blockchains have been developed, which take advantage of the features and properties of the DLTs. However, there are not games or mobile games that combine DLTs with the IoT. In this paper, we present and evaluate various aspects of a context-aware mobile game that has been developed for the SOFIE project and combines IoT devices and blockchains.

In addition to gaming companies, many research efforts have considered utilizing blockchain technology within gaming. Min et al. [91] survey and categorize existing blockchain-based games according to the properties and features of the blockchain they use, while Min and Cai [92] explore architectures used in blockchain-based games and the security issues that arise, and Cai and Wu [93]

[13] https://xaya.io/huntercoin-legacy/
[14] https://decentraland.org/



propose a gaming avatar framework that provides interoperability across multiple games and blockchains. On the other hand, Kalra et al. [94] have used blockchain technology, not for the game itself, but for implementing auxiliary functionalities for the games. In particular, they have designed an application that addresses two seemingly different problems in online gaming, cheating and DDoS attacks to game servers. Although all these efforts illustrate the advantages, as well as the potential for using blockchain in gaming, they do not provide insights about the impact of this technology in gaming applications. In this work, we fill this gap by discussing and evaluating the impact of the inclusion of blockchains to mobile gaming and propose the use of interledger technology in order to exploit both public and private ledgers, while achieving the high performance expected in gaming.

This work summarizes our efforts in evaluating a context-aware mobile game, as part of the H2020 project SOFIE. Manzoor et al. [90] describe in detail this location-based mobile game. They present an architecture for such games and then they describe a specific game, which they have designed and implemented, investigating the latency and the throughput aspects of the system. The game is a full implementation with a user interface and can be played on real mobile devices, using real Bluetooth beacons or Wi-Fi access points as beacons. Because of their approach, they have to make specific implementation decisions and limit the choices and parameters they investigate, but they also obtain concrete (but technology and solution specific) performance metrics, e.g., for the (mean) time to detect a beacon and the number of players that can be supported. In our work, we use mainly emulation in order to investigate a larger universe and potential applications and mobile gaming ecosystem (but we do actual implementation for all smart contracts and related DLT functionality). This allows us to research many more architectural scenarios that jointly utilize public and private blockchain technologies, determine the corresponding trade-offs, and investigate many more performance metrics.

5.1.5 Conclusions and future work

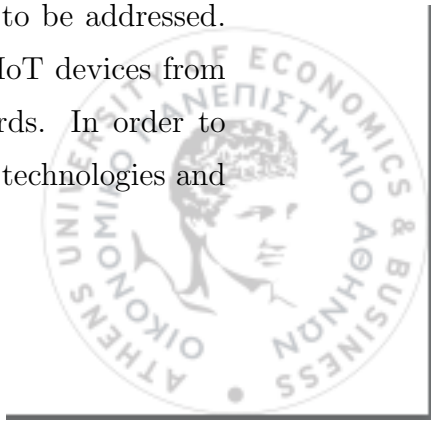
In this work, we have discussed how DLTs and IoT devices can be utilized to enable and expand novel mobile gaming ecosystems and improved in-game features, such as in-app payments without third parties, and personalized and context sensitive advertisements, in an open and secure manner. Furthermore, we described a specific prototype of a location-based mobile game and we evaluated it from various aspects based on defined KPIs.

At a high level, our evaluation shows the gains that can be achieved in terms of cost and performance, when public and private ledgers are combined. We concluded that using only one permissioned ledger is better in terms of time and cost. However, using two ledgers, a public one and a private one, is better in terms of transparency, trust, and openness. In particular, we claim that using (public) Ethereum for increased trust and convenient payments and Fabric for high performance, scalability, and low cost, is better than using only one type of ledger for many types of games and business logic situations.

Our paper is a first step towards assessing the impact of the IoT and DLTs in the mobile gaming industry. Our findings indicate great potentials and for this reason, it is in our future plans to further investigate this field. Our emulations can be complemented by experimentation with real mobile games and more IoT devices, which will provide us with insights about potential deployability issues, as well as by analytical evaluation through system modeling, which can helps us evaluate large scale scenarios, as well as predict market directions.

5.2 Smart contract-based digital twins

IoT is envisioned to be an ecosystem of interconnected devices meaning to provide a multitude of services to improve the quality of our lives. However, as we have mentioned, there are some IoT challenges that need to be addressed. First of all, IoT systems are fragmented. There is a plethora of IoT devices from different manufacturers that use different protocols and standards. In order to deal with that, the WoT W3C working group [17] leverages Web technologies and



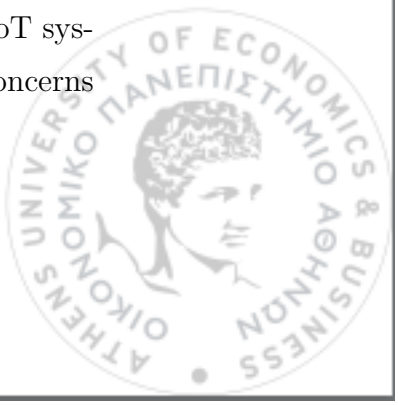
standards to deal with interoperability issues among different types of IoT devices and platforms by developing an *interoperable* IoT architecture. The WoT builds on well known Web protocols and enables IoT device discovery and access using REST-based APIs, over popular application layer protocols, such as HTTP(s). The WoT comes with a lot of benefits and address the problems of fragmentation and lack of interoperability in IoT.

In addition, securing IoT services, i.e., sensing and in particular actuation, requires complex security solutions, using advanced cryptographic techniques and algorithms. However, these solutions have not been designed for the IoT, in which many IoT devices are usually less powerful than typical computers, hence they cannot perform complex security and cryptographic operations. Therefore, more lightweight solutions that take into consideration the limitations of IoT devices are required. Furthermore, since the real world can be directly impacted by the IoT, and the IoT devices are even physically exposed to many, if not to all users, security, safety, and privacy are serious concerns. One solution in the direction of securing IoT systems and IoT devices is the use of digital twins. A digital twin is the virtual replica of a physical (IoT) device, system, or asset [95]. In most IoT systems, digital twins are typically used for testing, monitoring, and simulating IoT devices. On the contrary, we propose the use of digital twins as an *isolation, protection, and indirection mechanism*, instead of interacting directly with the actual IoT devices. In particular, users instead of communicating with the actual IoT device, they communicate only with its digital twin. Then, all valid state modifications of the (virtual) digital twin will be securely transmitted to the actual IoT devices, which will consequently perform the requested actions.

Digital twins usually operate in a more powerful and secure network location than the actual device, such as a Web server or a Cloud. Many companies, such as Amazon and Microsoft, are already providing such services.[15,16] Nonetheless, these solutions are vendor specific and often result in "vendor lock-in" situations. Furthermore, these services lack transparency; given the pervasiveness of IoT systems and their access to sensitive and private data, security and privacy concerns

[15]https://aws.amazon.com/iot-core/?c=i&sec=srv

[16]https://azure.microsoft.com/en-us/services/digital-twins



arise. Finally, these systems may occasionally suffer from outages due to their centralized nature, making IoT devices inaccessible.[17] To address all these, we propose using DLTs to create and "host" secure and reliable digital twins.

In this work [96], we summarize our efforts presented in [97, 98] for designing smart contract-based digital twins for WoT-enabled IoT devices and using them as transparent proxies to perform actions (actuation and sensing) on physical IoT devices. In particular, the contributions of this work are:

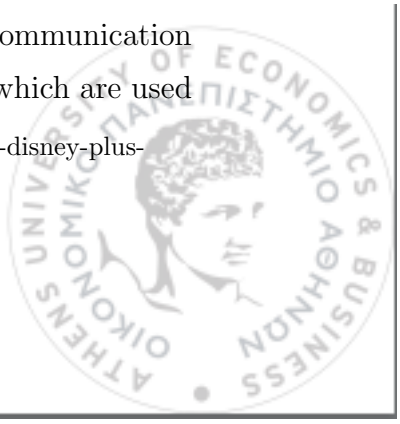
- We design and present secure, decentralized, reliable, auditable, and flexible smart contract-based digital twins of WoT "virtual entities", i.e., entities that integrate one or more IoT devices.
- We implement our solution in two different blockchain, Ethereum and Fabric. Each one of them constitutes a better fit for different use cases and purposes.
- We design, implement, and evaluate an IoT system that uses the smart contract-based digital twins to offer secure sensing and actuation.

Our solution achieves the following. It improves interoperability, by adopting the WoT architecture, auditability, since every interaction is recorded immutably on the DLT, enhances security by removing the need for a trusted entity that hosts the digital twin, and increases system availability by implementing its core functionality in a smart contract. In addition, we make consumers oblivious to IoT devices and IoT device (vendor-)agnostic, since they communicate with the digital twin and not the actual IoT devices, allowing abstractions and virtualization.

5.2.1 Background – Web of Things

The WoT architecture [99] structures well-known Web protocols and tools for connecting IoT devices to the Web. In the WoT architecture communication model, IoT devices are made available through REST-based APIs, which are used

[17] https://www.theverge.com/2021/12/7/22822332/amazon-server-aws-down-disney-plus-ring-outage

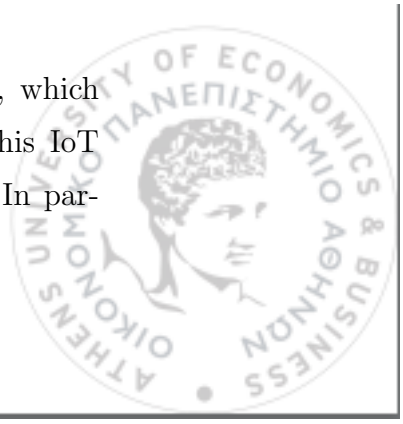


by *consumers* to access device *properties*, to trigger device *actions*, as well as, to receive device-generated *events*. In order to improve the *interoperability* and *usability* of IoT platforms, the WoT model uses a common format for describing IoT devices, referred to as the *Thing Description (TD)* [100]. The TD is machine-readable and includes metadata about the IoT device, such as its identifier, a title, and security definitions among others, as well as, IoT device properties, actions, and events that can be accessed or invoked through Web *links* and *forms*.

Listing 5.1: Thing Description for a smart lamp.

```
{``@context'': ``https://www.w3.org/2019/wot/td/v1'',
 ``id'': ``lamp1'',
 ``title'': ``My lamp'',
 ``securityDefinitions'': {...},
 ``security'': [...],
 ``properties'': {
    ``status'': {
       ``type'': ``string'',
       ``forms'': [{``href'':
                   ``.../things/lamp1/status''}]
    }
 },
 ``actions'': {
    ``toggle'': {
       ``type'':``boolean'',
       ``forms'': [{``href'':
                   ``.../things/lamp1/toggle''}]
    }
 },
 ``events'': {...} }
```

Listing 5.1 provides an example of a WoT TD for a smart lamp, which is encoded using JSON-LD. This TD includes information about how this IoT device can be accessed, i.e., via an HTTP request to the specified URI. In par-

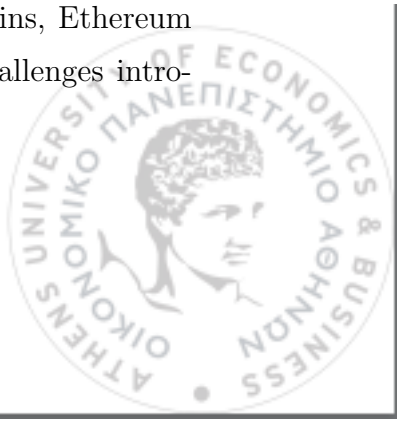


ticular, to switch on/off the smart lamp, a POST HTTP request should be sent to https://example.com/things/lamp1/toggle. Similarly, in order for someone to read the current status of the smart lamp, a GET HTTP request has to be sent to https://example.com/things/lamp1/status.

5.2.2 Smart contract-based digital twins design

In this section, we present the design of the smart contract-based digital twins of IoT devices. As IoT devices, we consider sensors and actuators that follow the WoT standards and specifications, namely, they expose a TD. The use of WoT addresses the problems of fragmentation and interoperability that IoT faces, by providing a set of standardized technologies, following the well-known Web paradigm. Furthermore, we follow the most typical architectural pattern in legacy IoT systems, where the IoT devices are paired with a more powerful device than them, a gateway. In this pattern, users instead of interacting directly with the IoT devices, they interact with the gateway. Therefore, in our design, we consider IoT devices that are paired with WoT gateways. Each one of the WoT gateways in the system represents a WoT "virtual entity." A virtual entity is the composition of one or more IoT devices, e.g., a building consisting of several IoT devices. A virtual entity provides a *single* WoT TD that includes the actions, the properties, and the events of all its IoT devices. So, if in our system there are two WoT gateways, including many IoT devices, then we have two virtual entities and two TDs accordingly. This is depicted in Figure 5.6. IoT devices are identified by URIs (see Listing 5.1) that can be used for performing sensing and actuation processes. In our design, the URIs are composed of the URL of the WoT gateway, plus the name of the IoT device, plus the action or data that is provided from the corresponding IoT device, e.g., http://gw1.iot/lamp1/toggle.

We design and implement digital twins on the two most popular representatives of public/permissionless and private/permissioned blockchains, Ethereum and Fabric, taking under consideration the limitations and the challenges introduced by each type.



