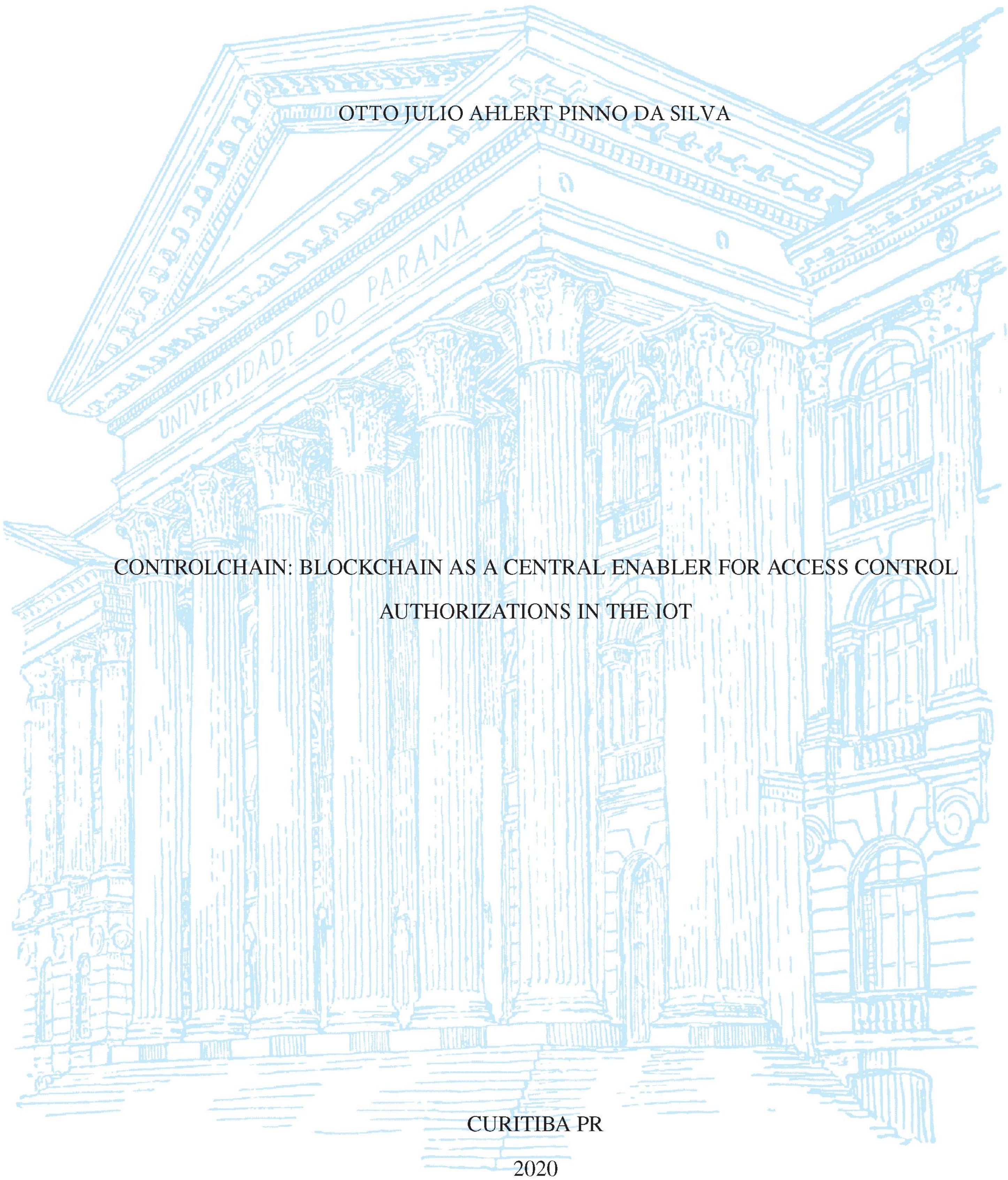
UNIVERSIDADE FEDERAL DO PARANÁ



OTTO JULIO AHLERT PINNO DA SILVA

CONTROLCHAIN: BLOCKCHAIN AS A CENTRAL ENABLER FOR ACCESS CONTROL AUTHORIZATIONS IN THE IOT

CURITIBA PR

2020

OTTO JULIO AHLERT PINNO DA SILVA

CONTROLCHAIN: BLOCKCHAIN AS A CENTRAL ENABLER FOR ACCESS CONTROL AUTHORIZATIONS IN THE IOT

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Luis Carlos Erpen De Bona.

Coorientador: André Ricardo Abed Grégio.

CURITIBA PR

2020

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

P656c Pinno, Otto Julio Ahlert
Controlchain: Blockchain as a central enabler for access control authorizations in the IOT [recurso eletrônico] /Otto Julio Ahlert Pinno. – Curitiba, 2020.

Tese - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós - Graduação em Informática, 2020.

Orientador: Luis Carlos Erpen De Bona. Coorientador: André Ricardo Abed Grégio.

1. Internet das coisas. 2. Internet - Controle de acesso. 3. Redes locais de computadores. 4. Blockchains (Base de dados). I. Universidade Federal do Paraná. II. De Bona, Luis Carlos Erpen. III. Grégio, André Ricardo Abed. VI. Título.

CDD: 004.678

Bibliotecária: Vanusa Maciel CRB- 9/1928

MINISTÉRIO DA EDUCAÇÃO
SETOR DE CIENCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO INFORMÁTICA -
40001016034P5

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **OTTO JULIO AHLERT PINNO DA SILVA** intitulada: **CONTROLCHAIN: BLOCKCHAIN AS A CENTRAL ENABLER FOR ACCESS CONTROL AUTHORIZATIONS IN THE IOT**, sob orientação do Prof. Dr. LUIS CARLOS ERPEN DE BONA, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.
A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 07 de Janeiro de 2020.

[illegible]
LUIS CARLOS ERPEN DE BONA
Presidente da Banca Examinadora

[illegible]
ADRIANO CANSIAN
Avaliador Externo (UNIVERSIDADE ESTADUAL PAULISTA)

[illegible]
ALTAIR OLIVO SANTIN
Avaliador Externo (PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ)

[illegible]
LUIZ CARLOS PESSOA ALBINI
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

[illegible]
ANDRÉ RICARDO ABED GRÉGIO
Coorientador (UNIVERSIDADE FEDERAL DO PARANÁ)

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

RESUMO

A IoT está alterando a forma como interagimos com o mundo. Logo, quase todas as nossas tarefas diárias serão realizadas através sistemas inteligentes embarcados em dispositivos espalhados por toda parte. A missão deles é tornar nossas cidades, sistemas de transporte, construções, residências e corpos em ambientes inteligentes. Estes ambientes trarão mais conforto, melhorarão nossos desempenhos, aumentarão nossos lucros e removerão tarefas consumidoras de tempo. No entanto, junto com esses ótimos benefícios, a IoT também é uma grande fonte de preocupações, principalmente porque uma boa parte dos seus dispositivos manusearão informações privadas e confidenciais. Casos recentes de invasões de IoT bem sucedidos (com possibilidade de comprometimento de privacidade e confidencialidade) somente pioram este cenário e mostram a nós que os sistemas de controle de acesso adotados atualmente precisam ser substituídos por sistemas mais eficientes e seguros. Essas falhas de controle de acesso dificultam a adoção ampla da IoT, principalmente em ambientes que lidam com informações pessoais ou outras informações confidenciais. Infelizmente, características da IoT, como diversidade de dispositivos, quantidade e dispersão geográfica, inserem um alto grau de complexidade em projetos de controle de acesso. Apesar dos diversos estudos científicos na área, existem lacunas que precisam ser preenchidas. Alguns trabalhos usam soluções centralizadas que prejudicam a escalabilidade e a disponibilidade da IoT. Outros trabalhos proveem arquiteturas decentralizadas, no entanto, suas soluções não permitem à IoT atingir seu potencial total. Para sobrepor essas e outras barreiras, neste trabalho, nós fizemos um levantamento de requisitos para o controle de acesso na IoT e, então, projetamos o ControlChain, uma arquitetura de autorização de controle de acesso que é fortemente baseada na tecnologia Blockchain. Ela agrupa todos os requerimentos de tendencia fundamentais levantados em uma solução. Nós demonstramos a viabilidade do ControlChain através do E-ControlChain, uma prova de conceito desenvolvida para executar sobre a rede Ethereum. Nós também demonstramos a sua viabilidade através de uma análise de custo e de desempenho utilizando um Raspberry Pi como um dispositivo IoT. Finalmente, nós avaliamos E-ControlChain sob uma ampla variedade de ataques e discutimos como eles podem ameaçar ele e como mitigá-los.

Palavras-chave: Internet das coisas, Controle de Acesso, Autorização, Blockchain

ABSTRACT

The IoT is changing the way we interact with the world. Very soon, almost all of our daily tasks will be made through self intelligent systems embedded in devices scattered all around us. Their mission is to turn our cities, transportation systems, buildings, homes and bodies in smart environments. These environments will bring us more comfort, improve our performance, increase our profits, and take away time-consuming tasks. However, besides its great benefits, the IoT is also a big source of concerns, mainly because a good part of its devices will handle private and confidential information. Recently cases of successful IoT invasions (with possible privacy and confidentiality compromising) only worse this scenario and show us that the today's adopted access control systems need to be replaced by more efficiently and secure ones. These access control faults hinders the broadly adoption of the IoT, mostly in environments that deal with personal or other confidential information. Unfortunately, Features of the IoT such as device diversity, quantity and geographic dispersion place high degree of complexity in access control projects. Despite the many scientific studies in the area, there are gaps that need to be filled. Some of the works use centralized solutions that harm the scalability and availability of the IoT. Other works provide decentralized architectures, however their solution does not allow the IoT to achieve all of its potential. To overcome these and other barriers, in this work, we made an requirement gathering for the access control in the IoT and, then, we designed the ControlChain, an access control authorization architecture that is heavily based on the Blockchain technology. It groups all the main fundamental tendency requirements gathered in one solution. We demonstrate the viability of the ControlChain through the E-ControlChain, a proof-of-concept developed to run over the Ethereum network. We also demonstrated its viability through a cost and a performance analysis of E-ControlChain using a Raspberry Pi as an IoT device. Finally, we evaluate E-ControlChain under a wide variety of attacks and discuss how they can threat it and how to mitigate them.

Keywords: Internet of Things, Access Control, Authorization, Blockchain

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ABAC	Attribute-Based Access Control
ACL	Access Control List
E-ControlChain	Ethereum-based ControlChain
CapBAC	Capability-Based Access Control
DAC	Discretionary Access Control
DHT	Distributed Hash Table
DSS	Digital Signature Standard
IoT	Internet of Things
MAC	Mandatory Access Control
OAuth	Open Authorization
OM-AM	Objective, Model, Architecture, Mechanism
OrBAC	Organization-Based Access Control
RBAC	Role-Based Access Control
RSA	Ron Rivest, Adi Shamir, and Leonard Adleman
SWoT	Social Web of Things
UCON	Usage CONtrol
UMA	User Managed Access
XACML	eXtensible Access Control Markup Language

SUMMARY

1 INTRODUCTION

The Internet of Things (IoT) is a network of physical devices with embedded technology to sense and interact with their internal state or the external environment [1]. These devices are capable of collect, exchange, process and store data. Although a lot is discussed about the correctness of the predicted numbers of IoT devices for the next years [2], even the most conservative forecasts are predicting tens of billions of IoT devices and its produced data will have a potential economic value of USD 11 trillion [3].

The IoT emerged with the objective of providing new "transparent" intelligent services and commodities to facilitate our daily tasks. Its devices are pervading our cities, public buildings, roads, airways, factories, retail stores, offices, hospitals, homes and bodies [4]. Furthermore, with their sensors, communication and information processing capabilities, they affect our interactions on all applications domains: personal, home, government, utilities, enterprise and industry [5], and create the so known smart-everything, like smart homes, smart grids, smart cities, and so on.

One example of the possible intelligent services would be a wardrobe that could suggest combinations of clothes taking into consideration, between other things, the available clothes, user's preferences and scheduled activities, weather forecast for the time out, and past feedback received from sensors and user in similar days. However, note that to provide this service, the wardrobe requires personal and private information like user's preferences, agenda, location and corporal behavior in each activity. Not everyone is willing to grant access to this information if it can easily fall into the hands of unauthorized people or devices. Therefore, together with the great features that arise with IoT integrated systems, there are many security and operational concerns that hinders its broadly adoption by users, governments and industries.

The main IoT adoption concerns are about privacy, technological constraints, social and economical aspects, confidentiality and integrity, reliability and availability, and usability [5]. All of them have different level of importance depending of the application domain, for example, an industry could be more concerned about reliability and availability than a mayor of a smart city. However, although it does not receive all the attention that it deserves [6], the privacy and confidentiality concerns have a big role in the prevention of IoT broadly adoption.