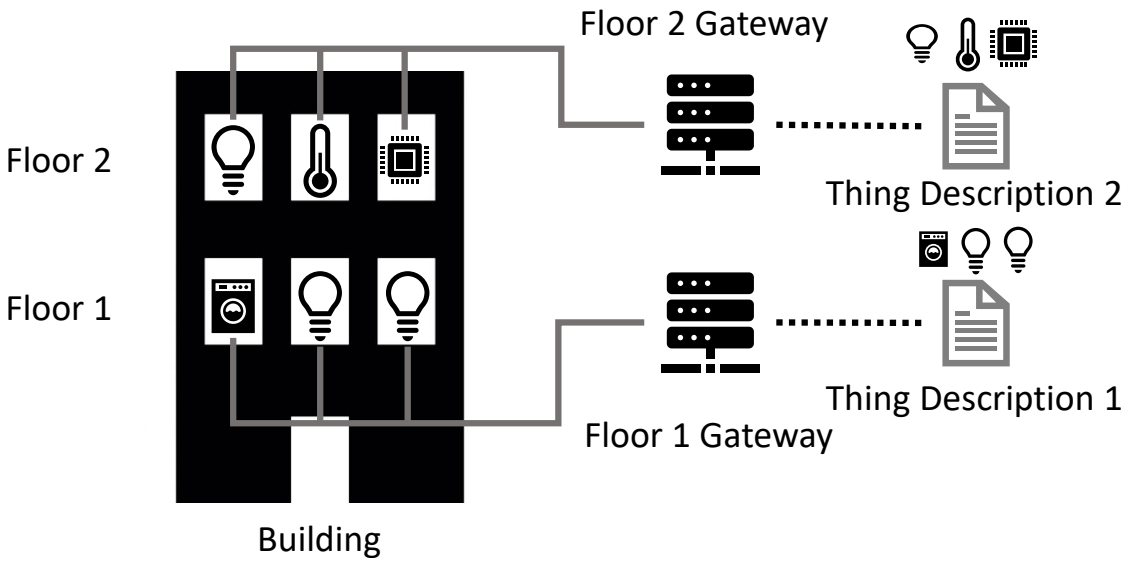
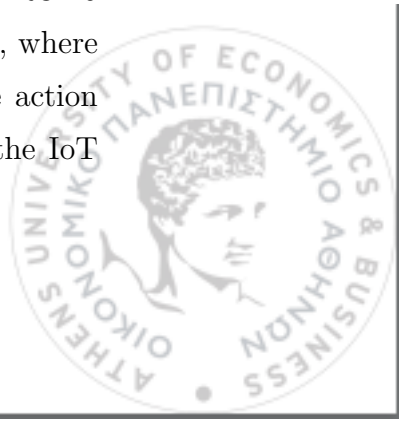


Figure 5.6: The Internet of Things/Web of Things architecture considered in the smart contract-based digital twins design.

Ethereum-based digital twins

First, we introduce the design based on the public, permissionless Ethereum blockchain. Since Ethereum smart contracts cannot communicate (or send requests, call a REST API, etc.) with an application, or anything that is off-chain, we cannot consider a design, in which the smart contract communicates with the WoT gateway directly to learn the available actions and properties of the virtual entity. Thus, we decide to embed the TD of the virtual entities in the smart contract. However, in Ethereum, it is too costly to store the actual TD in the smart contract. To avoid having enormous monetary costs, we store in the smart contract a stripped down version of the virtual entities' TDs. Each action of the TD is stored in a data structure in the smart contract, called *actionsList*. This structure contains (a) an action name, (b) the input parameters, including the type and the number of parameters, (c) the defined price for each action, expressed in ERC-20 tokens, and (d) the status. This is shown in Figure 5.7. and in Figure 5.8, where the design and the source code of the smart contract is illustrated. The action name is composed by the name of the gateway underscore the name of the IoT



https://doi.org/10.26219/heal.aueb.9346

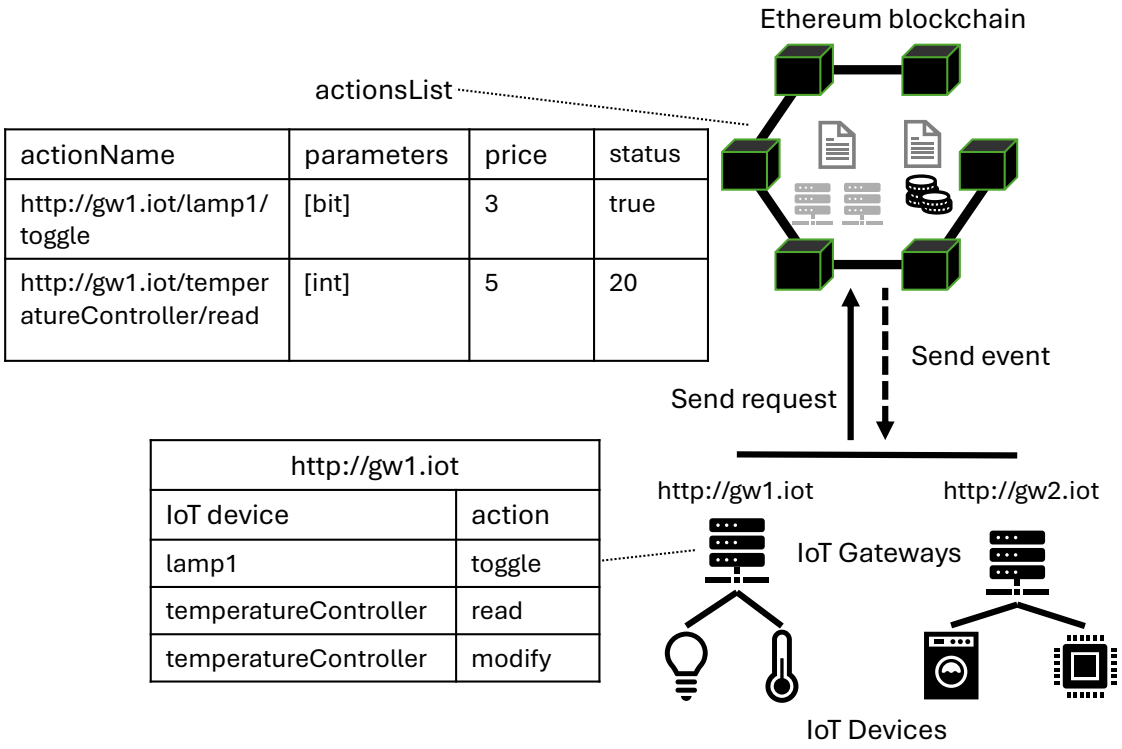
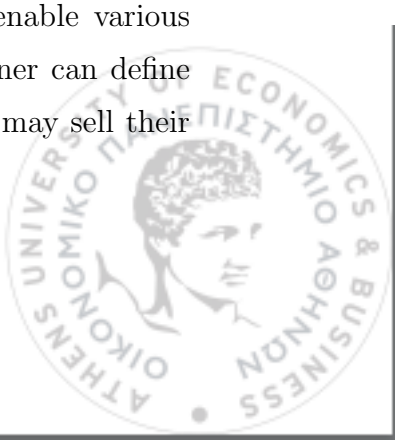


Figure 5.7: Smart contract-based digital twin's structure on the Ethereum.

device underscore the name of the action. The parameters field is essentially the dataschema that describes the data format of the actions, properties, and events of a TD.[18]

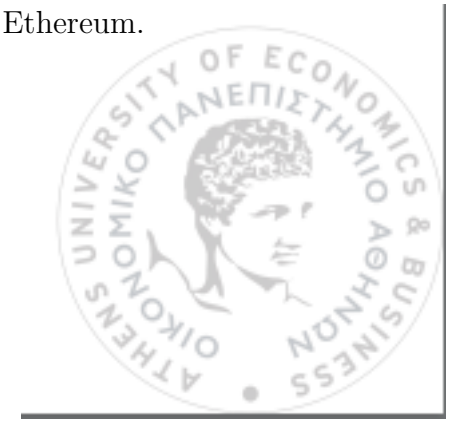
Finally, since Ethereum is public and everyone can have access to the smart contracts deployed on it, to restrict the access on the IoT devices, we propose a form of access control. Consumers can gain access to the smart contract-based digital twin, by obtaining some owner-specific tokens (owner is the person, who owns the virtual entity), implemented using the ERC-20 token standard. Therefore, only the consumers that have obtained these tokens can perform operations on the virtual entity. To obtain some tokens, consumers have to communicate with the owner directly, offline, and off-chain. This feature can also enable various business models. For example, depending on the use case, the owner can define a price for these tokens and sell them to consumers, or consumers may sell their

[18]https://www.w3.org/TR/wot-thing-description/#dataschema



```
pragma solidity >=0.4.16 <0.9.0;
import "./ERC20DT.sol";

contract DigitalTwin is ERC20DT {
    struct Actions { string actionName; string[] parameters; uint price; string status; }
    Actions[] public actionsList;
    event PerformAction(strin actionName, string[] parameters, address client);

    function performAction(uint actionIndex, string[] parameters) public {
                require(actionIndex < actionsList.length);
                Actions action = actionsList[actionIndex];
                require(balances[msg.sender] > action.price);
                require(parameters.length == action.parameters.length);

                balances[msg.sender] = balances[msg.sender] – action.price;
                balances[address(this)] = balances[address(this)] + action.price;
                emit PerfromAction(action.actionName, action.parameters, msg.sender); }

    function endOfAction(uint actionIndex, address client, bool success, string status) public {
                require(msg.sender == owner);
                Actions action = actionsList[actionIndex];
                if (success) {
                   balances[owner] = balances[owner] + action.price;
                   balances[address(this)] = balances[address(this)] – action.price;
                   status = status; }
                else {
                   balances[client] = balances[client] + action.price;
                   balances[address(this)] = balances[address(this)] – action.price; } }

    function getAction() { … }

    function addAction(string actionName, string[] parameters, uint price, string status) public {
                Actions action = Actions(actionName, parameters, price, status);
                actionsList.push(action); }

    function deleteAction() { … }
    function modifyAction() { … }
}
```

Figure 5.8: Source code of the smart contract-based digital twin on the Ethereum.



https://doi.org/10.26219/heal.aueb.9346

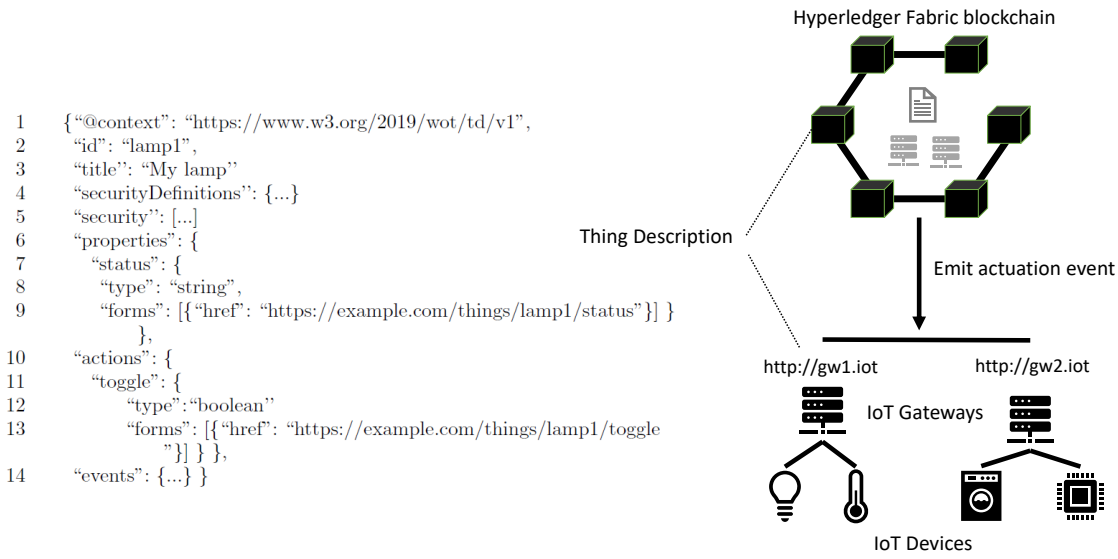


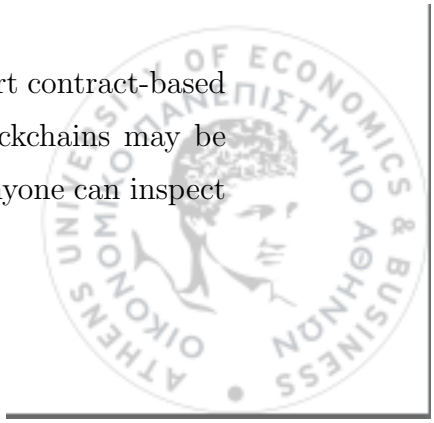
Figure 5.9: Smart contract-based digital twin's structure on Hyperledger Fabric.

tokens to other consumers, or even consumers can form "alliances" and purchase a large amount of tokens, potentially achieving a discount and forming a secondary market.

As we observe from the simplified smart contract, shown in Figure 5.8, the owner, in order to "create" a digital twin of an IoT device, he should call the `addAction` function and insert all the actions with the corresponding parameters and the price for these actions that the IoT device supports. Similarly, to modify an action, e.g., change the price of an action, or remove completely an action, he calls the `modifyAction` and the `deleteAction` functions, which just modify accordingly the `actionsList`. Then, consumers can call the `performAction` function of the smart contract-based digital twin to request and perform actions on the provided IoT devices (a more detailed flow on how consumers request to perform actuation or sensing processes is presented in the next section).

Hyperledger Fabric-based digital twins

In the previous subsection, we presented the design of smart contract-based digital twins in a public blockchain. However, using public blockchains may be inappropriate in some use cases, where privacy is needed, since anyone can inspect

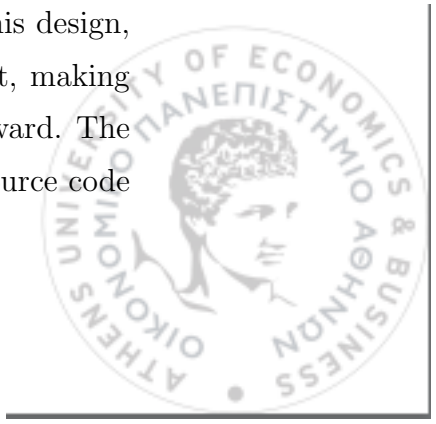


```
Const { Contract } = require('fabric-contract-api');

Class DigitalTwin extends Contract {

    //initialize ledger with the TDs
    async initLedger (ctx) {
    const TDs = [{…}, {…}];
    for (let i=0; i<TDs.length; i++) {
        //write to the ledger
        await ctx.stub.putState(TDs[i]);
    } }

    async performActuation (ctx, tdNumber, action, parameters) {
        //read the appropriate TD from the ledger
        const tdAsBytes = await ctx.stub.getState(tdNumber);
        const tdAsJSON = JSON.parse(tdAsBytes.toString());

        //check that action exists in the TD
        //check that the number of the parameters are correct

        tdAsJSON.Actions.action = parameters.action;
        tdAsBytes = Buffer.from(JSON.stringify(tsAsJSON));
        //send event
        ctx.stub.setEvent('performActuation, tdAsBytes);
        //write the result of actuation on the ledger
        ctx.stub.putState(tdAsBytes);}

    async getThingDescription (ctx) { … }
    async getAction (ctx, tdNumber, action) { … }
    async addAction (ctx, tdNumber, actionName, action) { … }
    async getProperty (ctx, tdNumber, property) { … }
    async addProperty (ctx, tdNumber, propertyName, property) { … }
    async addTD (ctx, tdNumber, td) {
        const temp = JSON.parse(td);
       await ctx.stub.putState(tdNumber, Buffer.from(JSON.stringify(temp))); }
}
```

Figure 5.10: Source code of the smart contract-based digital twin on the Hyperledger Fabric.

and read a public blockchain. The latter, in our case, means that anyone can find out the provided actions, as well as, who has invoked which actions. Therefore, to avoid that, we have also designed and implemented smart contract-based digital twins in a private, permissioned blockchain, and in particular in Fabric.

Fabric does not introduce monetary costs, as opposed to Ethereum, so it is not critical to have a minimal design for the digital twin. Thus, in this design, we can store the whole TD of the virtual entities in the smart contract, making the implementation of the smart contract much simpler and straightforward. The design in Fabric is shown in Figure 5.9 and a simplified version of its source code



is shown in Figure 5.10. Additionally, in Fabric, smart contracts can communicate directly with the "outside" world. Thus, in the Fabric-based design, the smart contract can communicate directly with the WoT gateways, instead of communicating indirectly, through events. However, in a design like that, every peer participating in the network that has deployed the smart contract, would send a request on the gateways. So, if in a system there are 100 peers and everyone has to execute the smart contract, then all will send a request on the gateway. The gateway will eventually end up receiving 100 requests for the same actuation/sensing request. This can lead in DoS attack to the WoT gateways, if there are too many peers in the network. In order to address this challenge, the smart contract does not send directly any request to the WoT gateway, but as in the Ethereum-based design, it generates an event, which is caught by the WoT gateways. Furthermore, in Fabric, there is no need for creating an external access control mechanism, as we did in Ethereum using the ERC-20 tokens, since Fabric is a permissioned blockchain and the access is already checked by the MSP, which is a trusted authority. Finally, in this design we do not have an end actuation function, since this function is useful to return the tokens back to the consumer in case that the action does not completed successfully. In Fabric, we do not introduce any tokens, thus this function is useless.

5.2.3 IoT system overview

In this section, we present an overview of the proposed IoT system, as a whole, and its architecture, which integrates the proposed smart contract-based digital twins to offer secure and interoperable sensing and actuation services to consumers. The architecture of the system is illustrated in Figure 5.11. The system is composed of the following entities and components:

- WoT gateways and IoT devices
- The blockchain network
- The smart contract-based digital twins

https://doi.org/10.26219/heal.aueb.9346

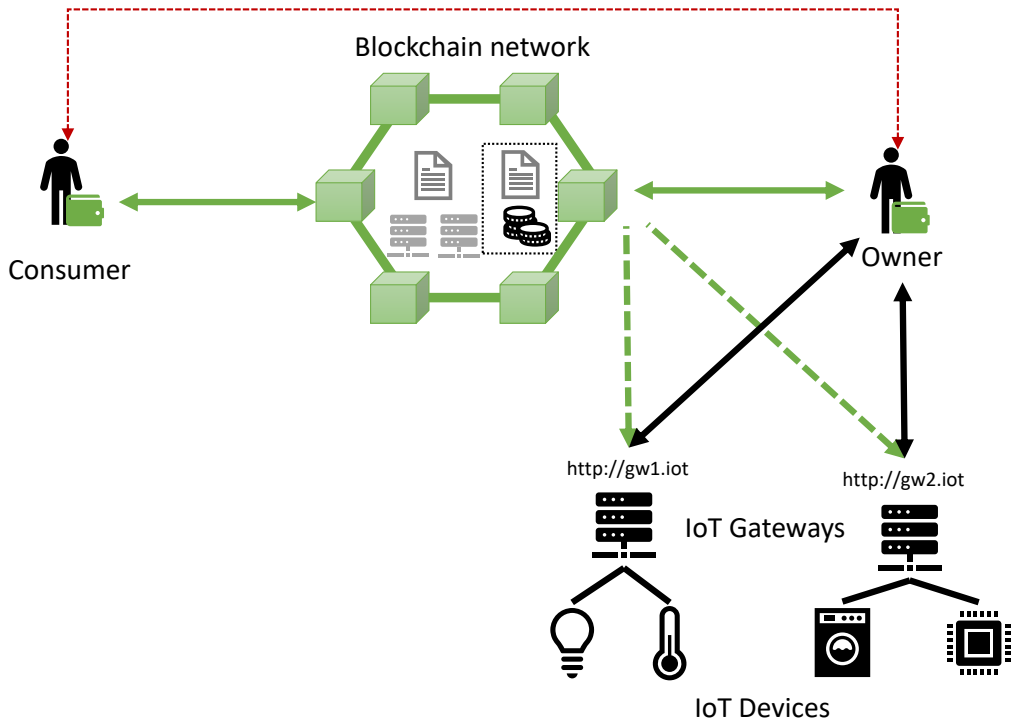
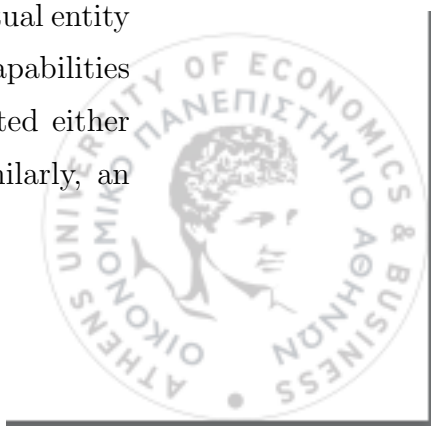


Figure 5.11: An overview of the IoT system's architecture that includes smart contract-based digital twins.

- Owner(s) that administer the WoT gateways and the IoT devices
- Consumer(s) that interact with IoT devices
- An ERC-20 token smart contract (applied only in the Ethereum-based design)

As we observe from the Figure 5.11, consumers do not access IoT devices directly, instead they interact with their smart contract-based digital twins, which are deployed on the blockchain (either on Ethereum or Fabric). Consumers, to interact with the blockchain, should own a blockchain wallet.

As we mentioned above, each WoT gateway constitutes a WoT virtual entity and each virtual entity provides a single WoT TD, which contains the capabilities of all IoT devices. The actions of the virtual entity can be implemented either by a single IoT device, or by orchestrating multiple IoT devices. Similarly, an



action may correspond to multiple interactions with an IoT device, enabling *mass actuation or sensing*. For example, an action "turn on all the lights of the floor" results in instructing multiple light bulbs to be switched on. The WoT gateways are responsible for mapping an action of the virtual entity to the corresponding action(s) of the real IoT device(s).

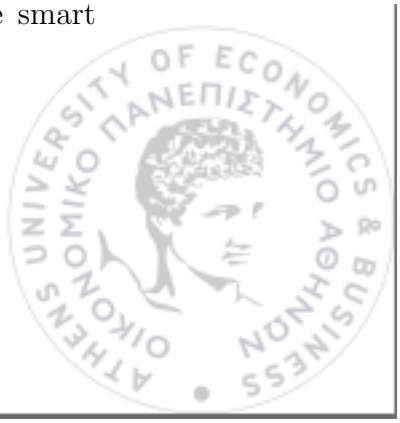
From a high level perspective, the entities in our system interact with each other as follows. Initially, the owner physically deploys the IoT devices and pairs them with a WoT gateway. Then, she creates the digital twin of the virtual entity, composed of all IoT devices paired with the corresponding gateway, and deploys them on the blockchain network. Depending on the use case, the design of the digital twin slightly changes. When the setup has been completed, a consumer can gain access to the provided actions, hence to the IoT devices. To that end, he has to obtain permission from the owner. Then, the consumer learns from the smart contract all the available actions and the required parameters. From this point on, a consumer can perform an IoT device access request by sending a transaction to the smart contract-based digital twin. The smart contract verifies the transaction and forwards the request to the appropriate WoT gateway. Finally, the WoT gateway forwards the request to the appropriate IoT device(s), which eventually perform the requested action. The communication between the WoT gateway and the IoT devices is specified by the device vendor (which is out of the scope of this work). The requested action may lead IoT devices to produce (sensitive) data. We should note here, that these data are not stored in the smart contract nor in the ledger. As in legacy IoT systems, data can be stored in a cloud, in the (WoT) gateways, in the endpoint devices, or even in a decentralized storage system, such as the IPFS. This choice does not affect our design (in both digital twins implementations) or the system as a whole. However, depending on the selected option, small changes might be needed. The life-cycle of the system involves the following phases.

Setup phase

Initially, the owner physically deploys the IoT devices and the corresponding WoT gateways. A WoT gateway includes the TD of the virtual entity and exposes all the actions, properties, and events of that virtual entity. Furthermore, the owner creates a smart contract (depending on the use case, she chooses the appropriate design, Ethereum-based or Fabric-based), which represents the digital twin of the virtual entity and she deploys it on the blockchain network. The smart contract's address on the blockchain is considered well-known.

In both cases, the owner has also to configure the WoT gateways to "watch" the blockchain for events generated by the corresponding smart contract. In case of the Ethereum blockchain, she should also create and deploy another smart contract, which creates and manages the owner-specific tokens, implementing the ERC-20 token standard. A consumer, in order to acquire tokens, contacts the owner to come to an agreement on the price of a token. Then, he has to pay the agreed amount of money to the owner, in fiat currency. The payment can also be done with cryptocurrencies, e.g., ether or btc, with small changes on the source code of the ERC-20 smart contract. In the current design, this process takes place offline and off-chain using fiat currency in order to keep consumers-owner business relationship private. When the payment is completed, the owner transfers the agreed amount of the blockchain-based tokens to the consumer's blockchain wallet. This process does not happen in the Fabric-based design, since Fabric is a private blockchain. Thus, in Fabric, a consumer, in order to be able to interact with the smart contract-based digital twin, he has to obtain an identity, namely a X.509 certificate. To do so, he communicates with the admin of the network and the corresponding CA to obtain the certificates.

Finally, consumers can learn all the available operations provided by the IoT devices, the appropriate parameters, and the cost for each action (this only applies to the Ethereum-based design), by just calling a function of the smart contract, called `getAction`, which returns them.

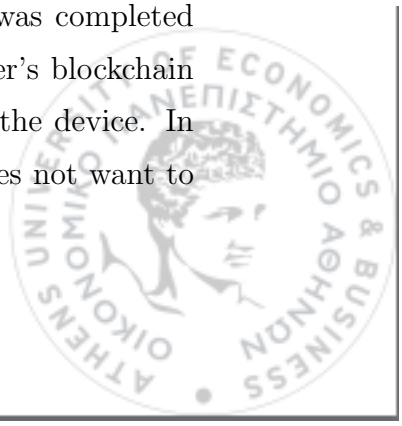


IoT device access

From this point on, a consumer can perform an IoT device actuation request, or access data request. To do so, he sends a transaction to the smart contract-based digital twin that includes the requested action and the corresponding parameters. The smart contract-based digital twin verifies that the transaction is valid. Namely, it checks that the action exists in the actionsList or in the TD and that the parameters are correct. From now on, depending on the case, the flow slightly changes.

(a) Ethereum-based design. In case of the Ethereum-based design, the consumer has to purchase the required number of ERC-20 tokens. To do so, the consumer pays the owner and the owner calls the `transfer` function of the ERC-20 smart contract to transfer the agreed tokens to consumer. So, the smart contract-based digital twin has to verify also that the blockchain address of the consumer has the required number of tokens. In order to perform this verification, it interacts with the smart contract that manages the ERC-20 tokens, and calls the `balanceOf` function, which returns how many tokens an address has. Then, these tokens are deposited to the smart contract's address. If all requirements are met, then an event, named `PerformAction` is triggered, as we can see from Figures 5.7 and 5.8. The event includes the action name, the parameters, and the blockchain address of the consumer. The address of the consumer is needed in order to send the tokens back to him, if the action will not be completed successfully.

The event is eventually "caught" by the appropriate WoT gateway, which finds out which device should serve the request. Subsequently, the WoT gateway forwards the request to the appropriate IoT device(s), which perform the requested action. When the action has been completed, the WoT gateway configured with the owner's blockchain wallet, sends a transaction to the blockchain. In particular, it calls the function `endOfAction` to transfer the tokens from the smart contract-based digital twin's address to the owner's address, if the action was completed successfully, otherwise, it transfers the tokens back to the consumer's blockchain wallet. Furthermore, it sends to the blockchain the new status of the device. In case of consumer not spending all of his ERC-20 tokens and he does not want to



use the provided services anymore, he has two options. The first option is to give back the remaining tokens to the owner and receive back fiat currency. The second option is to sell his tokens to other consumers. The latter allows the creation of secondary markets and enables various new business structures.

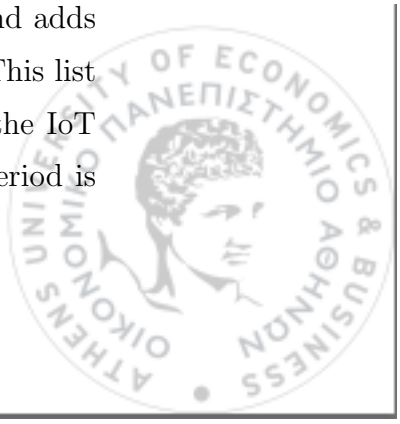
(b) Hyperledger Fabric-based design. In Fabric, the flow is the same with small differences. After the validation of the transaction, the smart contract-based digital twin generates an event, which is "caught" by the WoT gateway. Then, the WoT gateway forwards the request directly to the appropriate IoT devices, which eventually perform the requested action.

IoT device management

The smart contract-based digital twins include either the whole TD or a stripped-down version of it, the actionsList. Both of them contain all the properties, actions, events, and status of the virtual entity. These structures in the smart contracts can only be modified by the owner. In particular, the owner can add a new entry, as well as, modify, or even delete an existing one. These operations do not involve any interaction with the consumers. Additionally, an owner can replace a physical IoT device; since the consumer interacts with the virtual entity, which is device independent. This replacement affects only the configuration of the corresponding WoT gateway and its smart contract-based digital twin.

Ephemeral interactions and revocation

There might be cases, where the owner should grant ephemeral permissions to guests. These cases concern mainly the Ethereum-based design, which is a public blockchain and this design is used in use cases, where many mutually non-trusting entities interact with the IoT devices. In such cases, the guest should come to an agreement with the owner for the price of tokens and the period that they need them. Then, the owner sends to the guest the agreed amount of tokens and adds the guest to a list, called *guestList*, along with the corresponding period. This list exists in the WoT gateways. From now on, the guest can interact with the IoT devices as any other consumer. The difference is that when the agreed period is



up, the owner calls the `withdraw` function of the smart contract that manages the ERC-20 tokens, to take back the remaining tokens from the guest. This function can only be invoked by the owner and in essence resets the token balance of a consumer, returning all the tokens back to the owner's blockchain wallet. Thus, this function can also be used in cases of security breaches. Therefore, the withdraw function can also act as a revocation mechanism. The consumers, whose tokens are revoked, can no longer have access to IoT devices and perform any action, thus the revocation is instantaneous.