The access control is one of the most important tools to prevent unauthorized access, privacy invasion and confidentiality breaching. Therefore, it is directly or indirectly related to a wide range of the concerns turning around the IoT adoption. In order to grant privacy and confidentiality in IoT, the access control needs to be capable of defining who, when and how someone is authorized to use or access a device or its information. As stated before, to provide automatic services in many environments, the IoT needs to collect, process, generate and consume sensitive, personal and confidential information. Thus, an improper or fault access control system could cause a big privacy and economical harm to individuals, governments and enterprises depending on it.

The IoT concerns are further increased by recently security breaches on IoT devices. Back in 2014, researches were capable of hacking and disabling a car transmission and breaks [7]. This is one example of the vast collection of improper accesses exposed in [8]. In 2016, security breaches on access control systems lead to more than 150,000 world wide IoT devices being compromised and used on Internet attacks [9]. More recently, the VPNFilter malware [10] infected over 500.000 routers around the world by exploiting known access control vulnerabilities of these devices. All these successful attacks keeps remembering us how the currently adopted access control systems are ineffective. It also proves that, although the access control is an old discussion, it still requires a lot of attention, specially now, that we are entering in the IoT era, where all the things are planned to be connected and, in some way or another, exposed to the world.

A complete access control solution involves three components [11]: authentication, authorization and auditing. The authentication identifies the correctly identity of the requester. The authorization (also simply known by "access control") verifies if the requester has the rights to do some operation on the resource. Finally, the auditing (or accountability) allows the posterior analysis of the realized activities in the system. All these components have important roles in securing the system, however, the authorization requires special attention because it is responsible for enforcing the access rules.

Design a suitable authorization mechanism that is capable of dealing with all the complex characteristics of the IoT is a challenge task. Some of these characteristics, like the access control in big networks [12], were already studied in other environments that held them separately. However, they were never studied all together before the advent of the IoT [5]. Common examples of such characteristics are the network size, network connections dynamism, heterogeneity of devices and the data sensitivity.

Most of the researches in the IoT access authorization field employ three traditional and well known architectures: XACML, OAuth and UMA. However, all these three architectures are centralized by design and, therefore, fail to provide essential IoT access control requirements, like scalability, transparency and resilience for the authorization process. Changes in its design are required to turn them into a suitable solutions for IoT. One of these works [13] tried to reduced the gap between these architectures, specifically the XACML, and the IoT environment. They externalized the Policy Decision Point (PDP) to a virtually unlimited resource server. However, it still does not provide transparency and resilience, that we believe to be essential for the broadly adoption of the IoT. Therefore, even with a lot of effort to bring a suitable access control to the IoT [5], there are design barriers in the traditional architectures that prevent them to achieve a complete success [14].

In an attempt to overcome limitations of traditional access controls, turning their authorization process in a more decentralized, user-driven and transparent, some works [15, 16, 17, 18, 4, 19, 14, 20] employed a disruptive technology in the access control, the Blockchain [21]. It is composed by a set of mechanisms that allows it to work as a decentralized data ledger where, eventually, all the networks participants end up agreeing on the same records.

In a more specific definition, the Blockchain is a public, decentralized and Byzantine fault-tolerant ledger, where registers are appended in a chronological order and become more immutable each time a new register is appended to it [22]. All the Blockchain participants are free to verify the legitimacy of all published (appended) data, have a copy of the entire database and publish its own data that, of course, will also be verified by the other participants [23]. Every participant, or a good part of them, has a copy of the Blockchain, for ensuring no downtime. Network participants being capable of verifying if the published content is valid through the agreement and ignoring the not valid content provides no fraud. And each participant being capable of publishing its own data grants no censorship and no third-party interference. Actually, these problems could happen, however, there are pretty good incentives that keep all the network participants honest and, with the majority of them behaving in a honest way, it is extremely improbable that a small group of them can launch a successful attack.

Seeing all this potential, South Korea is investing more than USD 200 million in the development of Blockchain-based projects [24]. The supported projects focus on electronic document distribution, marine logistics, easy real estate, online voting, personal customs clearance and management of livestock records. Besides these, there are plannings to support also Blockchain as a Service (BaaS) solutions. This effort is to bring more legislative clarity and transparency.

FairAccess [4, 18] is one example of proposal that applies Blockchain to the access control. In their work, the Blockchain is used to publish smart contracts that provide valid access tokens (secrets that grant access to a resource). However, as will be discussed later, their proposal also have big issues like the support to token-based authorizations only (does not being compatible with a lot of already adopted IoT access control models), necessity of contact with the owner of the resource for each new access or renew of expired token, the highly time cost involved in getting an access permission, or renewing one, and the lack of integration of the access control with a proper relationship network that has a big importance in a collaborative and integrated IoT. Our proposal promises to tackle these issues and bring improvements to the authorization process.

Although the benefits of using the Blockchain are more easily visible, there are also a few drawbacks. We consider as the worse of them the following problems: information recently appended can be more easily modified or deleted from Blockchain when compared with the more old ones; normally, the process of appending an information to the Blockchain requires the generation of a proof-of-work, which consumes a lot of computing power; furthermore, all the participants are invited to verify all the published information before trust in it. These last two items can be a problem if using low processing power hardware or energy supply, however, as we will see, our solution also brings directions to minimize these effects. Finally, even with these disadvantages, we believe that the benefits of the Blockchain are bigger than its ill effects for a wide range of applications and environments.

In this work, we present the ControlChain [19, 14], an authorization architecture that is heavily based on the Blockchain technology. It uses the Blockchain as a main platform for storing and synchronizing all the data essential for authorization decision-making and, since its second version [14], an off-Blockchain side channel to propagate time-sensitive data. Besides this, it also brings legacy compatibility with a wide variety of access control models already employed on the IoT and also declare and incorporate relationships in the authorization process. Finally, we design, implemented, test and analyze a ControlChain architecture implementation, namely E-ControlChain, that runs over a Blockchain-based network, namely Ethereum.

E-ControlChain was implemented in according to ControlChain specification and currently has access control mechanisms based on attribute, capabilities and ACL. They were chosen because, with minor additions, they bring compatibility with a large variety of access control models and mechanisms already used in the IoT [14], like RBAC [25, 26, 27, 28], ABAC [29], UCON [30], CapBAC [31, 32, 33, 34], ACL [15, 16, 20] and others [35, 36, 37]. It can also be extended to support token-based access, working in a similar way to FairAccess [38], however this feature is not implemented yet. Finally, we expose the theoretical analysis of ControlChain, a result of tests running E-ControlChain in IoT devices and its security analysis.

1.1 CONTRIBUTIONS

The contributions of this paper is the explanation, discussion and presentation, in a comprehensive way, of the following topics:

- The ControlChain, an authorization architecture that emerged from IoT access control requirements.
- The E-ControlChain, a ControlChain implementation model that was design to work over the Ethereum network.

- A performance analysis of the Ethereum and E-ControlChain running on an IoT device.
- A security analysis of the E-ControlChain.

1.2 ORGANIZATION

The remaining of this work is divided as follows:

- Chapter 2 and 3 present the background and related work. In Chapter 2, we show explain the pillars of the access control, define the adopted nomenclature and discuss about the IoT access control authorization requirements. besides this, we show the related works that are based in the traditional architectures XACML, OAuth and UMA. Finally, we also describe the characteristics and specifications of mainly access control models that were already in used in IoT. In Chapter 3, we present the Blockchain and its main characteristics, structures and behaviors. Besides this, we discuss about its application areas, which also contains the access control. Finally we discuss about the Blockchain issues and challenges that could affect the access control.
- Chapter 4 describe the ControlChain, an Blockchain-based architecture to provide a secure and reliable access control authorization in IoT. This chapter describe the entities, relationships and interaction between them, makes a comparison between it and other architectures of the literature, discuss some of the concerns about the viability of Blockchain use in limited resource devices environments. Finally, we also show how access control models already adopted in IoT could be easily adapted to the ControlChain architecture, allowing full compatibility with almost all of them;
- Chapter 5 presents the E-ControlChain, a ControlChain implementation. We expose its main interface and how it can be used to make the access control over the IoT. This chapter also demonstrate how to use it in different scenarios and the burden caused through an IoT scenario, represented by a Raspberry Pi device. Finally, we expose the open challenges and issues of our implemented architecture;
- finally, Chapter 6 presents the final considerations about this work.

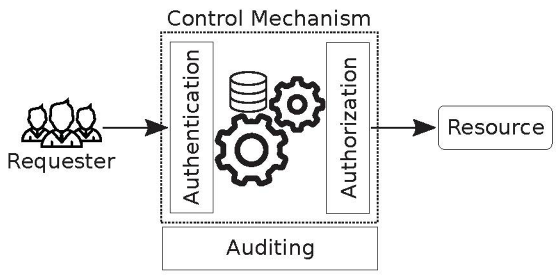
2 ACCESS CONTROL

One of the basic protections to bring privacy and confidentiality to a system or data is to control all the access to it, i.e. block all the undesired access attempts. It is even more necessary in the IoT era, when all we know is expected to be connected to the internet and handling users private and confidential information. These control systems are required to have mechanisms to define who, when and how systems and data can be accessed.

Privacy and confidentiality are both focused in protecting the information, however they have slight different meanings. While ensuring privacy means that the information will not be shared with anyone else, ensuring confidentiality means that the information will not be disclosed to unauthorized entities by anyone with access to it. There are discussions about if the access control in the IoT can grant, besides confidentiality, privacy. If we consider IoT devices as a network element independent from their owners, the access control only provides confidentiality. However, if we consider IoT devices as an integral part of their owners, *i.e.* an extension of them, the access control to them can provide not only confidentiality but also privacy. From now on, we consider the last and, thus, an access control can provide privacy and/or confidentiality. Furthermore, we use the term "privacy" to represent both of them.

A complete solution for securing the access to a system is comprised of three components [11]: authentication, authorization and auditing. The `authentication` identify the correct identity of the requester. For example, one user could prove his identity signing a random message with his private key. The `authorization` verify if the requester has the rights to perform the required actions on the resource. For example, check if a user has the rights to read the measurements of a sensor. Finally, the `auditing` (also known as accountability) allows posterior reviews of the performed activities in the system. All these components and their common positioning in a system are represented in the Figure 2.1. In this work, we sometimes present directions on how to make authentication and auditing, but our mainly investigation is about the authorization process. Also, in order to facilitate the understanding of the leading discussions, we show the adopted nomenclature for the access control in Table 2.1.

Figure 2.1: Access control overview



Source: adapted from [11]

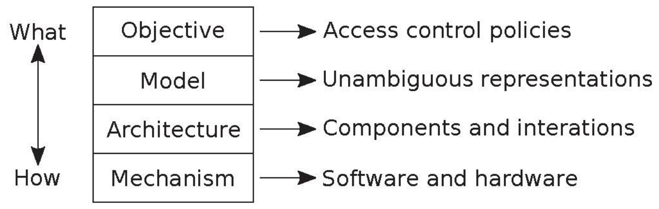
The literature presents a wide range of approaches to allow the authorization process and, very often, complementary authorization approaches are mistakenly treated as rival approaches [5]. Trying to reduce these comparisons between complementary solutions, Ouaddah et al.[5] created an OM-AM-based reference model that try to divide the authorization approaches in independent

Table 2.1: Adopted nomenclature

Nomenclature	Meaning
Requester/Subject	Entity trying to access a resource
Action	Activity performed by a requester on a resource
Resource/Object	Generic device, data or service that can be accessed
Entity	Requester, resource or a set of them
Identity	Value that uniquely identifies an entity
Context	Set of unambiguous attribute-value pair
Context identifier	Ordered pair (entity, variable)
Miner/Validator/Sealer	Who append blocks to the Blockchain

and interoperable layers (Figure 2.2). Although this division is not always straightforward, they try to classify the state-of-the-art in four layers, namely Objective, Model, Architecture and Mechanisms. The `objective layer` concentrates the investigation of the access control policy, i.e., the definition of the high level rules. The `Model layer` defines unambiguous means for apply the access control policy to a system. The `Architecture layer` describes the entities involved in the authorization process, their workflow and interactions to attend models requirements. The `mechanism layer` determine the software and hardware used in the access control policy enforcement. Using this reference model, our mainly related work is on the architecture layer. However, we also make a gathering of models used in the IoT to find out what is the components and characteristics that an architecture has to have to be compatible with them.

Figure 2.2: Ouaddah's reference model



Source: adapted from [5]

Commonly, the first steps in a unknown environment are almost fully based on knowledge acquired from past experiences and already known environments. In the IoT access control field, this was not different. The majority of the architectures employed in the IoT access control until now are based on architectures that was widely used in other computing scenarios, most of them centralized ones. We call them the `traditional architectures`. In this chapter, we discuss works based only on such architectures. In Chapter 3, we will present approaches based on Blockchain.

In this chapter, we first discuss about the IoT access control requirements (Section 2.1), then we present the traditional architectures and the works that apply them in the IoT (Section 2.2). Finally, in order to be able to design an access control framework that is compatible, by design, with the maximum number of IoT access control models (see the OM-AM-based reference above), we also review some of the main access control models employed in the IoT and the works that makes use of them (Section 2.3).