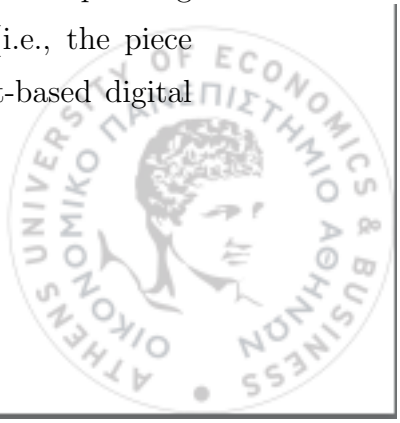
The Fabric-based design also supports revocation. In the Fabric blockchain, the admin of the network (who can be the owner of the IoT devices/gateways) can revoke the certificate of a consumer. This process creates a Certificate Revocation List (CRL), which includes the revoked consumers. Then, this CRL is included in the Fabric's network configuration and the consumer can no longer have access to the blockchain's network nor to the deployed smart contracts.

5.2.4 Implementation and evaluation

Implementation

We developed a proof of concept implementation of the presented IoT system. Our WoT gateway is based on Eclipse's Thingweb. As IoT device, we emulated a smart lamp. The smart contract-based digital twin in Ethereum was implemented using Solidity, while in Fabric, it was implemented using JavaScript. Both implementations of the smart contract-based digital twins are open-source and available at GitHub,[19] while a simplified version of them is show in Figures 5.8 and 5.10. Furthermore, our client application for Fabric, which is responsible for acquiring the certificate, and interacting with the smart contract-based digital twin through the Fabric gateway, is implemented in JavaScript using the Fabric SDK, while the client application for Ethereum is also implemented in JavaScript using the web3.js library. Similarly, the corresponding event listeners (i.e., the piece of software that "catches" events generated by the smart contract-based digital

[19] https://github.com/mmlab-aueb/DLT-DigitalTwins



twins) are implemented in JavaScript too.

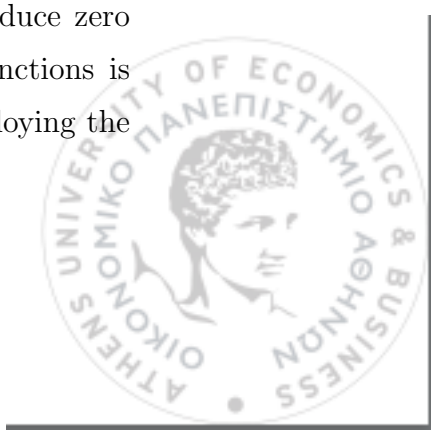
Cost and performance evaluation

Cost evaluation concerns only the Ethereum-based design, since only Ethereum introduces monetary costs. First of all, all actions performed in our system involve the invocation of a function implemented in a smart contract. In order to measure the cost required by our smart contracts, we deployed the smart contracts on the Rinkeby Ethereum test network. The cost, measured in gas units, for each of the functions is shown in Table 5.6.

Smart contract	**Operation**	**Cost measured in gas**
Digital twin	contract deployment	2341723
	performAction	52329
	endOfAction	43628
	addAction	119934
	modifyAction	41064
	removeAction	46632
	getActions	-
	getAction	-
ERC-20	contract deployment	805618
	balanceOf	-
	transfer	33664
	mintTokens	34786
	withdraw	29450

Table 5.6: Ethereum Virtual Machine execution cost (gas) of the construction building blocks of the IoT system utilizing smart contract-based digital twin.

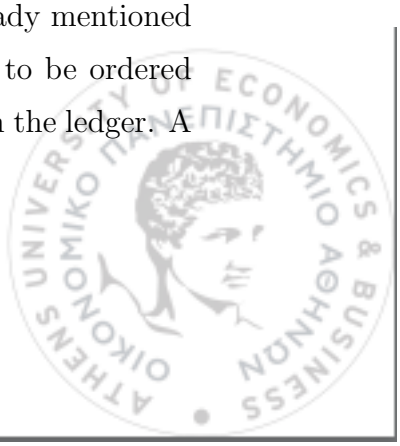
The functions that only access the blockchain for reading introduce zero cost. As we observe from Table 5.6, the cost introduced by some functions is not negligible. The most expensive functions are those required for deploying the



smart contracts on the blockchain (≈ \$49.46 and \$17.02),[20] which however take place only once. The function that is responsible for adding a new action in the *actionsList*, i.e., the `addAction` function, costs ≈ \$2.53. Similalry, the function used by consumers for invocation, i.e., the `performAction` function, costs around \$1.10. However, we should note here that the price of the actions expressed in fiat currency is not stable, since it depends heavily on the price of the Ethereum's cryptocurrency, which fluctuates highly. Thus, given a high price of ether, the cost may be prohibitive for some use cases, such as the use case of a smart home. In addition to gas, the use of the public Ethereum blockchain incurs a transaction delay, which is ≈ 15. On the other hand, read transactions do not depend on the block mining time, thus they do not incur any transaction delay.

To evaluate the performance of our Fabric-based design, we conducted some experiments to find out whether the overhead of the blockchain is bearable or not. The first experiment measures the time required for requesting an actuation/sensing process. In particular, we measured the time from the moment a user sends a request to the digital twin, until that request ends up at the WoT gateway. For comparison, we implemented an instance of our system without blockchain, i.e., the digital twin is just a Web service, as well as, an instance of the solution without digital twin, i.e., the user sends the request directly to the WoT gateway. We conduct this experiment 1000 times, i.e, we send 1000 transactions sequentially, in a single-node local testbed, i.e., the whole Fabric network runs in a single machine. The results from this experiment are presented in Figure 5.12. The average time required for the smart contract-based design is $88ms$, for the non smart contract-based design is $61ms$, and for non digital twin-based is $16ms$. As we observe from the results, the overhead added from the blockchain is ≈ 20 milliseconds, which is tolerable in almost any case. In Figure 5.12, we observe that there are several high spikes that happen often in the smart contract-based digital twin scenario. This is happening due to the ordering service of Fabric. As we have already mentioned (Section 2.1.2), transactions are sent to the orderer node in order to be ordered into blocks and then, blocks are sent back to peers to be appended in the ledger. A

[20] Prices for December 2022.



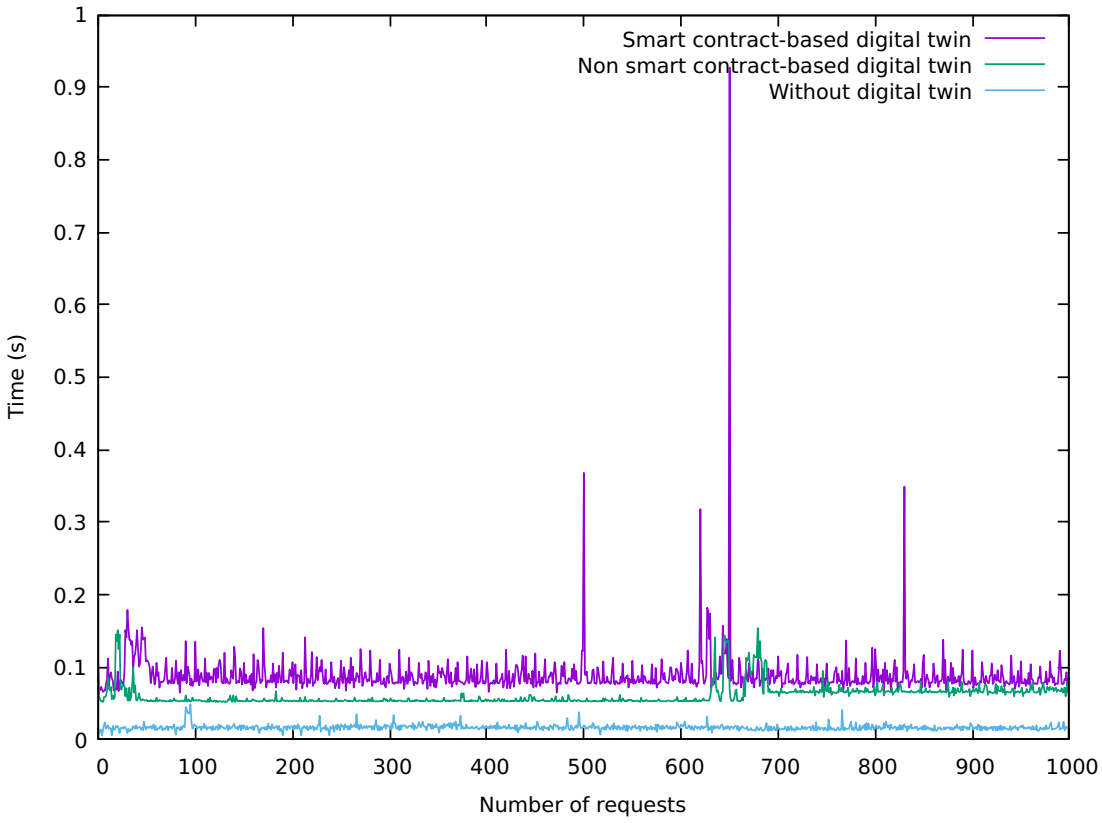
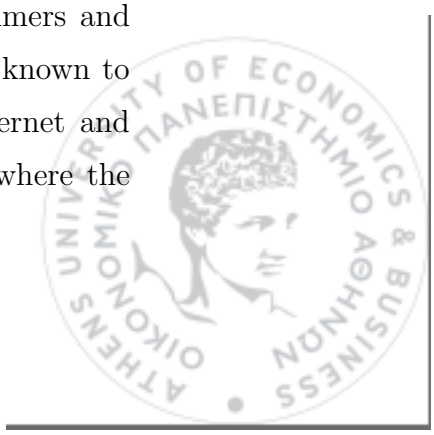


Figure 5.12: Time required for requesting an actuation/sensing process through the smart contract-based digital twin in Hyperledger Fabric blockchain.

block is created either after an adequate number of transactions has been collected, or after the expiration of a configurable timeout period. Because our experiment does not always generate many transactions, in some cases the orderer has to wait for the whole timeout period: this is a spike.

Security evaluation

Our design provides increased protection to WoT gateways. In particular, since digital twins act as an indirection mechanism between the consumers and the WoT gateways, the location of WoT gateways does not have to be known to the consumers; the gateway can even be unreachable through the Internet and disconnected from the outside world. In the Ethereum-based design, where the

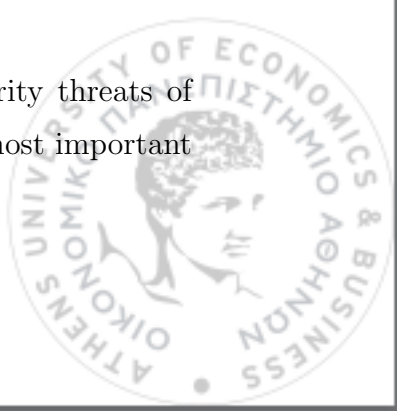


smart contract-based digital twins and the WoT gateways communicate through events generated by the digital twins, the URLs of the WoT gateways are not included in the smart contract. On the other hand, on Fabric-based design, the URLs are included in the smart contracts, since the whole TD is included. However, since Fabric is a permissioned blockchain, we assume that consumers, who have access to the smart contract-based digital twins are considered trusted by the owner. Furthermore, since the communication with the WoT gateways is achieved through events, we can remove completely the URL of the WoT gateways from the TD. Therefore, our design secures the WoT gateways from a variety of risks and attacks that are common to Web services, such as DoS attacks

Moreover, in both our designs, we provide a form of access control. In the Fabric-based design, we rely on the permission system of Fabric. In the case of the Ethereum-based design, we leverage the ERC-20 tokens standard. In particular, only users that have acquired the owner-specific tokens can request and invoke the available actions. Thus, these tokens, except for enabling new business models, also act as ACTs, offering many properties as shown in Section 4.1. However, consumers once they acquire ERC-20 tokens, they can transfer them to anyone they want. In order to avoid this, the corresponding ERC-20 smart contract can be configured to allow consumers to transfer their tokens only to the owner's address, e.g., to got reimbursed.

In addition, our system inherits all the properties of blockchains. DLTs offer reliability, robustness, and increased availability by design. Therefore, there is no single point of failure and the data, the actions, and the smart contracts are always available to the consumers. Thus, by implementing the digital twins as smart contracts, we guarantee that they will always be live and they will not depend on trusted third parties. Blockchains also offer auditability, in the sense that every interaction with an IoT device is recorded and can be revisited at any time. Therefore, in case of security incidents, blockchains can provide undeniable auditing information, offering non-repudiation.

Except of the properties, our design inherits also the security threats of the blockchains. In case of the first design (Ethereum-based), the most important



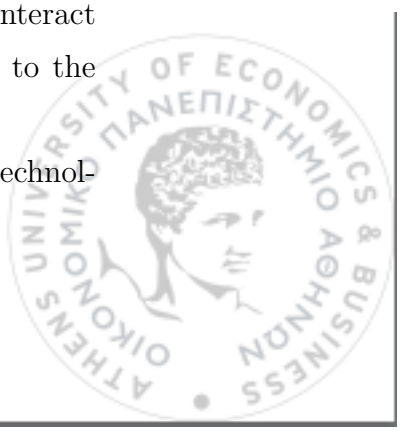
security threats are the 51% attacks, long-range attacks, eclipse and routing attacks, and private key attacks. In 51% attacks, an attacker controls the 51% of the blockchain and he can rewrite the history of the blockchain, or refuse to add transactions to it, performing a DoS attack against its users. With these attacks, the attacker gains full control of the blockchain. In addition, by compromising a user's private key, an attacker can generate transactions or perform actions on his behalf. On the other hand, Fabric's security threats differ from the popular public, permissionless blockchains. For example, 51% attacks are not significant threats, since in Fabric users are known and access is managed by ACLs. The most significant security threats on Fabric are DoS attacks, MSP compromise, private key attacks, consensus manipulation, and smart contract exploitation [101].

The only part of the system that does not involve the blockchain is the communication between the WoT gateways and the IoT devices. The security of this communication is managed by the protocols used by each vendor.