2.1 IOT ACCESS CONTROL REQUIREMENTS

A secure IoT-suitable access control system for IoT must be capable of (1) precisely identify the entities with robust anti-fraud mechanisms in order to prevent non-authorized malicious devices or users of accessing resources, (2) enforce the access policies defined by resource owners and would be (3) gainful if it attends all the following characteristics [39, 5, 18, 14, 13]: scalability, lightweight, transparency, user-friendly, fault-tolerance, privacy-friendly, delegation-capability, context-aware, fine-grained, relationships-aware and legacy-compatibility. Therefore, developing an access control for the IoT is a challenge task mainly because it commonly has to deal with these characteristics. We will use them to compare the architecture from state of the art with ours in Chapter 4. Next, we describe why they are important to the IoT.

Scalability. Even the most conservative forecasts are predicting the existence of tens of billions of IoT devices for the next years [2]. A centralized access control architecture will easily become a bottleneck and give limits to this fast growing. In this type of scenario the most recommended is the adoption of decentralized or distributed architectures.

Lightweight. The IoT is known to be composed by a huge diversity of devices, ranging from powerful processing ones, like smartphones, to powerless ones, like RFID sensors. All these devices will be protecting some resources and, therefore, checking the existence of authorization for each access attempting. Therefore, it is important that the architecture provide low cost operations, mainly, for checking authorization as it will be accomplished by IoT devices in most of the cases.

Transparency. A considerable part of IoT will be composed of personal and intimate devices. They will collect and manage user's private information. The more intimate is the information, more concerned the owners are about who, when and how the information are being handled. The authorization process is required to be transparent in order to win the trust of users.

User-friendly. Even in the IoT era, there are still users with almost no knowledge in computer mechanisms and systems. This means that a confused or complicated authorization system has a good chance to keep away a lot of potential IoT users or turn them into victims of unauthorized accesses. So, the access control has to be user-friendly and has to have a low learning curve for new users.

Fault tolerance. The IoT will be surrounded by unreliable devices and network connections. Some examples of these instabilities are: IoT devices could run out of power supply, wireless channels could suffer from interference, and mobile devices can move away from each other and loose connections. Thus, these and other instabilities generated by the IoT characteristics should not affect authorization mechanism.

Privacy-friendly. If not all, at least the personal and intimate information has to be strong protected. So, an access control has to be based on reliable and robust mechanisms, like cryptography, that enforces access policies and hide the information from undesired accesses. This is also an important requirement to win the trust of users.

Delegation-capability. Delegation is the forwarding of a given permission from one entity to another. However, while providing it brings a new set of functionalities, it also could be dangerous since one single entity with the permission could forward it to any number of entities. Therefore, we argue that the owner has to be capable of controlling the delegation involving its resource.

Context-aware. In the IoT, all things are meant to be interconnected and to exchange information. The exchanged information can help the IoT to make better decisions. For example, if there is an emergency, security devices could unlock all doors and let anyone to enter the site. Therefore, it is desirable that the access control can deal with context-aware access rules.

Fine-grained. A lot of works already saw the potential of having an access control with fine-grained control [40, 13, 41, 42]. As the IoT has a wide variety of information, the fine adjustments in the access control is very welcome in complex environments that requires it.

Relationship-aware. Collaborative, integrated and interconnected system that takes in consideration relationships are becoming more and more popular. An example of this popularity is the recently wave of games in the social networks that allows user friends to interact with it in its game play [43]. Probably a lot of these games would not have such acceptance if its interaction was only with completely strangers. Therefore, in a similar manner, we also see a great potential in exploring relationships in the IoT access control. A study [44] with 245 persons revealed a major impact of relationships and context on access control, corroborating with our understanding. They also showed the consequence of false-positive and false-negative in authorizations.

Legacy-compatibility. As we will see, there is a wide range of access control models been proposed and used in the IoT. Each one has its peculiarities and we don't believe that one single model is capable of completely satisfying all the different scenarios and environments requirements. Furthermore, in order to does not require the disable of all the old models, it is desirable that a new architecture be legacy-compatible and, also, support as many models as possible.

2.2 TRADITIONAL ARCHITECTURES

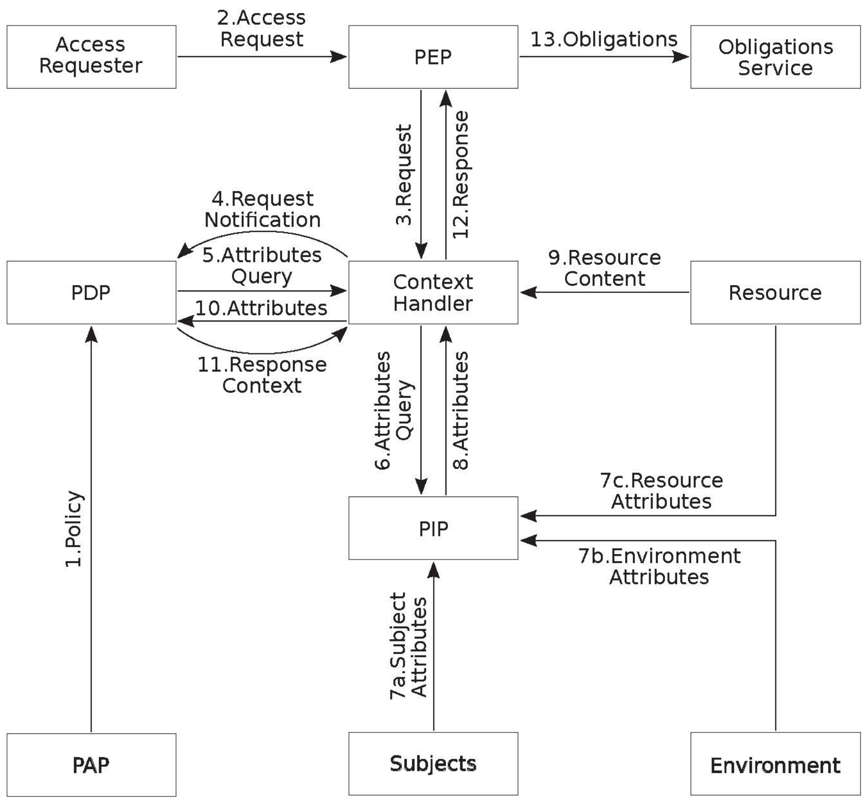
The access control architectures compose the second layer (bottom-up orientation) of the reference model defined by Ouaddah et al. [5]. They describe the entities involved in the authorization process, their workflow and interactions. We named as "traditional architectures" all the access control architectures that were created before the IoT era and were employed or adapted for use in the IoT. There are three main traditional architectures employed in the IoT access control: XACML [45, 46], OAuth [47] and UMA [48]. Next, we describe the original architecture and discuss some works that employ or adapted them for the IoT.

2.2.1 XACML

The eXtensible Access Control Markup Language (XACML) is a standard that includes a declarative fine-grained and attribute-based access control policy language, an architecture and a processing model to evaluate requests. We present the XACML components and its data flows in the Figure 2.3. As can be seen, the XACML has many components, however, the Police Enforcement Point (PEP) and Police Decision Point (PDP) can be considered the main ones. The PEP is responsible for enforcing the application of the policies. To do so, it intercepts the access request and consults the PDP about the existence of authorization for the request. The PDP can base its decision on the policies (provided by the PAP - Police Administration Point) and on subjects, objects and environment attributes (handled by the context handler). Upon the receipt of the PDP evaluation result, PEP make the access decision and executes the obligations defined by the PDP.

Both, [49] and [13], propose to use the XACML in the IoT access control. [49] built an architecture on top of the java-based OSGi framework. Among other tasks, the OSGi modules are responsible for the management of the discovered devices and for the access control. For the access control they adopted a hybrid approach of OSGi User Admin service and XACML. The OSGi User Admin service manage subject roles and the XACML specify the policies and enforce them. [13] presents a brief list of authorization framework requirements. In order to fulfill two of these requirements, specifically the fine-grained and low overhead on the object, they choose to use the XACML architecture with the PDP externalized to a virtually limitless back end. In order to a subject get access in their proposal, a subject needs to: (1) get device's

Figure 2.3: XACML architecture



Source: adapted from [46]

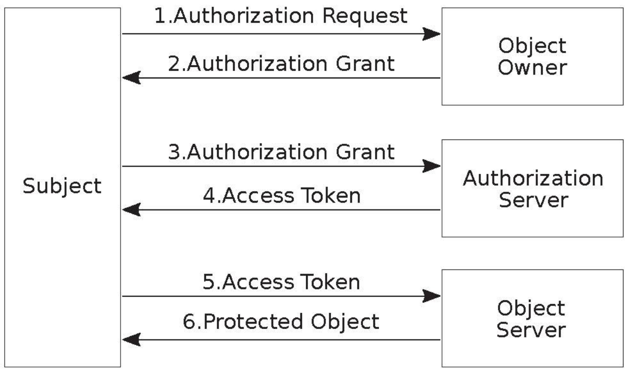
meta-data, like object's trusted authorization server and URI, in a resource discovery; (2) get an authorization assertion from the object's trusted authorization server; (3) use the authorized assertion to access the object. Finally, they also discuss about lightweight protocols that could be used in their proposal.

2.2.2 OAuth

The Open Authorization (OAuth) is a token-based authorization protocol, commonly used for admission control. There are two special pieces of information in this protocol: the authorization grant and the access token. The authorization grant is a credential representing the owner authorization to access the object and it is used to obtain the access token. An access token is a credential that can prove the expedition of the authorization to the client. It could be, for example, an apparently random string that the object being accessed can somehow check its validity. The Figure 2.4 shows an overview of the architecture and the protocol flow. Initially, the subject request, to the owner, the authorization grant to access a protected object (1). After the authorization grant is issued by the owner (2), the subject exchanges it for an access token through the authorization server (3-4). Finally, the subject access the protected object using the access token (5-6).

There are four authorization grant types: authorization code, implicit, resource owner password credentials, and client credentials. In the *authorization code type*, the client directs the owner to an authorization server. After the owner authenticates and give the access permission, it redirects the owner back to the client together with the authorization code, which can be exchanged by an access token through the authorization server. In the *implicit type*, the owner

Figure 2.4: OAuth architecture



Source: adapted from [47]

issues the access token directly, i.e., there is no authorization code. In the *resource owner password credentials type*, the credentials (for example, user and password) of the owner are used by the client to obtain an access token. In the *client credentials*, the credentials of the client are used for the authorization. This last case occurs when the client is acting in its own behalf, i.e., it is also the owner, or when it is requesting access to protected objects based on authorizations previously arranged with the authorization server. The benefits and security risks of each one is described in [47].

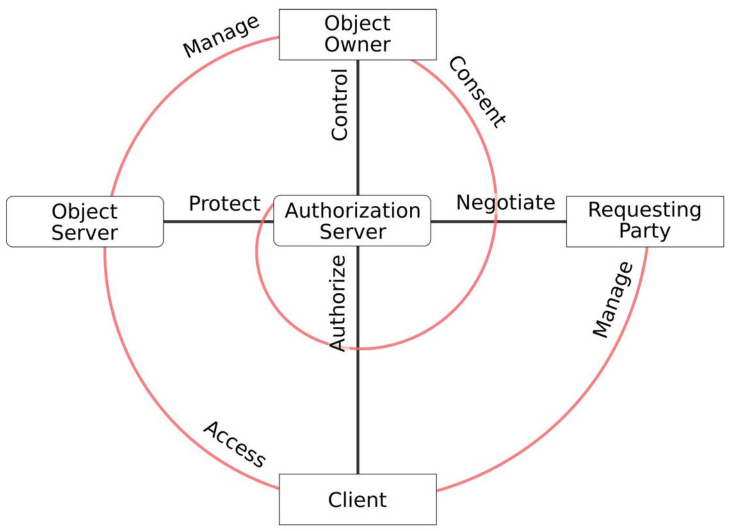
Cirani *et al.*[50, 51], Varadharajan *et al.*[6] and Selander *et al.*[52] are example of works based on OAuth. Cirani *et al.* introduces an OAuth-Based architecture namely IoT-OAS[50]. It implements all the back-end OAuth logic and is backward compatible with standard OAuth clients. They also complement the IoT-OAS with a set of messages to manage ownership and shared access to objects, allowing even a proactive access authorization[51], i.e. authorizing the access before the request of authorization. Varadharajan *et al.* discuss about the IoT security risks and challenges, and data security requirements and recommendations[6]. In order to avoid security problems, the authors suggest the usage of selective masking of the data, the adoption of contextual information and OAuth in the access authorization. Finally, Selander *et al.* uses CoAP together with OAuth to build a framework for authentication and authorization in constrained environments[52].

2.2.3 UMA

The User Managed Access (UMA) is an OAuth-based protocol that unifies, in the perspective of the object owner, the control point of object access authorizations. As shown in Figure 2.5, the UMA architecture is composed by five entities: Object Owner (OO), Object Server (OS), Authentication Server (AS), Client and Requesting Party (RP). The OO provides the object in the OS, delegates the object protection to authorization server and control the access to it trough the AS. The RP negotiate with the AS to get access to the object and manage the client that wants to access the object. The AS protects the object, negotiate access with the requesting party and authorize clients to have access to the object. All the tasks executed by the AS needs the consent from the OO. Finally, the Client requests access tokens to AS and uses it to access the object in the OS.

Cabarkapa[53], Rivera *et al.*[54] and Tschofenig *et al.*[55] proposed the use of UMA in the IoT. Cabarkapa proposes to use an UMA-based architecture for controlling authorizations on a Social Web of Things (SWoT)[53]. In its work they present the characteristics and structures of

Figure 2.5: UMA architecture



Source: adapted from [48]

a SWoT, a study of the security protocols and architectures, like XACML, OAuth and UMA in order to identify the more suitable solution for it. After choosing UMA for their architecture, they provide a detailed description about all the phases involved in the architecture system interactions. Rivera *et al.* applies UMA to provide a unified access control in a heterogeneous IoT environment composed by IoT low processing power devices (dummy devices) and intelligent agents (powerful devices)[54]. Finally, Tschofenig *et al.* shows how OAuth can be integrated in the UMA and also presents a mapping of their architecture for several use cases[55].

2.2.4 Traditional architectures flaws

The traditional architectures have been employed in a lot of works. However, all of them have big flaws when taking into consideration the IoT access control requirements. For example, all of them depend on centralized architecture components. This dependency makes the traditional architectures-based proposals susceptible to a wide range of limitations, like having the single point of failure problem, applying limits the growth of the IoT, requiring third-party trust, does not giving support to off-line mode, generally involving less privacy and others. This limitations goes against a lot of IoT systems requirements, like availability, scalability, transparency and so on. Finally, Table 2.2 shows the IoT requirements achieved by each one of the traditional architectures (more details in Section 4.5.1). Furthermore, in Table 2.3, we compare the possible positive aspects of both approaches, centralized and decentralized.

Although the architectures show how the components will interact to enforce the policies, it says practically nothing about how the access policies rules are structured. The definition of this structure is role of the access control models (See Figure 2.2). However, architecture has to give support to them in order to be compatible with them. Thus, we made a gathering of the main access control models employed in the IoT. We present them in Section 2.3.

2.3 ACCESS CONTROL MODELS

The access control models compose the third layer (bottom-up orientation) of the reference model defined by Ouaddah et al. [5] summarized in Figure 2.2. They define unambiguous

Table 2.2: IoT requirements achieved by traditional architectures

Requirement	XACML	OAUTH	UMA
Scalability	no	no	no
Lightweight	yes*	yes	yes
Transparency	no	no	no
User-friendly	no	no	no
Fault tolerance	no	no	no
Privacy-friendly	yes	yes	yes
Delegation capability	yes	yes**	yes
Context-aware	yes	no	no
Fine grained	yes	no	no
Relationship-aware	yes***	no	no
Legacy-compatibility	yes	no	no

* IF only the PEP is on the object server
** It allows applications to execute actions in behalf of the user
*** Through the Policy Information Point (PIP) and context

Table 2.3: Advantages of centralized and decentralized access control

Centralized	Decentralized
Easy maintaining and managing	No existence of single point of failure
Less time to discover and fix bugs	Bugs rarely affects all devices
Less fragmentation of devices and systems	Allows off-line mode
Easier reuse of traditional mechanisms	Possibility of overcoming centralized limitations
Less complex mechanisms	No need of dedicated authorization servers and controllers

means for apply the access control policy to a system and are implanted over architectures. This section reviews some of the most common access control models used in the IoT and works that apply them in the IoT.

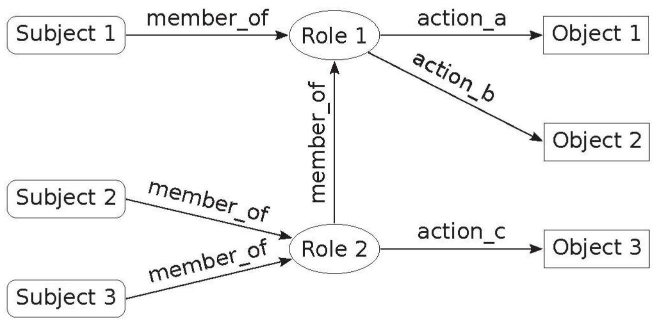
RBAC. The Role-Based Access Control (RBAC) [56] is a model composed by subjects, roles and objects. The subjects become members of groups, namely roles, and the roles receive authorizations to perform actions over the objects. Therefore, the only way to a subject access an object is if it is member of a role that has the proper authorization. Another feature of the RBAC is the multi-role relationship. It allows a role to become member of others roles and, therefore, inherit all the authorization from those roles.

Figure 2.6 shows an example of the RBAC structure. In this example, “Subject 1” is a member of “Role 1” and, therefore, it can perform “action_a” over “Object 1” and “action_b” over “Object 2”. “User 2” and “User 3” are member of “Role 2” and, therefore, they can perform “action_c” over “Object 3”. Furthermore, as the “Role 2” is also a member of the “Role 1”, all the allowed actions for members of the “Role 1” are also allowed for the members of the “Role 2”.

OrBAC. The Organization-Based Access Control (OrBAC) [57] is very similar to the RBAC, i.e., roles are used to group subjects with the same authorizations. However, as shown in Figure 2.7, there are three main differences: (1) OrBAC not only abstracts the subjects using roles, but also makes abstraction of actions and objects, namely, activity and view, respectively; (2) all these abstractions are organizations dependent, i.e., each organization define its own vision of the world through roles, activities and views; Finally, (3) OrBAC also use the context for the authorization process.

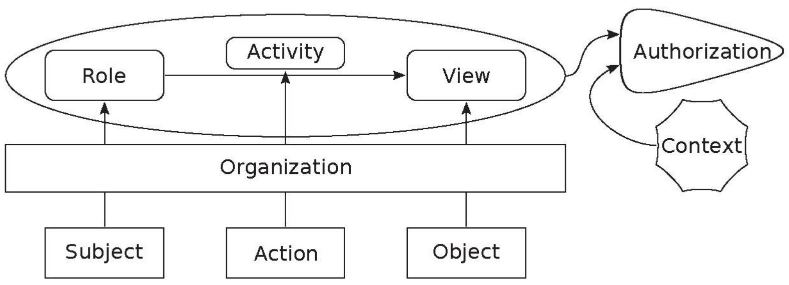
ABAC. The Attribute-Based Access Control (ABAC) [59] relies upon policies evaluation of subject and object attributes, and also has support to evaluate environment conditions (context). The policies and attributes are define by an authority and can be used to enforce both Discretionary Access Control (DAC) and Mandatory Access Control (MAC). While DAC controls the access

Figure 2.6: RBAC model



Source: adapted from [56]

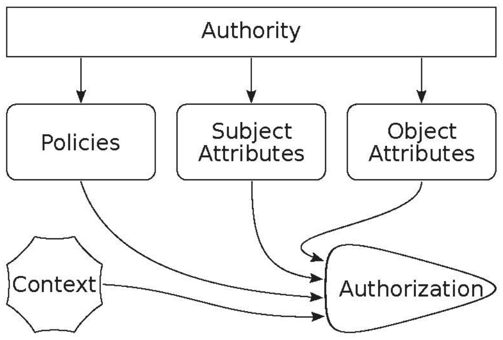
Figure 2.7: OrBAC model



Source: adapted from [58]

based on the subject identity or groups that it participates identities, the MAC test the attributes of both, subject and object, against the rules to decide if the access can be granted. Figure 2.8 shows an example of ABAC structure.

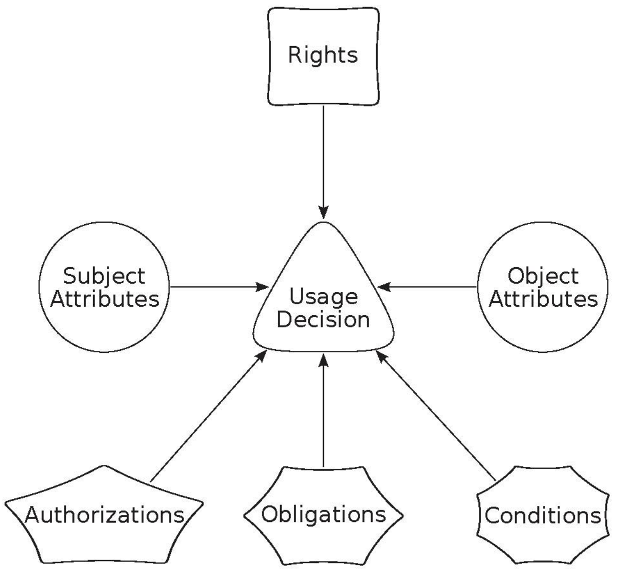
Figure 2.8: ABAC model



UCON. The Usage CONtrol (UCON) [60] model is composed by subjects, objects, subjects and objects attributes, actions (namely rights), rules (namely authorizations) contexts (namely conditions) and obligations. As can be seen in Figure 2.9, it is very similar to the ABAC. The only exception is the obligations. The obligations are predicates that verifies mandatory requirements that have to be performed by the subject before and during the access. Also, by design, the UCON requires the immediately interruption of an access if the authorizations, obligations and conditions are not satisfied at any time. So, devices need to keep monitoring and validating the access for each change made in rights, authorizations, obligations and conditions.

CapBAC. The Capability-Based Access Control (CapBAC) [61] is based on the ordered pair *(o,a)*, where *o* is the identification of an object and *a* is the actions that the subject in

Figure 2.9: UCON model



possession of the capability have over the object. Originally, anyone that posses this pair can perform the described actions and, therefore, it is completely transferable. As we will discuss in Chapter 3, newer CapBAC commonly append the subject identity to the pair in order to avoid unwanted access authorization delegation.

2.4 CONCLUSION

In this chapter, we presented the fundamentals of an access control, showing the components, its functions and the how they complement it other. After, we highlighted the requirements of an IoT access control, trying to identify the main desired characteristics for an actual access control system. Then, we discussed the traditional access control architectures and, finally, the most common IoT access control models in order to help with future decisions, specially, in the compatibility of our proposition as we will show in the Chapter 4.

During the requirements gathering, we miss two requirements when searching the literature: relationship-aware and legacy-compatible. In our opinion, they are important requirements that worth to be present in IoT solutions. The first one, relationship-aware, allows still more fine-grained and contextual information to be used in the authorization rules. The second one, legacy-compatible, gives the architecture more support for the already existent access control models, expanding the actuation area and easing its adoption.

Traditional access control architectures proved to have a great value in the access control. However, in our analysis, none of them provide a suitable access control for the identified requirements of the IoT. They fail to provide scalability, require third-party trust, does not give support to off-line working mode and, depending on how it is structured and implemented, little or no transparency. Therefore, it is evident that reformulated architectures are necessary. In Chapter 3, we will discuss about a revolutionary and emerging technology that have been used to build many interesting solutions for many areas. besides this, we also believe that it can key to revolutionize access control solutions and will be part of the wave in the replacement of the traditional ones.

3 BLOCKCHAIN

Until recently, almost all financial transactions were intermediated and centralized by third party organizations. As discussed in Chapter 2, this dependency brings several drawbacks, like the requirement of trust in the central entity and the lack of transparency in the operations. In a pursue for a secure and decentralized transaction environment, the Bitcoin cryptocurrency [21] was created. Roughly, the Bitcoin is composed by two things: (1) the application protocol and the (2) Blockchain. The application protocol defines the transaction rules (what information are required and when a transaction can be made) and the Blockchain is a multi-field construction composed by a wide variety of technologies, like cryptography, mathematics, economic model, peer-to-peer networks and algorithms [62] that grants security to transactions.

Since its creation, the Blockchain popularity has been getting more and more to unthinkable levels. Although a good part of this popularity is because of its successful adoption in the cryptocurrencies, it also has proved its capability to revolutionize many other areas like access control, IoT, smart contracts, smart property, digital content distribution, botnet, P2P broadcast protocols and so on [4, 63]. The next sections present its application in some of these areas.

This chapter is divided as follows. Section 3.1 discuss some of the areas where the Blockchain was successfully employed. Section 3.2 presents the main structures and tools existent inside the Blockchain. Section 3.4 discuss the application of the Blockchain in the access control. Finally, Section 3.5 reveals the issues and challenges of Blockchain-based access control approaches.

3.1 APPLICATION AREAS

With its origin in the cryptocurrencies, more specifically in the Bitcoin, the Blockchain is the main enabler technology of several of them [23, 64, 65, 66, 67]. besides the already cited researches, there are a plenty of other areas [68] where the Blockchain is being successful employed, like cryptocurrencies [21], transportation systems [69], management of medical records [15], decentralization of the Web [70], predictions [71], applications platform [23], configuration [72], communication [73], collaborative resource usage [74] and information sharing [75]. Some of this works are presented in this section and others are presented in Section 3.4.