Discussion

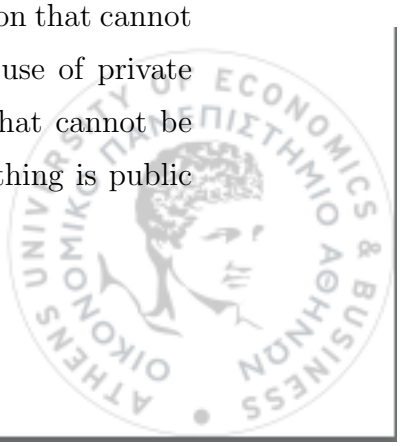
Our proposed system has many desirable usability properties, particularly compared to legacy IoT systems. Firstly, new IoT devices can be seamlessly added in our system, since the owner has only to update the WoT gateways and the TDs or the `actionsList` in the smart contract-based digital twins, without having to redeploy the smart contract or the client application. Similarly, the owner can add new WoT gateways that expose new TDs, without having to change anything to the underlay system. Furthermore, consumers are IoT device (vendor-)agnostic, since they communicate with the IoT devices through the blockchain infrastructure. Consumers do not need to know anything specific about the IoT devices, except the actions or data they provide, which they can easily learn from their digital twins. In addition, consumers do not have to deploy and use different client applications for the IoT devices, which is otherwise a very common case. Therefore, to interact with any IoT device of any vendor, they just have to send a transaction to the smart contract-based digital twin.

Our system does not inherit only the benefits of the blockchain technol-



ogy but also its drawbacks. Firstly, as we showed in the performance evaluation, blockchains incur a transaction cost and delay, which in some use cases can be intolerable. Another drawback, especially in the Ethereum-based design, is the need for storage. Blockchain nodes need to store the whole blockchain, which in Ethereum is over $1219.20GB$. However, this does not affect directly our system. Furthermore, there are many workarounds, e.g., light nodes, RPC nodes, etc. In addition, the scalability of our system is also affected. Blockchain scalability is the ability of a blockchain network to process many transactions. Towards this direction, many solutions have been proposed for scaling blockchains. Regarding the Fabric blockchain some findings show that Fabric can scale up to $20k$ TPS [102]. On the other hand, many scaling solutions have also been proposed for the Ethereum blockchain, such as sharding, proof of stake variations, or Layer-2 (off-chain) solutions. In our design, the issue regarding scalability lies to the fact that in the smart contract-based digital twin, we use a mapping (*actionsList*) to store the provided actions. So, the question arises as to whether the system is affected by the size of this mapping. Nevertheless, as we insert more actions in the *actionsList*, the gas cost and the transaction delay will not be affected or increased, as Solidity reserves the maximum storage when creating and initializing the mapping (which is 2^{256} slots of 32 bytes).

Next, we discuss the differences between the two implementations of the smart contract-based digital twins. Fabric smart contracts include the whole TD of the "virtual entities" as opposed to the Ethereum smart contracts, which should not store the whole TD, since that would be too costly. Furthermore, Fabric presents better performance than Ethereum, as we have shown. Moreover, Fabric can serve many different use cases, since it introduces the concept of organizations. In a Fabric network, there can be two, or more, organizations. Each organization can have their digital twins, without the others knowing anything about them. In Fabric, we can even have smart contracts within the same organization that cannot be accessed by some peers of the same organization through the use of private channels, a mechanism introduced by Fabric. This is something that cannot be achieved with the use of the Ethereum blockchain, in which everything is public

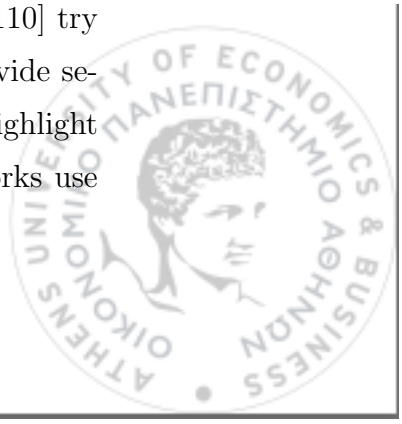


and everything recorded on the ledger can be accessed and inspected by anyone. Therefore, Fabric is a much better fit performance and cost-wise. However, as a permissioned blockchain, it is not appropriate for use cases, where openness, decentralization, and full transparency is desired.

5.2.5 Related work

In recent times, digital twins have witnessed increasing attention. There are several research efforts that try to integrate digital twins in IoT applications. Chevallier et al. [103] present an architecture for creating and managing digital twins for smart buildings. Liu et al. [104] propose a framework based on digital twins for an indoor safety management system. Moreover, Mohamadi et al. [105] propose a smart city digital twin paradigm. Similarly, White et al. [106] present a design for digital twins of smart cities. These works use digital twins mainly as monitoring tools and for performing simulations, which cannot be performed in the actual IoT systems. However, in our work, we are using the digital twin as an indirection mechanism to the actual IoT devices. We are using them to perform real actions on the IoT devices, instead of using them to extract information about the physical devices.

On the other hand, many efforts investigate the intersection of blockchains and digital twins, i.e., blockchain-based digital twins. Yaqoob et al. [107] present some potential use cases, architectures, and technologies that can enhance digital twins to be more effective in industrial problems. They use the blockchain as storage for digital twin's data. Similarly, Khan et al. [108] propose a framework, which uses the blockchain to securely store digital twin data. More recent efforts [109, 110, 111] are using the blockchain to address the problem of sharing the data generated by the digital twins. In particular, Putz et al. [109] introduce a Decentralized Application (DApp) that facilitates digital twin data sharing among multiple parties, without the need for trusted third parties. Dietz et al. [110] try to address the same problem by examining how DLTs can be used to provide secure sharing of data generated by the digital twins. All these efforts highlight the advantages of integrating DLTs and digital twins. However, these works use



the blockchain as a means for digital twin data sharing, while we are using the blockchain to implement the actual digital twins as smart contracts to isolate and secure the physical IoT devices.

5.2.6 Conclusions and future work

In this work, we presented a novel design of secure, available, and reliable smart contract-based digital twins of IoT devices that combines the WoT standards and blockchain technology. By using the WoT standards, we address successfully the problems of interoperability and fragmentation of the IoT. Moreover, by implementing digital twins as smart contracts, we achieve decentralization, high availability, robustness, flexibility, and auditability. To cover a wide range of IoT use cases and applications, we offer designs in two different blockchains, a public and a private blockchain, each one introducing different properties. Furthermore, to verify the feasibility of these digital twins, we designed and developed an IoT system prototype that offers secure sensing and actuation. Our solution secures the real IoT devices and the corresponding WoT gateways by allowing consumers to interact with them only through their smart contract-based digital twins. It also makes the users oblivious to the actual IoT device details and IoT device vendor-agnostic.

We presented the design and the implementation of smart contract-based digital twins and showed how they can be used in an IoT system. However, we have not tested and evaluated our proposed system in a real-world scenario. For this reason, a next step would be to implement and evaluate the solutions in a real life smart home testbed, or to a larger testbed, e.g., a smart-city or district, to experiment with it in a real setting and assess its performance, efficiency, usability, and scalability. Another direction would be to implement the smart contract-based digital twins in more blockchains, such as the Solana blockchain [112].

https://doi.org/10.26219/heal.aueb.9346

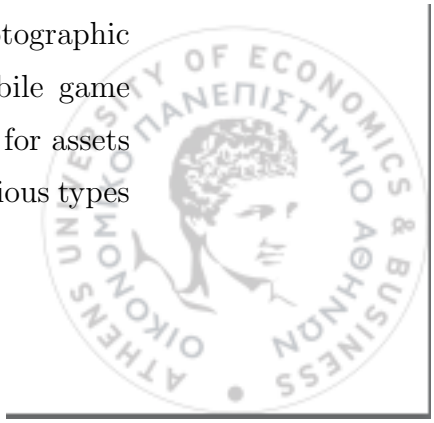
Chapter 6

Blockchain as enabling technology: Broader implications

In the previous chapters, we have thoroughly explored the benefits of integrating blockchain technology within the IoT, particularly emphasizing its robust capabilities in enhancing IoT access control and overall IoT architecture security and interoperability. Having proposed many solutions demonstrating the blockchain's benefits in these applications, we now discuss and demonstrate the versatility and transformative potential of the blockchain technology across a broader spectrum of domains. This chapter delves into its innovative applications beyond the realm of IoT, including the development of novel blockchain-based games and the facilitation of transparent marketplaces. By discussing these varied implementations, we aim to underscore blockchain's capacity as an enabler of modern technological solutions, illustrating its profound impact across diverse industries.

6.1 Blockchain-based games

DLTs have found application in many aspects of our lives, as they promise secure, trustworthy, and decentralized transactions with the use of cryptographic techniques. Lately, they have also caught the eyes of game and mobile game development industry. Offering solid proof of uniqueness and ownership for assets and rules transparency, they are fertile ground for the development of various types



of games, as we have shown in Section 5.1 or other types of games, such as trading games and mobile games.

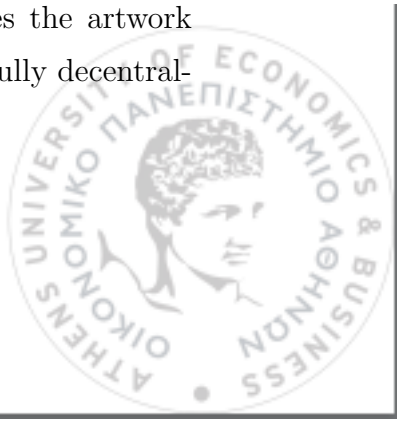
Gaming models have changed over the years, targeting to keep pace with the evolution of technology, trying to be attractive to users, and at the same time be more profitable for the gaming industry. The traditional Pay-to-Play model, where users pay upfront for the game, has lately been replaced by the Free-to-Play model. In the Free-to-Play model, users acquire the game at no cost, but they are incentivized to spend money for in-game assets and services. The advent of DLTs combined with the trend of gamers to earn an additional income from gaming, has given birth to the Play-to-Earn (P2E) model. In the P2E gaming model, not only do users play for free, but they can potentially earn cryptocurrencies. All in-game merchandise they earn playing rely on NFTs owned by them, and not by the company, which can be sold or exchanged for cryptocurrencies.

6.1.1 Trading games

In 2017, blockchain trading games made their appearance using NFTs and since then, they have been growing in popularity, as well as in market capitalization. From the pioneering Cryptokitties, to Axie Infinity,[1] the most recent pokemon-like game following the P2E model with over $1,100,000,000 total volume.[2] As these players communities grow, more game categories seem to adopt the concept and it is abundantly clear that NFT games are here to stay. Although these games are considered to be DApss, centralization of their media files (relating to or representing the game environment) has been a thorn in the community's side for years. Artwork and metadata are typically stored in the game company's servers, making the need for decentralized storage crucial. To overcome the latter, in [113], we presented a "fully decentralized" trading game in order to answer arising questions like "Who owns the artwork of the game?," "What happens if the game company loses interest in the game?," "Does the artwork have any value?," etc. In [114], we extended this work, realizing a fully decentral-

[1] https://axieinfinity.com/

[2] https://nomics.com/assets/axs2-axie-infinity/



https://doi.org/10.26219/heal.aueb.9346

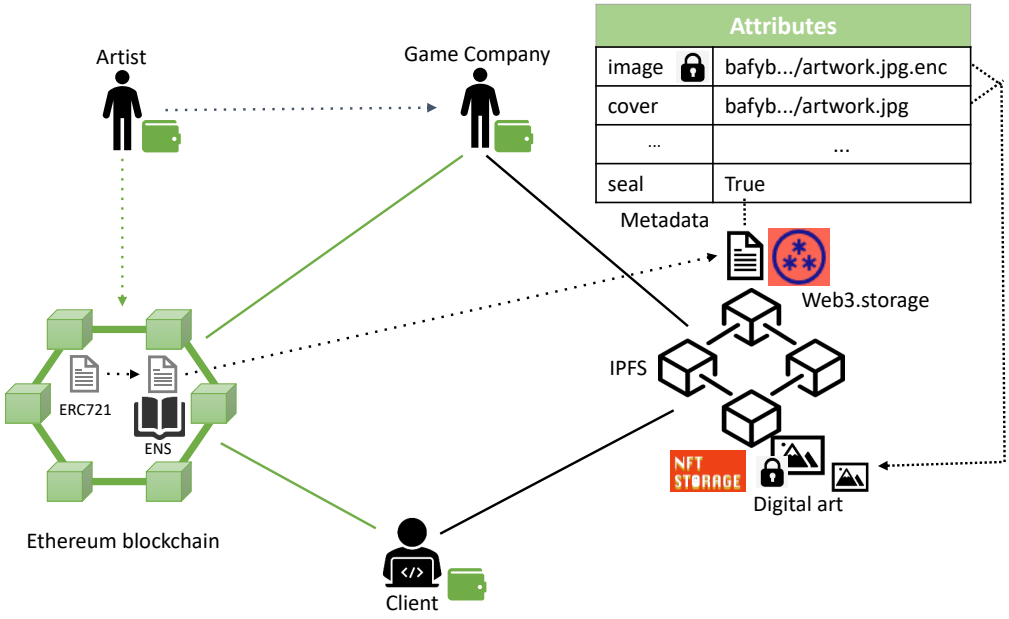
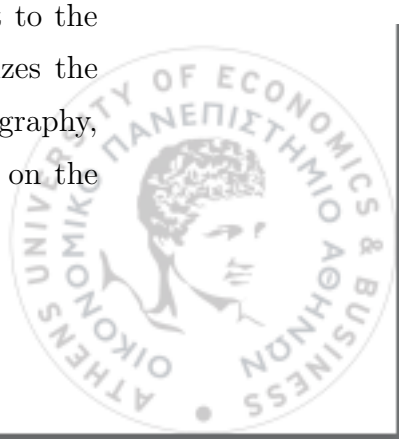


Figure 6.1: An overview of the architecture of a fully decentralized trading game.

ized trading game that follows the P2E model, through a proposed system, where the gaming company, artists, and users cooperate and interact over decentralized tools, in a tamper-proof and auditable manner. Blockchain, IPFS, and threshold cryptography are key mechanisms of the system.