The authors of [69] argument that without the proper security and trust in transportation systems, the high-level intelligence of it would be fake and fragile. In order to bring these characteristics and also decentralization to theses systems, they explore the Blockchain concept, creating the Blockchain-Based Intelligent Transportation System (B^{2}ITS). The B^{2}ITS follows a model of seven layers, ranging from physical layer (devices) until the application layer (potential use cases scenarios). The intermediate layers are the data, network, consensus, incentive and contract layer: the data layer defines the blocks; the network layer brings the propagation methods of the data; the consensus layer holds the algorithms used to achieve consensus in a decentralized system; the incentive layer defines the economical rewards for the network collaborators behave honestly; and, finally, the contract layer packages many scripts, algorithms and smart contracts, i.e., elements that can respond automatically to stimulus (new data entries) in the network. They also give a brief description of how the B^{2}ITS can be a first step to a Parallel transportation

Management Systems (PtMS), a system that allows the interaction of the real-world with its corresponding artificial or virtual counterparts.

Ethereum [23] is a Blockchain-based application platform for distributed computation. It has support to smart contracts and provides a decentralized turing-complete virtual machine, namely Ethereum Virtual Machine (EVM). The use of smart contracts is indicated for scenarios like market trades, register of debts or promises and similar cases. It allows developers to construct and run decentralized applications over worldwide spread nodes. It has also a cryptocurrency, namely Ether, that can be used to pay the collaborators for their processing resources. Currently, the Ethereum serves as a foundation to countless applications. Section 3.3 shows more detail about it.

The Golem project [70] aims to decentralize the Web and computing, creating a worldwide supercomputer. This supercomputer is composed by users computers connected to the public network and that accepted to share their computational power. The reward method uses the Ethereum Platform and has an interesting and dangerous payment system. Instead of given a small amount of payment for each user that would disappear in the fees charged by the mining nodes (see Section 3.2.2), a lottery scheme is placed to choose one of the collaborators to receive all the payment. This could be seen as unfair, however everyone will get payed eventually. It is worth noting that, while it is possible to gain with reduced fees for mining and therefore reduce the overall computation cost, it is also a dangerous approach because the entire system could fail if someone find a way to cheat the lottery. Furthermore, Golem also offers a peer-to-peer network, a reputation system, trading systems, and task computation.

As a lot of systems, a centralized prediction market can also easily suffer from mistakes and manipulations. So, the Augur project [71] was created as a decentralized prediction market platform based on Ethereum that reward users for correctly predicting future events. Participants of this platforms can buy and sell shares about an event. The prices of this shares are based on the probability of that event occurs. Everyone holding a share that correct predicted events, receives a rewarding. Its forecast tool is based on the principle of the wisdom of the crowd, which states that the opinion of large groups of peoples is usually more accurate than the one from a single expert.

Huh *et al.* uses smart contracts in the Ethereum platform to manage and configure IoT devices[72]. To demonstrate their solution, they constructed a test environment composed by a smartphone and three Raspberry Pi devices. The smart phone is responsible for the configuration of energy save mode policies through the contract. besides this, one Raspberry Pi device behaves like a power meter and is responsible to publish, through the contract, the amount of consumed energy. The other two Raspberry Pi represent home devices (an air conditioner and a light bulb) and continue monitoring the contract for policy changes or power meter updates. When the consumption of energy extrapolates the value defined in the policies, they enter in a save mode energy.

Biswas *et al.* proposes a security frameworks that allow entities from a smart city to communicate without compromising the privacy and security[73]. Its framework is divided in four layers. In the first layer (physical layer) are the sensors and actuators. The second layer (communication layer) includes different network link layer protocols and mechanisms to secure the transmitted data, like cryptographic algorithms. The third layer (database Layer) holds the distributed ledgers, public or private. Finally, the applications are in the last layer (interface layer).

Kianmajd *et al.* applies the Blockchain in a collaborative resource usage system. For his use case it used a virtual smart electric micro-grid[74]. It is composed of four phases: bidding, allocation, recording and verification. Unfortunately, this work has so many gaps that it turns out impossible to us give more details about it.

Finally, the list of application domains that can benefit from Blockchain seems to be infinity, and it covers many other different areas like IoT, smart contracts, smart property, digital content distribution, botnet and P2P broadcast protocols [63]. Furthermore, even banks are starting to use it for information sharing [75].

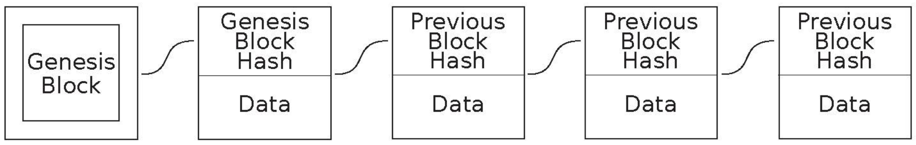
As we saw in this section, although the Blockchain was created for use with crypto-currency, its characteristics turned out to be useful in a wide range of areas. In the Section 3.4 we discuss one of this areas that we did not deeply discussed yet, the Blockchain-based access control. However, before discussing it, we explain the main mechanisms and assumption of the Blockchain's mode of operation in Section 3.2.

3.2 BLOCKCHAIN'S MODE OF OPERATION

The technologies employed in the Blockchain are arranged in a such way that they create a public, decentralized, Byzantine fault-tolerant, immutable, chronological-order registers database that stores all the transactions ever made in the platform. The public characteristic guarantees that everyone participating in the Blockchain has access to the ledger and can verify its correctness. The decentralized and byzantine fault-tolerant statement guarantees that the system keeps running correctly even under a failure or malicious behavior of a considerable part of the network. The immutable characteristics guarantees that once a register is included in the ledger it cannot be changed or removed in a normal network behavior (Section 3.5.1 discuss about some special cases in which changes or removals can occur). The chronological-order means that all registers have a fixed sequence (not necessarily the same of its generation).

In a more technical view, the Blockchain is an ordered sequence of blocks. These blocks are composed by a predefined amount of information and by a fundamental field that holds a link to the immediately prior block in the sequence, thus, creating a chain of blocks (see Figure 3.1). The information contained in these blocks is flexible and only depends on the system requirements where the Blockchain is being used. Furthermore, the ordered list of data records obtained from the sequence of blocks is commonly called ledger [63].

Figure 3.1: Blockchain main structure



In order to grant the security of the applications running over the Blockchain, some systematic safety instructions have to be followed by all participants in the network. If one of them choose not to follow these instructions, it will only be ignored by the rest of the network, harming only the participant that chose not to follow. Therefore all participants have the incentive to continuously follow the safety instructions and collaborate with the network.

We can divide the Blockchain usage in four phases: data creation, block construction, block appending and consensus reach. These phases are discussed, respectively, in Sections 3.2.1, 3.2.2, 3.2.3 and 3.2.4. After, we discuss the concept of wallets in Section 3.2.5.

3.2.1 Data creation

The data creation phase is the process of creating a message for publication in the Blockchain. It needs to be built according to the application protocol in order to be accepted by the network. Therefore, the rules for its creation are exclusively made by agreements between the participants of the application running over the Blockchain network. Normally, it is required signatures and data that could be validated.

In Bitcoin, every created message needs to be a transaction. It is composed of an input and an output. The inputs of a transaction are Unspent Transaction Output (UTXO), i.e. outputs from prior blocks that weren't used yet. The outputs of a transaction are composed by the favored identifications, i.e. the identifications of who will receive the payment, and the amount that each favored will receive. As soon as these transactions become part of the Ledger, the favored ones can use they as inputs for their own transactions. besides the Bitcoin particularities, many of its safety instructions are very common in many other Blockchain-based application. Examples are the use of hashes and signatures [21, 23, 4, 14].

Signatures are used to avoid repudiations. In normal situations, it is unfeasible for an entity to sign a message as another one. To sign messages, an entity needs to generate a pair of cryptographic keys (sk, vk) using cryptographic schemes, like the RSA [76] or the Digital Signature Standard (DSS) [77]. sk and vk keys are known as signing key (or private key) and verification key (or public key), respectively.

The algorithm to generate a signature M_{sig} for the message M, requires sk and M as input. In the other hand, the algorithm to validate a signature requires vk, M and M_{sig} as input. Therefore, while the holder needs to keep sk in secret, the vk can be public to anyone. One important still holding premise of public key schemes is the impossibility to infer the sk having access only to vk. However, because of their mathematical relationship, it is possible to verify if a signature of a message was generated a signing key using only the verification key, the message and the signature. Furthermore, a signature produced by sk over the messages M and M_2 will be equal if and only if $M == M_2$. Note also that, although the vk can be thought as an identity, it does not necessarily need to be bound to a real identity and, in this case, provide pseudo-anonymity.

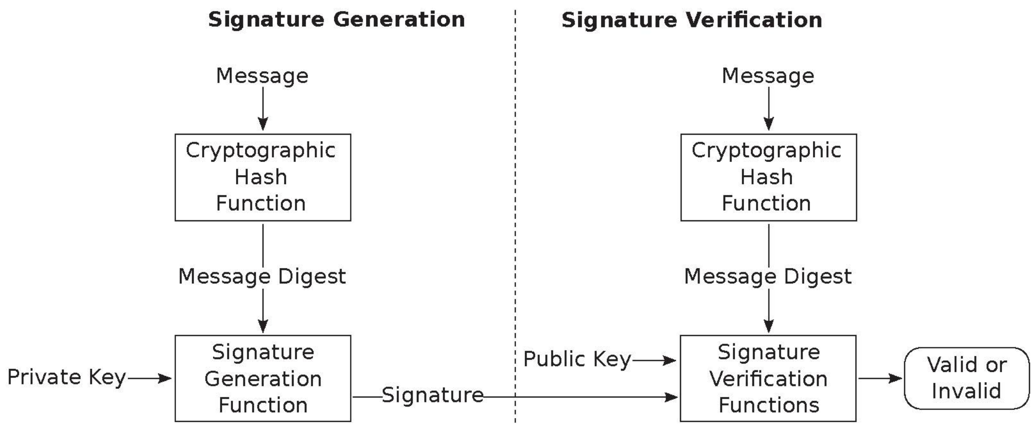
One problem with the public key signature algorithms, like the ones adopted in RSA and DSS, is their intense resource usage to sign large messages. Therefore, cryptographic hash functions [78], known also only by hash functions, are usually adopted to compress the message before signing. They are a one way function that receives a message of any length as input and always condenses it into an output, known as digest, of an arbitrary length. The digest length is dependent of the used algorithm. Of course, collisions, i.e. equal outputs for different inputs, can occur because it converts any message into a fixed-length digest reducing the representation space. Two important characteristics of hash functions to ensure its security even with the reduced representation space are presented in Section 3.2.2. The signing message and its validation process is summarized in the Figure 3.2.

Therefore, the complete creation data process commonly involves the generation of the message, the extraction of the digest and the signing of it. After all these steps the message has to be broadcasted to the network and reach the higher number of participants as possible. Section 3.2.2 presents what happens when it is received by other participants.

3.2.2 Blocks construction and appending

The ledger data is organized in blocks and these blocks are ordered in the Blockchain. The ordering is achieved by always creating new blocks with a reference to its predecessor block

Figure 3.2: Signature process flow



Source: adapted from [77]

in the Blockchain, thus, making a chain of blocks. Each block contain part of the valid messages received from the data creation phase. The creation process of new blocks, known as mining, is made by special workers, namely miners or, depending on the type of "mining", validators or sealers. Although any participant of the network can be a miner/validators/sealers, there are Blockchain network that has fixed ones (working through trust more than computing power). The first and most common mining type is the proof-of-work. Its process consists of finding a nonce (a number) that causes the hash of a block to be below a predefined value. When someone achieves this, the block can be considered mined. After its mining, each network participant receives a copy of it from other peers, verify its data and, if it is correct, append it to their local Blockchain.

As stated before, the data produced in the message construction phase is broadcasted to all network participants. Thus, eventually, all miners will have access to it. Each one has to verify the correctness of the data and its structure, and if the messages is in according to with the application protocol, for example, checking if the user has enough balance to do a transaction. These steps are required to protect the sanity of the Blockchain information and avoid waste processing and time with invalid data. A mined block will be ignored by the network if it contains invalid information. Therefore, there are pretty incentive to everyone check the produced information.

In Bitcoin, checking the transactions can be divided in three main steps. First, make sure that the inputs are UTXO (unspent outputs) and that its amount is not less than the output amount. Second, verify if the identity trying to make the transactions are the really owner of the inputs checking signature challenges. Finally, check the integrity and authenticity of the transactions also using cryptographic signature checking techniques. Note that these steps aren't required to be executed in the presented order.

Blocks ready to be mined are composed by valid messages and a reference to its predecessor (last known mined block). This reference is, commonly a hash of the predecessor block. In this stage, the Bitcoin miners starts the mining process, i.e. pursue a nonce that creates a valid proof-of-work. This process is a way to establish an agreement of the order of the messages (or transaction, in the Bitcoin network) in the ledger. As this nonce is hard to find, this process reduces disputes [79], however, they still occur (see Section 3.2.4). One problem with this mining methodology is its serious computation processing spending to accomplish the complex puzzle solving.

Because of proof-of-work complexity, a wide variety of other block ordering agreement methodologies were proposed after the arising of Bitcoin. These methodologies range from simple voting process to more complex tasks like proof-of-stake, proof-of-authority, proof-of-activity and proof-of-publication [80]. Some of them, like proof-of-stake and the proof-of-authority, are quite popular. In this methodologies, the "miners", known as validators or sealers (respectively), are selected by the network, reducing drastically the competition, if not ending with it. This makes the mining process much more cheap, however, commonly, requires trust in some of the participants.

When the proof-of-work is chosen as the mechanism for a Blockchain, three principles are fundamental for the used hash function: uniformity, avalanche effect and no result insight. The uniformity determines that every output are evenly possible for a random input. The avalanche effect ensures that a minimal change in the input causes the output to be completely different. The no result result insight determines that should be no clues of the hash output before its computation finishes. These principles hinders hinder and frustrate all the guesses on how the hash will behave based on the input. For example, the SHA-256 hash of "foobar" is
`c3ab8ff13720e8ad9047dd39466b3c8974e592c2fa383d4a3960714caef0c4f2`
and the SHA-256 hash of "fooba." is
`e3104c068edfbfc6020db9891e74278989f46fa85a31211ef6441262cf7019f6`.
Although the difference between the input strings is the substitution of a "r" for a ".", their resulting hashes are very different.

Because a lot of processing power is spent in the process, proof-of-work miners need to be rewarded to continue the process of mining. The reward is dependent of the application. For example, in Bitcoin, it is acquired from two sources. The first source is from each mined block. The miner responsible for a mined block can have, in the block, a special transaction that grants it a predefined amount of Bitcoins. The second source is payments from users. They originate from tips, i.e. extra payments, for mining the transaction and are calculated by subtracting the transactions outputs amount from its inputs ones. Therefore, transactions with a higher tip has also a higher chance to be mined in short time period.

After a block is mined, this block is broadcasted to the network participants and the Block appending phase described in Section 3.2.3 starts.

3.2.3 Block appending

When a new mined block is received, the participant has to verify if all the messages in the block are in according to the application protocol. besides this, it also has to check if the block follow the rules defined by the Blockchain network. For example, in the proof of work, it need to check if the proof defined in the block is valid and if the last mined block is correctly referenced. If the block does not pass on any of these validations, the participants reject the block. Otherwise, they append it to their local Blockchain.

If the participant is a miner, it has to adjust some aspects of its current mining block. It needs to update the reference of the last mined block and, eventually, remove or replace mined messages. Then, the mining process has to start over.

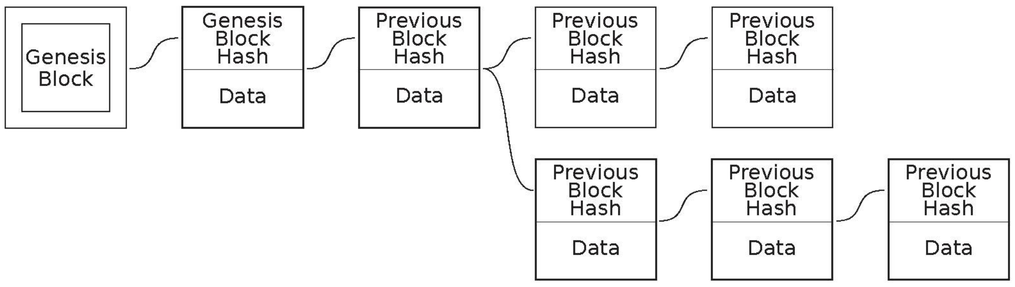
Eventually, the Blockchain also can fork and give origin to two or more branches, forming a tree-like structure instead of being linear. It occurs when two or more miners solve the puzzles independently of each other and broadcast their findings to the network. This scenario is discussed in the Section 3.2.4.

3.2.4 Consensus establishment

If the new received mined block, instead of pointing to the last known block, points to a prior block, a new branch arose (see Figure 3.3). This can happen by uncountable reasons, like delay of propagation, bipartition of the network because of link problems or malicious participants, or even the mining at same time. If the Blockchain application does not support branches it also is required to implement deterministic consensus algorithms to deal with synchronization problems and resolve disputes.

Some platform does not support multiple branches because it would be complex to control the branches or it could bring inconsistencies to the stored data. For example, in Bitcoin, the formation of a second branch can lead to double spending, i.e. a same UTXO could be used more than once as input to transactions. Bitcoin uses a simple and interesting consensus algorithm to resolve these disputes. The branch with the highest amount of effort spent in it is chosen as the master one, i.e. the correctly branch. The amount of effort is measured by the quantity of mined blocks that it possesses. So, the more processing power employed in the branch construction, the more likely it will be the main branch. If the participants remain honest, the odds of having two branches growing equally indefinitely reduces at each new level achieved. Thus, in some point, it will reach a consensus. From this perspective, no one has incentives to mine blocks to smaller branches because they are more likely to be ignored in the future.

Figure 3.3: Blockchain main structure with branches



As can be noted, the main security foundation of this type of consensus establishment is the processing power. Therefore, it requires that more than 50% of all network processing power behaves in a honest way. Otherwise, the network could become unreliable, may remove past transactions by increasing a smaller branch further than the main one or choosing what transactions will be published. Finally, in the consensus establishment, deterministic mechanisms are used to resolve any dispute like bifurcation of the Blockchain.

3.2.5 Wallet concept

All the Blockchain assets has to be registered in its ledger, *i.e.* are in the knowledge of the network. Public keys are used to manage them, for example, to transfer values from one account to another. The wallet concept is an abstraction used to represent all the objects that are manageable by someone's public keys and, more than that, the wallet is a software or hardware that keeps track of all possession from a user and helps it manage it. A complete guide of wallets on Ethereum (see Section 3.3) is presented in [81].

In this section, we explored some of the main details of the Blockchain structure, protocols and mechanisms. Next, in Section 3.3, we present the Ethereum, a successful Blockchain-based platform.

3.3 ETHEREUM

The Ethereum [82] is an open-source and public Blockchain network that provides Turing complete distributed computing through smart contracts. Therefore, Ethereum smart contracts supports branching and looping statements, and also state variables. Thus, they can be used to solve any problem that can be solved by computers.

Interactions in the Ethereum network are done using identifiers called addresses. Each user and contract has its own address. User addresses are derived from their public key. Contract addresses are computed using its creator address and a nonce, specifically, the transactions number of its creator. Both cases uses, as addresses, the last 20B of the result of the keccak-256 (SHA3-256) over the public key (for users addresses) or the creator address and the nonce (for contracts). Although address collision is possible, it is extremely improbable as there are 2^{160} different possibilities of addresses that can be generated with this methodology. Nowadays, there are already more than 40 million distinct addresses on the network and this number is trending up [83].

Compiled and published contracts are called "Decentralized Applications" (or DApps). Contract publications or interactions are charged in a unit called Gas. This unit is a measurement of the effort of doing the user request action. Each low-level operation has its own fixed Gas cost and the amount used in an action is the sum of the expended gas in each operation. The price of Gas (called GasPrice) fluctuates over time and is given in Ethers (the Ethereum cryptocurrency). There are already more than 1000 DApps launched since 2017 and some of these contract have more than 1000 daily active users [84], i.e. users that interact with them daily.

3.4 BLOCKCHAIN-BASED ACCESS CONTROL

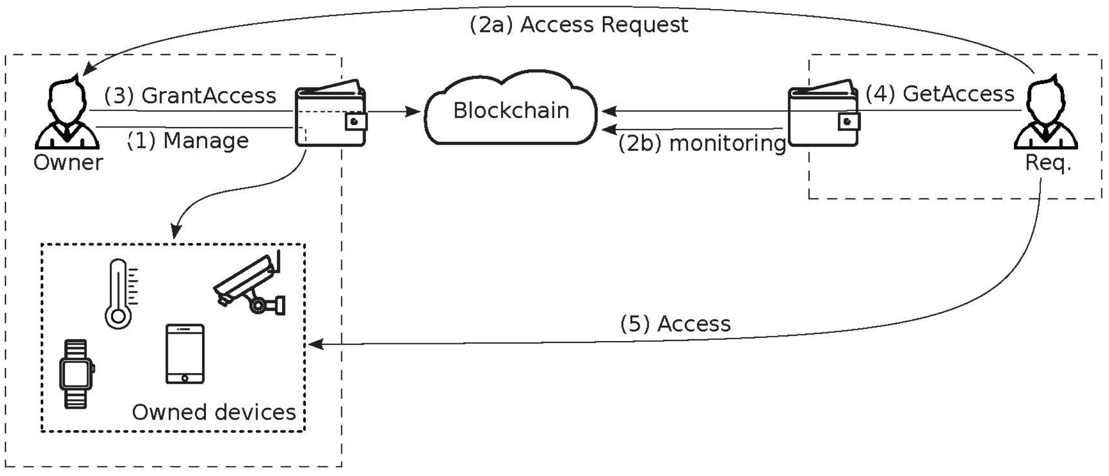
The work done in [85] mentions the benefits that could be achieved in IoT with the use of Blockchain through use cases examples. The presented use cases was the upgrading of firmware, creation of marketplace of services and energy, sharing of resources and services and supply chain. Although it does not mention the access control, there are other works, like [5], [18] and [4], that already saw its potential in this area too.

Although the division of access control approaches in different layers is not always straightforward, there were already efforts trying to standardize this classification based on the OM-AM reference model [5] (See Chapter 2). Next, we choose to present only works that are somehow related to the architecture layer because our work is more related to this layer.

Maybe, the most similar work to our proposal is the FairAccess [18, 4, 38]. It is defined as an IoT access control framework based on smart contracts, Blockchain and token-based access. The Blockchain is used as a database-like mechanism to store all the authorization transactions and smart contracts are used to trade fulfillment of access control policies for access tokens using the Blockchain. The framework specifies two types of authorization transactions: GrantAccess and GetAccess.

The resource owner grants the requester permission to access their devices through the GrantAccess transaction. This transaction has an access token and a locking script (a smart contract). The access token is delivered to the requester only after it is capable of proof the fulfillment of all access policy conditions, specified in the locking script. So, the mainly components of a GrantAccess transactions are the addresses of the resource and the requester, the locking script, and the token (encrypted with the public key of the requester). The requester proofs the fulfillment of the locking script requirements using the GetAccess transaction. This transaction enables the delivery of the token to the requester. Instead of generating an GetAccess transaction, a requester can delegate the access to another requester through a GrantAccess transaction. This delegation transaction contains the original GrantAccess, its unlocking script, the new locking script and the new requester. Furthermore, resource owners and requesters have a wallet that helps them in the authorization process and also allows the managing of devices. The Figure 3.4 shows the main interactions between the components of their framework.

Figure 3.4: FairAccess overview



Source: adapted from [4]

Except for the access grant request (step 2a) and the proper resource access (step 5), all the interactions are intermediated by wallets. Through its wallet, the owner can register and manages its owned devices (step 1), and generate GrantAccess transactions (step 3). In the mean time, the requester can use the help of its wallet to keep monitoring the Blockchain in order to find new GrantAccess that it is interested to (step 2b). Upon the receiving of an GrantAccess, the requester generates the unlocking script, a proof-of-fulfillment of all requirements established in GrantAcess, and use it in one of the two possible actions: the generation of a new GrantAccess where it delegates the authorization to another requester (not represented in Figure 3.4), or generates a GetAccess transaction (4) in order to receive the access token that allows it to use the resource.

Although, both, [18] and [4], are presenting the same framework, there are small differences on these works. For example, the grant access revoking/updating process occurs with a new grant access in [18], while [4] only mention revocation and through a token expiration time. Therefore, the first uses the Blockchain to revoke and update the authorization policies and the second uses a predefined expiration time inserted in the token. Furthermore, [4] also does not mention how a delegation of an access token can be revoked.

Perhaps, the biggest flaw of the FairAccess is imagining that the access token will be kept away from requester until it proves that it has fulfilled all the access conditions. As the token is encrypted with the public key of the requester, the only requirement to decrypt it is the private key of the requester. Therefore, this system requires full confidence in the requester as it can behave in a dishonest way and could reveal the token before fulfilling the access conditions. In our opinion, this could be worse than the centralization trust problem, since it requires trusting on several devices, systems and users. A possible simple solution to this problem is using a mechanism that makes the token valid only when the policies were fulfilled.

The advantages of our solution (presented in Chapter 4) over theirs are fivefold. First, although [4] mentions that its framework supports any access control model, they did give almost none or no directions on how to integrate other IoT models in their framework, except for token-based access. Second, using the token-based access, the subject needs to contact the owner asking him to create a new locking script and token each time the token expires for each device and requester. Third, it requires at least two blocks to be mined to the Blockchain for a new token be usable. This means that the solution is costly and could take a lot of time to grant the access. Fourth, their framework does not specify the usage of relationships information in the process of granting access to a subject. The only relationship information that could be inferred is: given two identities (public keys) it is possible to know if one is the parent of the other in the tree of generated keys. Our proposal specify an scheme that allow each identity to be linked with any other identity or groups of identities, allowing even the provision of attributes and characteristics to this links. Fifth, each single entity is independent and don't need to trust anyone other in our framework. Furthermore, nothing prevent smart contracts and their token-based approach to be also used within ours proposal, as specified by them.