In these works, the proposed system, illustrated in Figure 6.1, uses the Ethereum blockchain as the underlying infrastructure and two smart contracts, one that implements the ERC-721 token standard and one that implements the Ethereum Name Service (ENS). The ERC-721 smart contract is responsible for creating and managing the NFTs. For every NFT, there is a media file, e.g., character avatars, which is created by other entities, such as artists, and a metadata file, which shows information about the NFT. Anyone having an Ethereum wallet can acquire a NFT, through the smart contract, by paying the defined amount of money in ethers. From a high level perspective, the entities of our system interact with each other as follows. The artist creates the digital art and sends it to the gaming company. Then, the gaming company creates the NFTs, initializes the corresponding ENS entries, encrypts the digital art, using threshold cryptography, and uploads the encrypted digital art and the appropriate metadata file on the

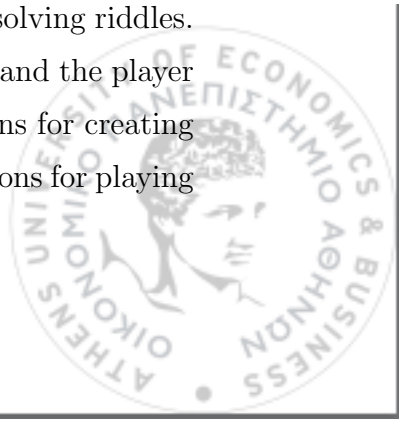


IPFS. The NFT's field $token_{URI}$ shows to the ENS entry, while the ENS entry is showing to the CID of the metadata file on IPFS. Finally, clients can acquire NFTs by paying the defined amount of money (in ethers) on the smart contract.

Our design leverages blockchain technology and other decentralized services, such as the IPFS, to introduce several novel features and intriguing properties that enhance the gaming experience. Key among these is the evolvability of in-game characters, represented as NFTs. With our design, there is no need for any change in the smart contract but only on the ENS entry, keeping the cost from gas consumption as low as possible. Additionally, our system realizes a digital equivalent of the "mint in sealed box" practice, commonly used by collectors of tangible, rare assets. By utilizing threshold cryptography alongside the blockchain's immutability, we ensure that the status of a NFT, whether encrypted or decrypted, is verifiably recorded. Furthermore, the use of smart contracts enables new and innovative business models, such as ensuring artists and other involved entities receive royalties on each resale of an in-game asset. Lastly, the fully decentralized nature of our system and the use of threshold cryptography guarantees its sustainability, even if the gaming company discontinuous support or ceases operations. In such cases, all game assets will remain accessible and recoverable (the owner will be able to decrypt them), as they are stored on the blockchain and IPFS, addressing a significant limitation in current gaming ecosystems.

6.1.2 Mobile games

Another example that showcases the benefits of blockchain technology in gaming is detailed in [115]. In this work, we integrate the IPFS with blockchain technology to develop a transparent and decentralized mobile game, named *R?ddle*. In this game, players compete in posing and solving mathematical riddles, designed by anyone, in order to earn rewards based on their performance. Our solution supports the P2E gaming model, as players are getting rewards for solving riddles. The game consists of two smart contracts, the game smart contract and the player smart contract. The player smart contract implements all the actions for creating and managing players, while the game smart contract includes functions for playing



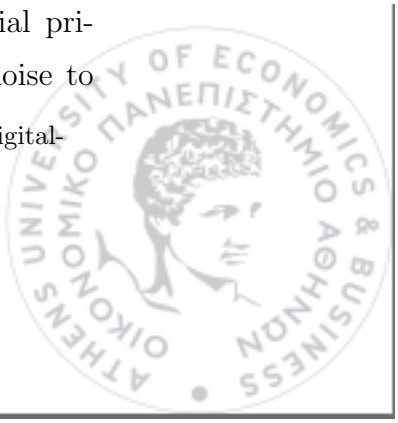
the game, e.g., play, reward, etc., and for managing the riddles and the tokenomics of the game, i.e., when someone adds a challenge, she should be get some tokens. The riddles along with the hints are stored on IPFS. The use of smart contract ensures that the game's rules are transparent and immutable, guaranteeing that they cannot be altered by anyone, including the gaming company itself, and they are always respected. Additionally, leveraging the Ethereum blockchain combined with IPFS for storing the riddles, ensures that the game remains permanently accessible and operational.

6.2 Blockchain-based data marketplaces: A privacy-preserving approach

We are living in a globalized, cyber connected society, where users have a plethora of choices. In this highly competitive environment, service providers try to offer as many personalized and consumer-tailored services as possible. In order to achieve their goal, they seek access to user profiling information. However, increased privacy concerns, as well as legislation, such as EU's General Data Protection Regulation (GDPR), have made the collection of such information a thorny challenge. Of course, this comes as no surprise, since such activities not only jeopardize users' privacy, but also, as we have recently witnessed, the collected information can be used for manipulating users' choices.[3] Hence, the research question "how can sensitive data be securely shared?" still remains open.

Traditionally, the collection of sensitive information has been protected using anonymization techniques. However, large-scale privacy breaches from supposedly anonymized datasets, such as those involving Netflix [116], have questioned the ability of those techniques to effectively protect user privacy. In this work [117], we propose a privacy-preserving solution that allows a "data consumer" to extract meaningful statistics from sensitive data protected using "local differential privacy" [118]. Local differential privacy enables "data providers" to add noise to

[3]https://georgetownlawtechreview.org/online-manipulation-hidden-influences-in-a-digital-world/GLTR-01-2020/



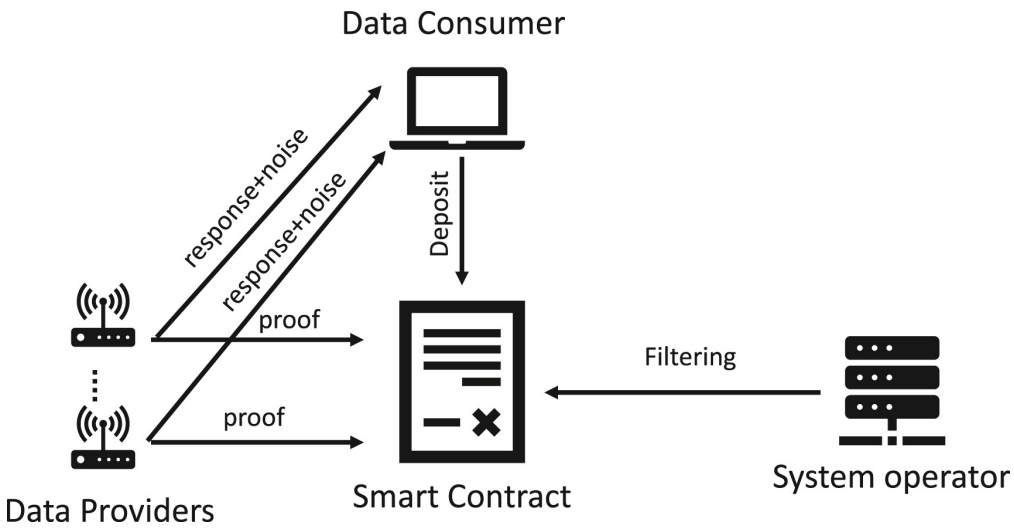
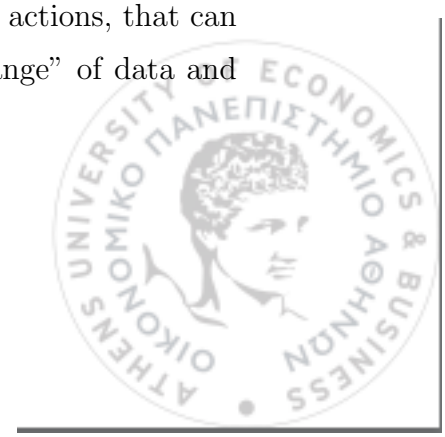


Figure 6.2: An overview of a blockchain-based marketplace for privacy-preserving statistics.

their (sensitive) data by themselves, and share them with untrusted 3^{rd} party data consumers without jeopardizing their privacy. Additionally, our solution considers an intermediate entity, referred to as the "system operator," that coordinates the whole process and applies filtering rules. With our approach, and as opposed to the state of the art, data providers are protected even against "curious" system operators. This is achieved by having the data providers send their (noisy) data directly to the data consumers.

Nevertheless, our approach leads inevitably to some tussles. For instance, a data consumer may be tempted to not pay the required fee. Similarly, data providers may indicate their interest to participate in a data collection process, receive the corresponding fee, but refuse to provide the actual data. Finally, a system operator may incorrectly filter out the responses of certain data providers. In order to resolve these tussles, we rely on the blockchain technology. In particular, as shown in Figure 6.2, we leverage Ethereum smart contracts to provide a privacy-preserving immutable log of operations, by storing hashes of the actions, that can be used for dispute resolution, as well as to provide "fair exchange" of data and service fees.



https://doi.org/10.26219/heal.aueb.9346

Chapter 7

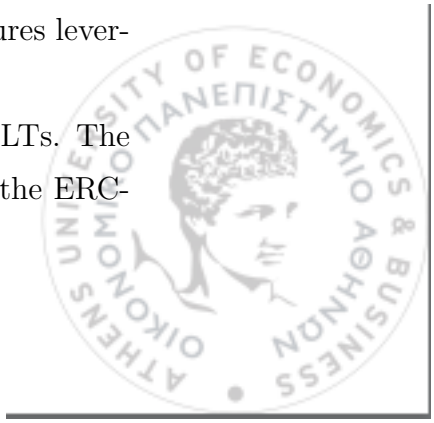
Conclusions and Future work

In this chapter, we first summarize the key findings and conclusions drawn from the studies reported in the preceding chapters. We then outline potential directions for future research, building upon the groundwork laid by this dissertation.

7.1 Conclusions

The IoT has emerged as a promising solution to enhance the quality of our lives, yet it significantly reshapes the existing landscape of the Internet and the Web. While the IoT offers numerous benefits, it also introduces new security threats and challenges. The first step into addressing these challenges is through access control. Traditional access control solutions, however, are deemed inappropriate for the IoT, due to it's peculiarities. We argue that DLTs can address these challenges, thanks to their inherently security properties. To this end, we revisited key blockchain systems and blockchain-based IoT access control solutions, we gathered security requirements for IoT access control, and we designed, implemented, and proposed novel blockchain-based access control solutions tailored to the IoT. Additionally, we designed, developed and proposed secure IoT architectures leveraging the blockchain technology.

Initially, we introduced IoT access control solutions backed by DLTs. The first presented solution is a token-based access control system utilizing the ERC-



20 blockchain-based tokens as ACTs. We showed that our solution has many intriguing security properties, notably secure, instantaneous, and effective revocation. Furthermore, our solution enables many novel features, such as panic mode, clients in probation period, and two-step access control. We demonstrated the feasibility of our solution through two real-life use cases, a large-scale IoT management solution and a multi-tenant smart city scenario. Subsequently, we proposed another type of ACT using the ERC-721 Ethereum token standard, demonstrating its applicability and feasibility by integrating it within the OAuth 2.0 authorization protocol. We showed that our design supports auditing, accountability, proof-of-possession, and added value token management. The last presented IoT access control solution is a consensus-based access control solution for multi-party collaborative environments. In particular, we proposed two approaches for consensus-based access control, one that utilizes smart contracts to act as PDPs and another that makes the actual consensus mechanism of Fabric to act as PDP. With our solution, we managed to decentralized the PDP and allow collaboration among many mutually non-trusting entities.

Additionally, we explored the broader integration of blockchain technology in the IoT. In particular, we proposed and developed blockchain-based IoT architectures for two use cases, the IoT mobile gaming and digital twins for IoT devices. Our evaluations indicate that the blockchain technology offers significant benefits, with the most notable of them being interoperability, while imposing minimal performance and cost overhead. Finally, we discussed the innovative applications of blockchains and smart contracts beyond the IoT. This includes blockchain-based gaming and transparent marketplaces, where we highlighted the key advantages and features enabled by blockchain technology, such as asset ownership and fair exchange.

In summary, this dissertation demonstrates that the IoT, while enhancing the quality of our life, introduces significant shifts in the Internet, accompanied by new security threats. Our research has shown that DLTs can effectively address these challenges through novel access control solutions, tailored to the IoT's unique needs. Furthermore, our exploration into the broader application of blockchain

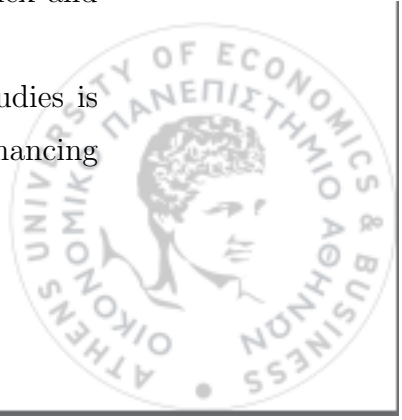
technology within IoT systems revealed substantial benefits, particularly in enhancing interoperability and security and enabling new business models. However, it is crucial to acknowledge that blockchain technology presents certain limitations, necessitating a careful evaluation of the trade-offs in its application. The dissertation's findings underscore blockchain's transformative potential in the IoT, paving the way for the development of more secure, interoperable, and efficient IoT architectures and systems.

7.2 Future work

In this dissertation, we demonstrated that while blockchain-based solutions offer robust security features, they can also be very costly and exhibit poor performance, which may be unbearable for real-life IoT applications, especially the public blockchains, such as Ethereum. Therefore, a promising direction for future research involves exploring the viability of the presented blockchain-based solutions, on newer blockchains, such as Solana, which offer lower transaction costs and higher throughput. Investigating such blockchains could enhance the performance and efficiency of the presented solutions. This exploration will not only aid in understanding how different blockchain characteristics impact the performance and effectiveness of these solutions, bu also broaden the range of blockchain technologies that can be effectively integrated into IoT environments.

An interesting future direction in IoT access control research would be to integrate context-awareness into blockchain-based access control solutions. Context-aware access control systems adjust permissions based on the context of the access request, such as the users' location, time of access, etc. Efficiently combining context-awareness with blockchain technology could lead to more adaptive and complete access control solutions. Exploring these possibilities could significantly enhance the functionality and security of access control systems in complex and dynamic environments like the IoT.