Similar to FairAccess, the IoTChain [86] also uses a solution based on tokens, however, instead of using the token gathered from the Blockchain to access the resource information, it uses it to get decryption keys from key servers. In its turn, they can be used in the decryption of downloaded data from proxy server or resources servers. Also, each owner publish a smart contract that will be used for the control of his or her devices. Our proposal only requires a single contract that will be used by all owners, users and devices. Finally, its access control is very coarse grained as they do not take any contexts into consideration in authorization decisions and only accepts one token by client address.

Another framework for IoT access control using Blockchain is presented in [87]. It is composed of three main contracts types: Register Contract (RC), Access Control Contract (ACC) and Judge Contract (JC). The RC mantain a lookup table used to identify the correctly ACC based on the subject and object. The ACC define the access policies through a table policy that keeps the columns "resource", "action", "permission" and "ToLR" (Time of the Last Request). Under an authorization validation, it also query the JC to verify subect's past misbehaviors and/or penalty decisions. The JC implements a misbehavior-judging method and determines a penalty for the requester when a misbehavior report is received from the ACC. Each ACC has a table policy with the fields "resource", "action", "permission" and "ToLR" (Time of the Last Request) and only support a single tuple (subject, object). Being so, a new ACC has to be created for each new par. Therefore, depending on the use case and the network, this design choice can be laborious and costly to maintain than a pure ACL or capability approach. On the contrary, we propose that all users use a single contract and dynamically define access rules through it.

The architecture, *i.e.* the components and its interactions, presented in [88] is similar to E-ControlChain when operating in gateway mode (see Chapter 5). However, although it is not clear, their smart contract seems to only allow the registering of rules similar to those in [87], *i.e.* a limited ACL-like rules. Furthermore, their access control does not take into consideration contexts. Therefore, in our opinion, this choice reduces the usability of it and harms the user experience.

The authors of [89] proposes a ledger-based privacy-aware access control system. They use the IOTA [90] that in its turn use Tangle, a Directed Acyclic Graph (DAG) [91]-based distributed ledger. To achieve a privacy environment, all the rules are encrypted and only the resource owner (who created the rule) and its chosen policy decision point (PDP) knows the key. However, with this choice the PDP could became a single point of failure. Furthermore, this may lead to the possibility of repudiation, because the key needs to be disclosed to prove the contrary and not everyone will be willing. So, they trade resilience and transparency for privacy.

A capability-based access control developed for Hyperledger Sawtooth [92], a modular platform for distributed ledger, is presented in [93]. Their solution comprise the issuance, revocation, delegation and use of tokens. Furthermore, they used docker and made their code public through GitHub [94].

The authors of [95], [96] and [97] highlight the benefits of joining Blockchain and IoT. In addition, [95] and [97] discuss the challenges of IoT networks and its integration with the Blockchain. [97] also constructed a systematic survey, where they categorized a wide range of works in application domains, usage domains and development level. [96] discuss the benefits of Blockchain adoption in an Identity and Access Management (IAM) in an enterprise context. It also proposes an architecture similar to ours previous work [19]. However, there are also some differences. First, they only bring support to attribute-based access control (ABAC) model. Second, they do not provide alternative solutions to applications that require exchange of real-time information, requiring all the data to be mined by the Blockchain in order to be available in new decisions. Finally, they did not evaluated their architecture on a real IoT device.

MedRec [15], Bright [16] and [20] also employ Blockchain in the access control. MedRec controls the access permissions to medical record data of patients. It employs smart contract between patients, providers and third parties to grant permissions of access. Bright controls the actions performed in a video rights management system. [20] uses the Blockchain as a mechanism of data sharing. The data is stored in a off-Blockchain DHT network and only the pointer to this data is stored in the Blockchain. Differently from our architecture, these works only allow creation of ACL-like rules and don't support other information in the authorization process, like the environment context.

IBM Watson IoT [17] is a platform that, between a wide range of services, manages and controls the access to IoT devices. It also allows device data to get published into a private Blockchain in order to reduce the dependence of a central management entity in the data access. However, all the configuration of the access control is centralized.

The solution proposed in [98] uses the Blockchain to store, among other things, pairs of resource identification and URL links. The links are addresses of access policies stored on the proper resource device. An smart contract is used to retrieve the policy and check if the user has permission to access the resource. It has the advantage of updating the access policies locally, i.e. does not require the mining of each update. However, as the policies are stored off-Blockchain, they are susceptible to modification. This could lead to security problems.

The authors of [99] use the Blockchain together with an ABAC model. However, in the workflow of its platform, they describe the use of the Blockchain as a simple log database for all actions executed by their ABAC modules. Also, no evaluation in the IoT environment was made.

FedCAC [100] proposes a hierarchical architecture for access control. It is based on two main control entities types, the Policy Decision making Center (PDC) and the Coordinator. The PDC is located in the Cloud and the Coordinator is located near of resources. The PDC is a common point in the exchange of messages between Coordinators, turning it into a single point of failure. In [101], they resolve the centralization problem with a Blockchain-based architecture called BlendCAC, however, in both works, they only support capability authorization using tokens.

The authors of [102] discuss how some of the IoT challenges could be minimized with the use of Blockchain and, also, define a Blockchain-based IoT architecture divided in three layers: infrastructure, control and application. The infrastructure layer consist of networking technologies and elements, and a local Blockchain with resource constrained devices. The control layer is composed by a global Blockchain that have more powerful devices as miners. Finally, the services and user applications reside in the application layer.

In [103], the authors propose an IoT authorization architecture over the Hyperledger Fabric [104], an Blockchain implementation held by the Linux Foundation. The main contribution of their work is the enhancement of the consensus component with a genetic algorithm-based solution, called GA Kafka. With this improvement, they achieve a better transaction transfer success rate.

Although [105] solution is more about providing authentication instead of authorization, they provide a scheme that stores relationships between requester and sensors. When a requester tries to access a data from a server, it gets a challenge as a response, that needs to be accomplished in order to authenticate and receive the requested data. Each challenge defines a sequence of actions for the requester. Sensors near the client, namely Related Devices (RD), have to detect and publish the client performed actions in the Blockchain. However, their solution have a big flaw: the server select the challenge RDs using the requester's published near RDs. This could lead to the selection of fake or malicious RDs. A simple solution would be letting the source of the data define the RDs or the choosing criteria for the RDs used in the authentication to access its data.

Bubbles of Trust [106] is another solution for authentication in IoT using the Blockchain. The main idea behind Bubbles of Trust is the creation of IoT devices groups where devices only trust in devices that participate in the same groups. This groups are created by a master device that distribute tickets for other nodes, called followers, join the group. The group creation and devices joining is registered in the Blockchain.

BSeIn [107] allows requester to make request to IoT industrial devices without revealing its identity. In order to do that, they make use of a one-time key generation schema and Blockchain. In their approach all requests need to go through the permissioned Blockchain. This design decision limits the number of requirements to IoT devices to the maximum number of requirements the Blockchain miners, validators or sealers can handle.

In [108], the authors enumerate some of the challenges that can be present in proposals that adopt Blockchain as part of their solutions. In a special way, they discuss about the high cost of having all the operations performed over the Blockchain, turning them impractical. Also, they propose approaches to minimize this cost, like outsource non-essential operations of the Blockchain. One of these approaches is called as "hybrid implementation". It separates contractual operations into d-op (decentralized operations for Blockchain interactions) and c-op (centralized operations for Trusted third party - TTP - interactions). As expected, they defend that all the operations that aren't crucial to be in the Blockchain should be moved away from it and directed to the TTP. In their example, the TTP is a gateway assistant that gives insights about allowing or denying attempts of data access. In our prototype, any Ethereum node can be a "TTP" and, for example, verify the access permissions executing the smart contract locally (without any extra cost and with a similar delay of consulting a lightweight database). Of course, one should be very cautious when relying on TTPs. Furthermore, our architecture also can be classified as a "hybrid implementation" solution as it also has an off-Blockchain side channel to exchange, for example, real-time messages.

The data exchanged over a side channel can be considered more volatile than the published on Blockchains, however, some scenarios require this feature. For example, to avoid the possibility of multiple spending of an onetime voucher that can be caused by the delay of Blockchain on registering its usage. The usage of a side channel also corroborate with the so-called "Layer 2" era of the Blockchain [109]. In this era, the computation is moved off-chain in order to enable privacy, reduce costs or save computing resources.

A delegation of rights over the Blockchain is presented in [110]. A delegation can contain conditions that needs to hold in order to be valid. A right can be delegated multiple times, each time with its own conditions. When using the rights the resource verifies the events generated by the delegation and allow if the requester can prove his identity and satisfies the conditions.

A survey of Blockchain-based access control is presented [111]. They classified the works based on privacy context, application domain, access control method and Blockchain platform. They also make a brief discussion about the challenges faced by Blockchain platforms, its use and access control methods.

The authors of [112] and [113] studied security aspects of the Blockchain. [112] gathered vulnerabilities and attacks on Blockchains. [113] make a survey of Blockchain-based security aspects on IoT. They identified the main organizational formats of IoT architectures when SDN, Blockchain and Fog are employed in the IoT. besides this, they also identified many Blockchain platforms used in the IoT and made a comparison between them based on their main characteristics and particularities. Finally, they made a gathering of the literature using Blockchain to solve problems in the IoT.

Also in the security area, however focusing more in attacks over smart contracts, SmartScopy [114] is a system to identify vulnerabilities in smart contracts. In their work they make a problem formulation, gathering many known vulnerabilities and create an implementation capable of identifying many of them. Unfortunately, although we tested our smart contract with many other tools, we did not find the SmartScopy implementation to test with it too.

In use cases studies, [115] proposes the use of Blockchain and smart contracts to allow or deny the access to public infrastructure. To grant the privacy of the user, instead of the user interact direct with the access object it would interact with a smart contract in the Blockchain and it in behalf of the user would trigger the liberation of the access with anonymity, *i.e.* without informing the user who requested access. However, at least for the Ethereum, a smart contract only generates events or triggers an action in the network it has to receive a transaction. Being so, an attacker can easily see which address requested the access and link the address to the person entering the location. Thus, unless associated with something different from the presented, their solution does not deliver what it promises.

3.5 BLOCKCHAIN-BASED ACCESS CONTROL ISSUES AND CHALLENGES

A plenty of works have studying the Blockchain limitations. Some of them focus on specific platforms, like the bitcoin [116]. Others are more specific yet and focus on details of these platforms, like the pseudo anonymity in the Bitcoin [117] and attacks on Ethereum smart contracts [118]. However, there are also works that are not tied to a specific platform [119, 22]. In this section, we discuss only those more likely to affect Blockchain-based access controls.

3.5.1 Branches in Blockchain

Bifurcation (branches) on the chain of blocks can also occur in the Blockchain (Figure 3.3). This can happen by uncountable reasons, like delay of propagation, bipartition of the network because of link problems or malicious participants, or even blocks being mined at the same time. Note that in all cases, it includes a mining nodes synchronization problem.

Many Blockchain applications cannot deal with branches as they could insert many inconsistencies in its data. Furthermore, attackers can also use branches to dishonestly get improper advantages, as we will see in Section 3.5.2. Thus, normally, there are also a consensus mechanism to decide which branch has to be adopted as the “correct” one.

In traditional proof-of-work Blockchain implementation, it is common to choose the longest branch as it required more computational effort to be built. All the other branches are discarded and their data are erased from Blockchain. However, fortunately, if more than 50% of the network processing power remain honest, it is extremely improbable that attackers can control for much time the appending of blocks in the network. Also, it is risk for a miner to keep mined blocks hidden as all the miner's earnings obtained with them can be lost if the network branches grow faster than the hidden branch. Thus, miners seems to have no real advantage over hiding mined blocks or mining for an "incorrect" branch, unless its processing power is bigger than the 50% of the network.

3.5.2 Attacks

Many vulnerabilities were identified by [63]. Most of them are power-based attacks, such as the 51% attack, selfish mine attack, transaction data malleability problems. Although several solutions to address these issues have been presented, many of them are just brief idea suggestions, lacking of concrete evaluation of their effectiveness.

The double-spending problem [120, 121, 122] arises when one perform two or more exclusive transactions on the Blockchain. This problem is normally linked to the appearance of branches on the Blockchain, where these exclusive transactions are performed in different branches. Although they are valid transactions in their own branch, both are concurrent and not valid in the whole Blockchain perspective view. For example, in Bitcoin it occurs when the intersection of two or more transactions inputs is not void, i.e., someone is spending the same Bitcoin in two or more transactions. In the access control, it could occur if a subject finds a way to allow and deny at the same time an action on an object or to cheat the system in order to get extra accesses. This situation also can lead to inconsistencies in the auditing data. Some circumstances when the double-spending problem occur are the delays on propagation (race, finney and vector76 attack) and by domination of the network processing power (brute force and >50% attack, also known as 51% attack) [122].

Fortunately, provided that at least 50% of the network is honest, there are some security mechanisms that, virtually, prevents the double spending problem and some other attacks from being a really threat for real world applications. For example, the Bitcoin network only consider a new block as valid if it has only valid transactions, references the latest mined block, and has the proof-of-work. Besides this, the Bitcoin also employ a consensus mechanisms, choosing the longest chain, to decide between eventual branches that emerge on the Blockchain. This mechanisms difficult a successful double-spending attack because someone that is making the attack, normally, does not has enough processing power in order to compete with the rest of the network. Furthermore, they also grant that the older is a register in the Blockchain, the more secure it is from attacks.