Finally, an aspect of access control systems not covered in our studies is the attenuation mechanisms. Therefore, future research could focus on enhancing



https://doi.org/10.26219/heal.aueb.9346

these mechanisms through the use of blockchain technology and smart contracts. Smart contracts are particularity well-suited for enforcing attenuation rules in a decentralized and transparent manner, ensuring that access rights are dynamically adjusted and restricted based on predefined rules. This approach could significantly improve the security of IoT systems, by strictly adhering to the principle of least privilege across decentralized environments.

https://doi.org/10.26219/heal.aueb.9346

Appendix A

Acronyms

ABAC Attributed-Based Access Control
ACL Access Control List
ACT Access Control Token
API Application Programming Interface
AR Augmented Reality
ARPU Average Revenue Per User
AS Authorization Server
BFT Byzantine Fault Tolerance
BLE Bluetooth Low Energy
CA Certificate Authority
CapBAC Capability-Based Access Control
CFT Crash Fault Tolerant
CoAP Constrained Application Protocol
CoRE Constrained RESTful Environment
CRL Certificate Revocation List
DAC Discretionary Access Control
DAG Directed Acyclic Graph
DAO Decentralized Autonomous Organization
DApp Decentralized Applications
DAU Daily Active Users

https://doi.org/10.26219/heal.aueb.9346

DID Decentralized Identifier

DLT Distributed Ledger Technology

DDoS Distributed Denial of Service

DoD Department of Defense

DoS Denial of Service

EIP Ethereum Improvement Proposal

ENS Ethereum Name Service

ERC Ethereum Request for Comments

ESO Environmental Situation Oracle

EVM Ethereum Virtual Machine

GDPR General Data Protection Regulation

HTTP HyperText Transfer Protocol

ILG Interledger Gateway

IoT Internet of Things

IPFS InterPlanetary File System

JSON JavaScript Object Notation

JSON-LD JavaScript Object Notation - Linked Data

JWT JSON Web Tokens

KPI Key Performance Indicator

MAC Mandatory Access Control

MAU Monthly Active Users

MSP Membership Service Provider

M2M Machine-to-Machine

N/A Not Applicable

NFT Non-Fungible Token

OrBAC Organization-Based Access Control

OSN Online Social Network

PAP Policy Administration Point

pBFT practical Byzantine Fault Tolerance

PDP Policy Decision Point

PEP Policy Enforcement Point

https://doi.org/10.26219/heal.aueb.9346

PIP Policy Information Point

PKI Public Key Infrastructure

PoA Proof of Authority

PoI Point of Interest

PoS Proof of Stake

PoW Proof of Work

P2E Play-to-Earn

P2P Peer to Peer

RBAC Role-based Access Control

RDE Reverse-Discoverable Encryption

ReBAC Relationship-Based Access Control

REST Representational State Trasnfer

RPC Remote Procedure Call

SFE Secure Function Evaluation

SOFIE Secure Open Federation for Internet Everywhere

TD Thing Description

TCSEC Trusted Computer System Evaluation Criteria

TPS Transactions Per Second

UCON Usage Control

URI Uniform Resource Identifier

UAV Unmanned Aerial Vehicle

UGC User Generated Content

URL Uniform Resource Locator

VC Verifiable Credential

WIBE Wildcard Identity-based Encryption

WoT Web of Things

XACML eXtensible Access Control Markup Language

Bibliography

[1] P. Asghari, A. M. Rahmani, and H. H. S. Javadi, "Internet of Things applications: A systematic review," *Computer Networks*, vol. 148, pp. 241–261, 2019.

[2] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Computer Networks*, vol. 76, pp. 146–164, 2015.

[3] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain Technology Overview," NIST, NIST Interagency/Internal Report (NISTIR) 8202, 2018.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

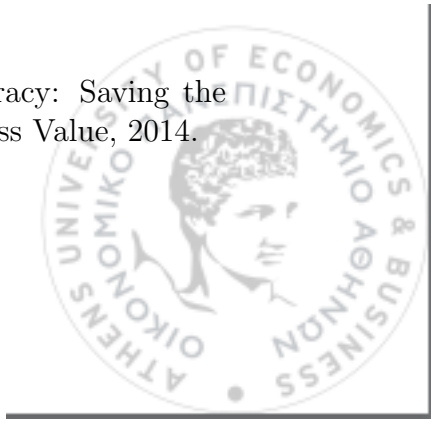
[5] S. Voulgaris, N. Fotiou, V. A. Siris, G. C. Polyzos, A. Tomaras, and S. Karachontzitis, "Hierarchical Blockchain Topologies for Quality Control in Food Supply Chains," in *2020 European Conference on Networks and Communications (EuCNC)*, 2020.

[6] M. Andoni, V. Robu, D. Flynn, S. Abram, D. Geach, D. Jenkins, P. McCallum, and A. Peacock, "Blockchain technology in the energy sector: A systematic review of challenges and opportunities," *Renewable and Sustainable Energy Reviews*, 2019.

[7] M. S. Devi, R. Suguna, A. S. Joshi, and R. A. Bagate, "Design of IoT Blockchain Based Smart Agriculture for Enlightening Safety and Security," in *Emerging Technologies in Computer Engineering: Microservices in Big Data Analytics*. Springer Singapore, 2019.

[8] N. Fotiou and G. C. Polyzos, "Smart Contracts for the Internet of Things: Opportunities and Challenges," in *2018 European Conference on Networks and Communications (EuCNC)*, 2018, pp. 256–260.

[9] J. Cohn, P. Finn, S. Nair, and P. Sanjai, "Device democracy: Saving the future of the Internet of Things," IBM Institute for Business Value, 2014.



[10] R. Sandhu and P. Samarati, "Access control: principle and practice," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 40–48, 1994.

[11] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero Trust Architecture," NIST, Tech. Rep., 2020.

[12] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, 2014.

[13] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: Association for Computing Machinery, 2018.

[14] V. Vogelsteller and B. Vitalik, "ERC-20 Token Standard," Tech. Rep., 2015. [Online]. Available: https://eips.ethereum.org/EIPS/eip-20

[15] W. Entriken, D. Shirley, J. Evans, and N. Sachs, "ERC-721 Non-Fungible Token Standard," Tech. Rep., 2018. [Online]. Available: https://eips.ethereum.org/EIPS/eip-721

[16] D. Hardt (ed.), "The OAuth 2.0 Authorization Framework," IETF, RFC 6749, 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6749

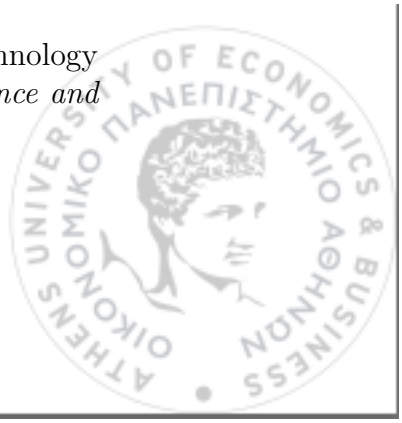
[17] W3C. (2017) Web of Things. [Online]. Available: https://www.w3.org/WoT/

[18] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. USA: USENIX Association, 1999, p. 173–186.

[19] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, p. 382–401, 1982.

[20] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing Safety of Smart Contracts," in *Procceedings of Network and Distributed System Security Symposium*, 2018.

[21] P. Aithal, P. Saavedra, S. Aithal, and S. Ghoash, "Blockchain technology and its types-a short review," *International Journal of Applied Science and Engineering*, vol. 9, pp. 189–200, 2021.



[22] V. Buterin and V. Griffith, "Casper the friendly finality gadget," 2019. [Online]. Available: https://arxiv.org/abs/1710.09437

[23] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," 2014. [Online]. Available: https://arxiv.org/pdf/1407.3561

[24] E. Androulaki, A. De Caro, M. Neugschwandtner, and A. Sorniotti, "Endorsement in Hyperledger Fabric," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019.

[25] P. Samarati and S. C. de Vimercati, "Access Control: Policies, Models, and Mechanisms," in *Foundations of Security Analysis and Design*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 137–196.

[26] J. K. Biba, "Integrity Considerations for Secure Computer Systems," Bedford, MA, 1977.

[27] D. E. Bell and L. LaPadula, "Secure Computer Systems: Mathematical Foundations," 1973.

[28] D. 5200.28-STD, *Trusted Computer System Evaluation Criteria*, Dod Computer Security Center, December 1985.

[29] D. D. Downs, J. R. Rub, K. C. Kung, and C. S. Jordan, "Issues in Discretionary Access Control," in *1985 IEEE Symposium on Security and Privacy*, 1985, pp. 208–208.

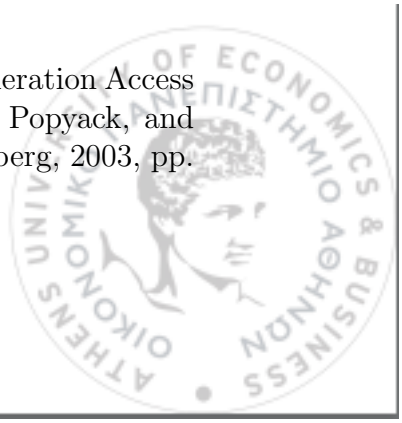
[30] Y. Jiang, C. Lin, H. Yin, and Z. Tan, "Security analysis of mandatory access control model," in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, vol. 6, 2004, pp. 5013–5018 vol.6.

[31] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[32] E. Yuan and J. Tong, "Attributed based access control (ABAC) for Web services," in *IEEE International Conference on Web Services (ICWS'05)*, 2005, p. 569.

[33] A. Kalam, R. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin, "Organization based access control," in *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, 2003, pp. 120–131.

[34] R. Sandhu and J. Park, "Usage Control: A Vision for Next Generation Access Control," in *Computer Network Security*, V. Gorodetsky, L. Popyack, and V. Skormin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 17–31.



[35] N. Sivaselvan, W. Asif, B. K. Vivekananda, and M. Rajarajan, "Authentication and Capability-based Access Control: An Integrated Approach for IoT Environment," in *2020 12th International Conference on Communication Software and Networks (ICCSN)*, 2020, pp. 110–117.

[36] C. E. Gates, "Access Control Requirements for Web 2.0 Security and Privacy," in *IEEE Web2.0 Privacy and Security Workshop (W2SP'07)*, 2007.

[37] "eXtensible Access Control Markup Language (XACML) Version 1.0," OASIS, Standard, 2003. [Online]. Available: https://www.oasis-open.org/committees/xacml/repository/oasis-xacml-1.0.pdf

[38] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective," *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 349–359, 2014.

[39] S. Ravidas, A. Lekidis, F. Paci, and N. Zannone, "Access control in Internet-of-Things: A survey," *Journal of Network and Computer Applications*, vol. 144, pp. 79–101, 2019.

[40] "eXtensible Access Control Markup Language (XACML) Version 3.0," OASIS, Standard, 2013. [Online]. Available: https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

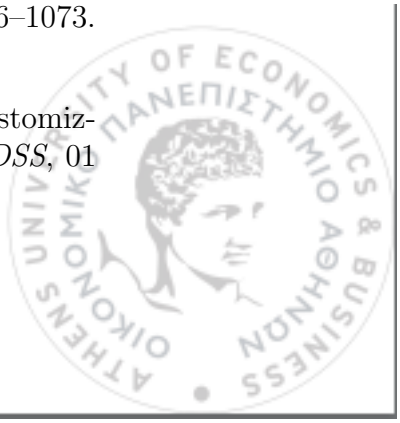
[41] F. Chen, Z. Xiao, L. Cui, Q. Lin, J. Li, and S. Yu, "Blockchain for Internet of Things applications: A review and open issues," *Journal of Network and Computer Applications*, vol. 172, p. 102839, 2020.

[42] 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*, vol. 7, no. 6, pp. 4682–4696, 2020.

[43] M. P. Andersen, S. Kumar, M. AbdelBaky, G. Fierro, J. Kolb, H.-S. Kim, D. E. Culler, and R. A. Popa, "WAVE: A Decentralized Authorization Framework with Transitive Delegation," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1375–1392.

[44] R. Schuster, V. Shmatikov, and E. Tromer, "Situational Access Control in the Internet of Things," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1056–1073. [Online]. Available: https://doi.org/10.1145/3243734.3243817

[45] H. Chi, Q. Zeng, X. Du, and L. Luo, "PFirewall: Semantics-Aware Customizable Data Flow Control for Smart Home Privacy Protection," in *NDSS*, 01 2021.



[46] Y. Liu, M. Qiu, J. Liu, and M. Liu, "Blockchain-Based Access Control Approaches," in *2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2021.

[47] A. Ouaddah, A. A. E. Kalam, and A. A. Ouahman, "FairAccess: a new Blockchain-based access control framework for the Internet of Things," *Secur. Commun. Networks*, vol. 9, pp. 5943–5964, 2016.

[48] M. T. Hammi, B. Hammi, P. Bellot, and A. Serhrouchni, "Bubbles of Trust: A decentralized blockchain-based authentication system for IoT," *Computers & Security*, vol. 78, pp. 126–142, 2018.

[49] O. Novo, "Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1184–1195, April 2018.

[50] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart Contract-Based Access Control for the Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2019.

[51] H. Liu, D. Han, and D. Li, "Fabric-iot: A Blockchain-Based Access Control System in IoT," *IEEE Access*, vol. 8, pp. 18 207–18 218, 2020.

[52] N. Fotiou, I. Pittaras, V. A. Siris, S. Voulgaris, and G. C. Polyzos, "Secure IoT Access at Scale Using Blockchains and Smart Contracts," in *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2019, pp. 1–6.

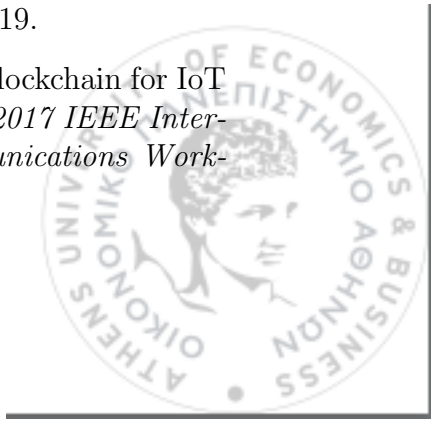
[53] I. Pittaras and G. C. Polyzos, "Multi-tenant, Decentralized Access Control for the Internet of Things," in *2023 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)*, 2023, pp. 28–34.

[54] A. Rahman and E. Dijk, "Group communication for the constrained application protocol (CoAP)," IETF, RFC 7390, 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7390

[55] C. Amsüss, Z. Shelby, M. Koster, C. Bormann, and P. V. der Stok, "Constrained RESTful Environments (CoRE) Resource Directory," IETF, RFC 9176, 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9176

[56] V. A. Siris, P. Nikander, S. Voulgaris, N. Fotiou, D. Lagutin, and G. C. Polyzos, "Interledger Approaches," *IEEE Access*, vol. 7, 2019.

[57] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for IoT security and privacy: The case study of a smart home," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, March 2017, pp. 618–623.



[58] Y. Hanada, L. Hsiao, and P. Levis, "Smart Contracts for Machine-to-Machine Communication: Possibilities and Limitations," in *2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS)*, Nov 2018, pp. 130–136.

[59] N. Fotiou, I. Pittaras, V. A. Siris, S. Voulgaris, and G. C. Polyzos, "OAuth 2.0 Authorization using Blockhain-based Tokens," in *3rd NDSS Workshop on Decentralized IoT Systems and Security (DISS)*, 2020.

[60] M. B. Jones and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage," IETF, RFC 6750, 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6750

[61] M. B. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF, RFC 7519, 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7519

[62] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," Network Working Group, RFC 4648, 2006. [Online]. Available: https://www.rfc-editor.org/info/rfc4648

[63] J. Richer, M. B. Jones, J. Bradley, M. Machulak, and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol," IETF, RFC 7591, 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7591

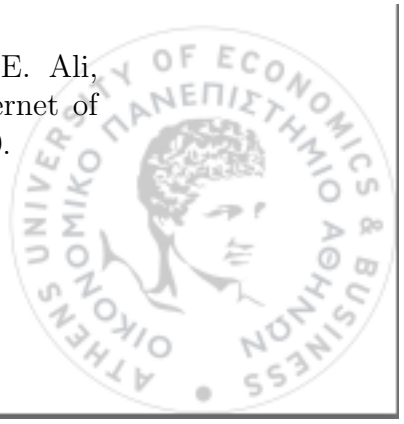
[64] T. Lodderstedt, S. Dronia, and M. Scurtescu, "OAuth 2.0 Token Revocation," IETF, RFC 7009, 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc7009

[65] M. B. Jones, J. Bradley, and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)," IETF, RFC 7800, 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7800

[66] S. Dziembowski, L. Eckey, and S. Faust, "FairSwap: How To Fairly Exchange Digital Goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 967–984.

[67] M. Sporny, D. Longley, D. Chadwick, and O. Steele, "Verifiable Credentials Data Model v2.0," W3C, W3C Recommendation, 2025. [Online]. Available: https://www.w3.org/TR/vc-data-model-2.0/

[68] G. Ali, N. Ahmad, Y. Cao, M. Asif, H. Cruickshank, and Q. E. Ali, "Blockchain based permission delegation and access control in Internet of Things (BACI)," *Computers & Security*, vol. 86, pp. 318 – 334, 2019.



[69] D. Di Francesco Maesa, P. Mori, and L. Ricci, "A blockchain based approach for the definition of auditable Access Control systems," *Computers & Security*, vol. 84, pp. 93–119, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818309398

[70] V. A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, and G. C. Polyzos, "OAuth 2.0 meets Blockchain for Authorization in Constrained IoT Environments," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019, pp. 364–367.

[71] T. Hardjono, "Decentralized Service Architecture for OAuth2.0," IETF, Internet-Draft draft-hardjono-oauth-decentralized-02, 2018. [Online]. Available: https://datatracker.ietf.org/doc/draft-hardjono-oauth-decentralized/02/

[72] F. Paci, A. Squicciarini, and N. Zannone, "Survey on Access Control for Community-Centered Collaborative Systems," *ACM Comput. Surv.*, vol. 51, no. 1, jan 2018.

[73] A. C. Squicciarini, S. M. Rajtmajer, and N. Zannone, "Multi-Party Access Control: Requirements, State of the Art and Open Challenges," in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, ser. SACMAT '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 49.

[74] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong, "Access Control in Collaborative Systems," *ACM Comput. Surv.*, vol. 37, no. 1, mar 2005.

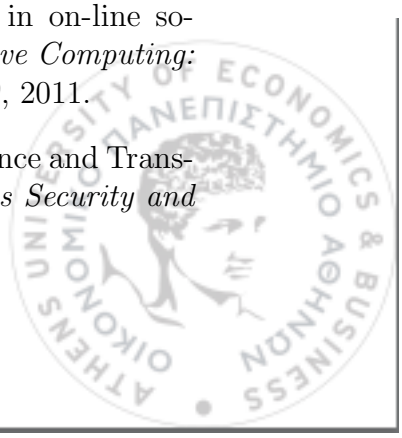
[75] H. Shen and P. Dewan, "Access Control for Collaborative Environments," in *Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work*, ser. CSCW '92. New York, NY, USA: Association for Computing Machinery, 1992.

[76] B. W. Lampson, "Protection," *SIGOPS Oper. Syst. Rev.*, vol. 8, no. 1, p. 18–24, Jan. 1974.

[77] S. Damen, J. den Hartog, and N. Zannone, "CollAC: Collaborative access control," in *2014 International Conference on Collaboration Technologies and Systems (CTS)*, 2014.

[78] B. Carminati and E. Ferrari, "Collaborative access control in on-line social networks," in *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, 2011.

[79] R. Mahmudlu, J. den Hartog, and N. Zannone, "Data Governance and Transparency for Collaborative Systems," in *Data and Applications Security and*



Privacy XXX, S. Ranise and V. Swarup, Eds. Cham: Springer International Publishing, 2016, pp. 199–216.

[80] M. Sheikhalishahi, G. Tillem, Z. Erkin, and N. Zannone, "Privacy-Preserving Multi-Party Access Control," in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, ser. WPES'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–13.

[81] D. R. George, S. Sciancalepore, and N. Zannone, "Privacy-Preserving Multi-Party Access Control for Third-Party UAV Services," in *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 19–30.

[82] A. Gouglidis and I. Mavridis, "domRBAC: An access control model for modern collaborative systems," *Computers & Security*, vol. 31, no. 4, pp. 540–556, 2012.

[83] S. Rouhani and R. Deters, "Blockchain Based Access Control Systems: State of the Art and Challenges," in *IEEE/WIC/ACM International Conference on Web Intelligence*, ser. WI '19. New York, NY, USA: Association for Computing Machinery, 2019.

[84] Y. Zhang, A. Memariani, and N. Bidikar, "A Review on Blockchain-based Access Control Models in IoT Applications," in *2020 IEEE 16th International Conference on Control & Automation (ICCA)*, 2020.

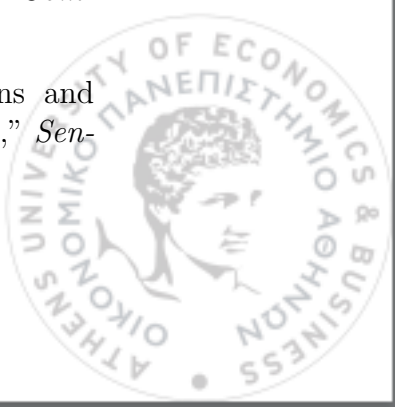
[85] I. Riabi, H. K. B. Ayed, and L. A. Saidane, "A survey on Blockchain based access control for Internet of Things," in *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, 2019.

[86] G. Gan, E. Chen, Z. Zhou, and Y. Zhu, "Token-Based Access Control," *IEEE Access*, vol. 8, pp. 54 189–54 199, 2020.

[87] Ahmad Raza Khan, "Zero trust-based blockchain based iot security with consensus and access control framework," *Journal of Intelligent Systems and Internet of Things*, vol. 12, pp. 110–128, 01 2024.

[88] V. A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, and G. C. Polyzos, "Interledger smart contracts for decentralized authorization to constrained things," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 336–341.

[89] I. Pittaras, N. Fotiou, V. A. Siris, and G. C. Polyzos, "Beacons and Blockchains in the Mobile Gaming Ecosystem: A Feasibility Analysis," *Sensors*, vol. 21, no. 3, 2021.



[90] A. Manzoor, M. Samarin, D. Mason, and M. Ylianttila, "Scavenger Hunt: Utilization of Blockchain and IoT for a Location-Based Game," *IEEE Access*, vol. 8, 2020.

[91] T. Min, H. Wang, Y. Guo, and W. Cai, "Blockchain Games: A Survey," in *2019 IEEE Conference on Games (CoG)*, 2019.

[92] T. Min and W. Cai, "A Security Case Study for Blockchain Games," in *2019 IEEE Games, Entertainment, Media Conference (GEM)*, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8811555

[93] W. Cai and X. Wu, "Demo Abstract: An Interoperable Avatar Framework Across Multiple Games and Blockchains," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019.

[94] S. Kalra, R. Sanghi, and M. Dhawan, "Blockchain-Based Real-Time Cheat Prevention and Robustness for Multi-Player Online Games," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT 2018. New York, NY, USA: Association for Computing Machinery, 2018.

[95] B. R. Barricelli, E. Casiraghi, and D. Fogli, "A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications," *IEEE Access*, pp. 167 653–167 671, 2019.

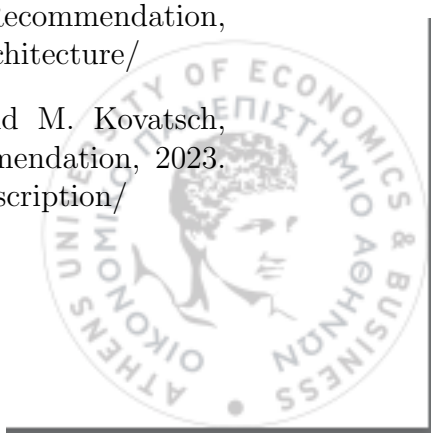
[96] I. Pittaras, N. Fotiou, C. Karapapas, V. A. Siris, and G. C. Polyzos, "Secure smart contract-based digital twins for the Internet of Things," *Blockchain: Research and Applications*, vol. 5, no. 1, p. 100168, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S209672092300043X

[97] ——, "Secure, Mass Web of Things Actuation Using Smart Contracts-Based Digital Twins," in *2022 IEEE Symposium on Computers and Communications (ISCC)*, 2022, pp. 1–6.

[98] I. Pittaras and G. C. Polyzos, "Secure and Efficient Web of Things Digital Twins using Permissioned Blockchains," in *2022 7th International Conference on Smart and Sustainable Technologies (SpliTech)*, 2022, pp. 1–5.

[99] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, and K. Kajimoto, "Web of Things Architecture," W3C, W3C Recommendation, 2020. [Online]. Available: https://www.w3.org/TR/wot-architecture/

[100] S. Kaebish, T. kamiya, M. McCool, V. Charpenay, and M. Kovatsch, "Web of Things Thing Description," W3C, W3C Recommendation, 2023. [Online]. Available: https://www.w3.org/TR/wot-thing-description/



[101] Christopher Cordi, "Hyperledger Fabric Security Threats: What to Look For," 2021, (Accessed Dec. 22). [Online]. Available: https://www.hyperledger.org/blog/2021/11/18/hyperledger-fabric-security-threats-what-to-look-for

[102] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "FastFabric: Scaling hyperledger fabric to 20000 transactions per second," *International Journal of Network Management*, 2020.

[103] Z. Chevallier, B. Finance, and B. C. Boulakia, "A reference architecture for smart building digital twin," in *2020 International Workshop on Semantic Digital Twins, SeDiT 2020*, vol. 2615, France, 2020.

[104] Z. Liu, A. Zhang, and W. Wang, "A Framework for an Indoor Safety Management System Based on Digital Twin," *Sensors*, 2020.

[105] N. Mohammadi and J. E. Taylor, "Smart city digital twins," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017.

[106] G. White, A. Zink, L. Codecá, and S. Clarke, "A digital twin smart city for citizen feedback," *Cities*, 2021.

[107] I. Yaqoob, K. Salah, M. Uddin, R. Jayaraman, M. Omar, M. Imran, "Blockchain for Digital Twins: Recent Advances and Future Research Challenges," *IEEE Network*, 2020.

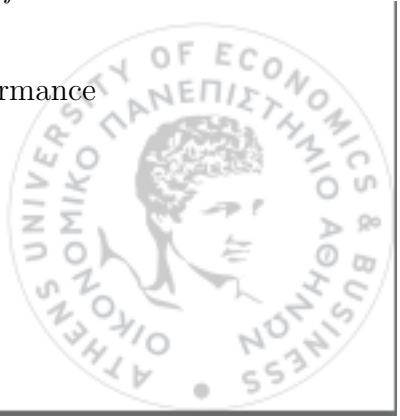
[108] A. Khan, F. Shahid, C. Maple, A. Ahmad, and G. Jeon, "Toward Smart Manufacturing Using Spiral Digital Twin Framework and Twinchain," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 2, pp. 1359–1366, 2022.

[109] B. Putz, M. Dietz, P. Empl, and G. Pernul, "EtherTwin: Blockchain-based Secure Digital Twin Information Management," *Information Processing & Management*, p. 102425, 2021.

[110] M. Dietz, B. Putz, and G. Pernul, "A Distributed Ledger Approach to Digital Twin Secure Data Sharing," in *33th IFIP Annual Conference on Data and Applications Security and Privacy (DBSec)*, vol. LNCS-11559, 2019, pp. 281–300.

[111] W. Shen, T. Hu, C. Zhang, and S. Ma, "Secure sharing of big digital twin data for smart manufacturing based on blockchain," *Journal of Manufacturing Systems*, 2021.

[112] A. Yakovenko, "Solana: A new architecture for a high performance blockchain," *Whitepaper*, 2018.



[113] C. Karapapas, I. Pittaras, and G. C. Polyzos, "Fully Decentralized Trading Games with Evolvable Characters using NFTs and IPFS," in *2021 IFIP Networking Conference (IFIP Networking)*, 2021, pp. 1–2.

[114] C. Karapapas, G. Syros, I. Pittaras, and G. C. Polyzos, "Decentralized NFT-based Evolvable Games," in *2022 4th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, 2022, pp. 67–74.

[115] A. M. Papathanasiou, C. D. N. Kyriakidou, I. Pittaras, and G. C. Polyzos, "R?ddle: A Fully Decentralized Mobile Game for Fun and Profit," in *Blockchain and Applications, 4th International Congress*, 2023.

[116] A. Narayanan and V. Shmatikov, "Robust De-anonymization of Large Sparse Datasets," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, 2008, pp. 111–125.

[117] N. Fotiou, I. Pittaras, V. A. Siris, G. C. Polyzos, and P. Anton, "A privacy-preserving statistics marketplace using local differential privacy and blockchain: An application to smart-grid measurements sharing," *Blockchain: Research and Applications*, vol. 2, no. 1, p. 100022, 2021.

[118] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating Noise to Sensitivity in Private Data Analysis," in *Theory of Cryptography*, S. Halevi and T. Rabin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 265–284.