However, many systems based on Blockchain requires a generous processing time to be spent until transactions are safely confirmed. This intensive resource consumption incentives the users to pool their computing power in centralized processes. This can give the control over the network to only a few, or even single, entities as occurred years ago [123]. Until today, most of the blocks are mined by only a few mining pools.

Although the proof-of-work is one of the most known methodologies, there are others proofs that could be adopted to avoid problems like the mentioned ones. For example, [80] presents four different methodologies: reaching a consensus, proof-of-stake, proof-of-activity and proof-of-publication. Each one with specific characteristics and designed for different environments, however, compared with the proof-of-work, they can be considered premature and are being adopted with great caution [124, 125, 126].

Another groups of attacks are the selfish nodes that does not collaborate with the network [127, 128]. They can be classified in two groups. In the first group are the ones that does not relay information to avoid spending processing and network resources. The other group is composed by participants that does not relay or insert substantial delay in the transmission of information that brings advantage for him. In Bitcoin, for example, a node could choose to relay only transaction with little fees and keep to itself the transactions that are more "generous" and pays more fees for the miner. Another example is the insertion of delay in the transmission of mined blocks in order to keep the adversaries participants always at least one block behind of the longest known chain, or to cause the N-confirmation double spending for example.

There is also a class of attacks that are focused on smart contracts [118, 129, 130]. They take advantages of transactions that can be replayed. Another approach is to find flaws in contracts that allows them to execute undesired actions. The majority of these problems were already in updated Blockchain platforms, but once in a while a new attack methodology gets discovered [129].

Finally, privacy violation and miners starvation attacks and problems are discussed in Sections 3.5.5 and 3.5.6, respectively. In Section 3.5.3, we discuss the problems caused by Blockchain forking. This occur when changes in the Blockchain code are accepted by only a part of the network participants, causing a segregation of them in the new and old versions and, consequently, increasing the success rate of some attacks.

3.5.3 Blockchain forking

Changes in a Blockchain-based application protocols could become quite complex since it, normally, requires all participants to agree with the proposed updates. When a change is proposed, it can result in three possibilities: (1) all participants agree and update their system; (2) all participants reject and maintain the current version of the system; (3) part of the participants agree and update and the other part does not. The worst of them is the last since it means that the community maintaining the Blockchain was divided and, therefore, this weakens the network and leaves it more susceptible to vulnerabilities like the ones discussed in Sections 3.5.2 and 3.5.6.

When participants don't reach a consensus about a change, two types of Blockchain forking can take place: hard and soft forks [119]. A soft fork is an update that brings compatibility with the older version, i.e. blocks mined by the updated node can be recognized by those that was not updated. This type of update ensures the consensus between all nodes. However, there could be also the hard forks. This fork type does not provide backward compatibility. So, for a node be capable of recognizing a block mined by updated nodes, it necessarily needs to be updated too. Therefore, hard forks are the most complicated because they can divide completely a network.

Recently, changes were proposed to the Bitcoin [131], namely SegWit2x. Between others changes, it was proposing larger blocks (with 2MB against the supported 1MB), increasing the volume of transactions that fit into each block with structural changes and removing transaction malleability [132]. Although, in mid 2017 about 90% of the hash power supported the changes, it had have problems in its adoption by the network. There were many people contrary to the changes and others even considered it as a 51% attack [133]. It was a real and very complicated example of a Blockchain fork.

3.5.4 Resource consumption

The traditional block mining approach proof-of-work works through the discovery of a nonce that causes the hash of the block be below a predefined value. This, normally, involves the waste of a huge processing power [22] and, consequently, energy power. However, this is an indispensable process to grant the security of the network.

To avoid proof-of-work original waste, new approaches are emerging. Some of them are Algorand [134], DAG-based ones [135, 90] and the Ethereum 2.0 [126]. They exchange the proof-of-work by, in short, consensus, voting and election approaches. Although they are still in a preliminary versions, they are also promising approaches.

3.5.5 Privacy

Providing privacy is a big challenge in Blockchain as everyone participating in it can see all the transactions ever made in the network. Most of the very popular Blockchain-based platforms uses a string derived from users public key as its identifier. However, although this scheme has the intention to hide the real user behind the public key, all the transactions are associated with its public keys. Therefore, only a pseudo anonymity is achieved instead of a completely one.

Meiklejohn *et al.* made a characterization of the payments in Bitcoin and successfully identified types of part of users behind the keys using transaction linking[136]. Also, although it is a complex and hard task, the users deanonymization could be possible employing profiling tools over the transaction network. Solutions that provide end-to-end communication and storage encryption are required to deal with this problem [116].

Fearing their identification, many users adopt the public key one-time usage policy. If they have to make a single operation in one of their accounts, all the funds behind it are transferred to one or multiple accounts in a single transfer operation. After it is confirmed, the public key of the transferring account is destroyed.

Although, it was not its objective, the lightning network [137] is a start for more privacy. It allows that an uncountable number of transactions between two trustless public keys be safely grouped in one single transaction. This reduces the information granularity available in the Blockchain network and, thus, increases the privacy level.

If by one side the anonymization can be seen as a good feature, by another it can complicate the confidence in transactions. Normally, there is no key authority in Blockchain-based systems [116] as it would break down the pseudo anonymity. Therefore, no one has absolute knowledge about the used keys.

3.5.6 The Blockchain as a bottleneck or as a big void space

Many Blockchain implementations requires a generous time to mine and safely confirm transactions. This prevents real time transactions and also brings questionings about the real Blockchain scalability [138] with respect to the number of transactions. In fact, if the number of transactions increase sufficiently, two things could happen: (1) only a part of the information produced is mined to the Blockchain and the other part is lost or (2) all the information is mined and the common nodes cannot deal with it or verify it. So, the Blockchain became a bottleneck in the transactions mining or creates a bottleneck in common devices that also need to analyze the information.

Trying to minimize this resource usage, the lightning network [137] specifies a hashed time-locked contract to allow any amount of payments to be executed off Blockchain and employ mechanisms that allow them to be validated in the Blockchain as one single result transaction and, therefore, reducing the number of transactions. It is specially valuable for micro payments in order to avoid fees that would represent a significant part of the payment itself. Further investigations could also bring similar solutions to reduce the amount of authorizations in Blockchain-based access controls without spoil the access, the auditing data and the context information.

Sidechains [139] are another proposal that could be used. Its is used to transfer values from one Blockchain to another. In practical terms it blocks the values in the origin Blockchain and creates this values in the destination Blockchain. However, as shown in [140], this could worse the problem because, instead of one single transaction, a transaction could require at least two transactions (one in the origin Blockchain and one in the destination Blockchain).

In fact, the amount of information produced and requested to be mined in the Blockchain is a double-edged knife. On the one hand, if it is produced to much information, the Blockchain miners or the other participants could not be capable of handling all the produced data. On the other hand, if only a small quantity is produced, it could turn in to what we call "mining starvation". The mining starvation occurs when the miners have no information to mine. This situation reduces the incoming of miners and, consequently, can make them give up mining. This also brings security problems, like the double spending (see Section 3.5.2), as dishonest miners can prefer to create fake blocks to overlapping the real ones when it is convenient for them. Therefore, methodologies and mechanisms to prevent both cases, i.e. the excess and lack of mining information, are also very important.

3.6 CONCLUSION

A lot of attention has been given to Blockchain because of its characteristics. The Blockchain is a set of tools, protocols and mechanisms that turn it into a public, decentralized, Byzantine fault-tolerant, immutable database that stores the data records in a chronological order. It was created as a database to store transactions made in cryptocurrencies avoiding centralizing entities. However, it turns out as a mechanisms with utility in a wide range of scenarios.

The Blockchain tools set range from simple, but interesting, consensus algorithms to the application of complex hash and signatures schemes. All applied to grant security and robustness even in hostile environments. Because of its characteristics it has been envisioned to become part of a lot of applications and services domains. Between the unthinkable number of them, that it was already employed, is the access control domain.

In the access control domain, the Blockchain was used in very different environments. There was solutions for protecting medical recording data and files rights management, and also as a source of distribution of IoT content and IoT access control. The IoT access control is a special environment because it encompasses a considerable quantity of challenges hardly found all together in other environments. However, only the FairAccess framework was built over this environment and in our opinion it did not explored all the potential of the Blockchain and also fail to provide aspects that we judge to be indispensable to today's IoT, like compatibility with the majority of the already adopted and used IoT access control models.

Particularly, for the access authorization process, we consider the time for a new information be part of the Blockchain as its main drawback. Of course, this is mainly dependent on the type of proof, its level of difficult, the complexity of the adopted consensus establishment approach, and the amount of produced and mined information per unit of time. However, even so, we still consider it as a mechanisms that brings more benefits than drawbacks to the IoT authorization process, specially when new Blockchain-like mechanisms are emerging, for example, Algorand [134], DAG-based ones [135, 90] and Ethereum 2.0 [126]. These mechanisms can drastically reduce costs and mining time.

The advantages of our solution (presented in Chapter 4) over theirs are fivefold. First, although [4] mentions that its framework supports any access control model, they did give almost none or no directions on how to integrate other IoT models in their framework, except for token-based access. Second, using the token-based access, the subject needs to contact the owner asking him to create a new locking script and token each time the token expires for each device and requester. Third, it requires at least two blocks to be mined to the Blockchain for a new token be usable. This means that the solution is costly and could take a lot of time to grant the access. Fourth, their framework does not specify the usage of relationships information in the process of granting access to a subject. The only relationship information that could be inferred is: given two identities (public keys), it is possible to know if one is the parent of the other in the tree of generated keys. Our proposal specify an scheme that allow each identity to be linked with any other identity or groups of identities, allowing even the provision of attributes and characteristics to this links. Fifth, each single entity is independent and don't need to trust anyone other in our framework. Furthermore, nothing prevent smart contracts and their token-based approach to be also used within ours proposal, as specified by them.

However, besides all the benefits that can be extracted from the application of Blockchain in the IoT access control, it also remains with open issues and challenges that also need to be investigated more deeply. besides this, new proposals to overcome this difficulties have to be discussed. In Chapter 4 we present our framework and an architecture derived from it. We believe that it is an step in the direction of resolution of these issues and challenges.

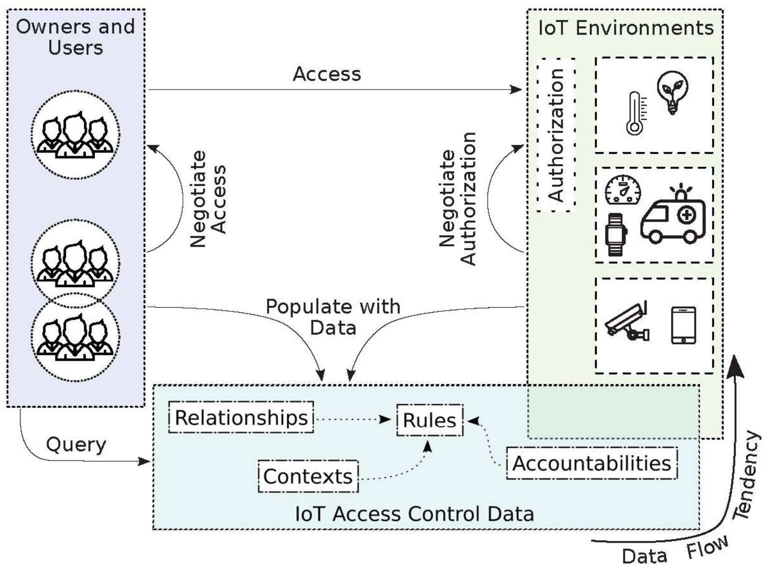
4 PROPOSED DESIGN AND ARCHITECTURE

As we saw in Chapters 2 and 3, the previous approaches are only able to cover a little part of the requirements that we identified in the Section 2.1. Some of them don't scale in many vectors, others don't take context information into consideration, don't have support to relationships or ignore the already used IoT access control models. Thus, we start to design a guidance for the IoT access control environment.

The purpose of design a guideline is to put all the IoT access control requirements together and smooth the construction of access control architectures aligned with the desired characteristics of the IoT access control approaches: decentralization, resilience, off-line working, low processor usage for authorizations, transparency, privacy, contextness and relationshipness, and compatibility with legacy models.

The Figure 4.1 summarizes the main aspects identified for a suitable access control approach. It shows the main interactions between the entities commonly involved in access control environments. We divided the scenario in three groups: "Owners and Users", "IoT Access Control Data" and "IoT Environments". The owners and users are the managers and clients of the IoT services, respectively. The users have to negotiate with owners to receive an access permission. The access authorization needs to be performed inside the IoT environments, preferably by the device being accessed, in order to avoid service interruption, for example, caused by network links disruptions.

Figure 4.1: ControlChain guidelines



If the device is not capable of executing authorization process, it needs to negotiate the delegation of this task to another nearby device it trusts. The access authorization is based on policies (called, now on, rules), defined by the owners, and other data pointed by these rules. This other data could be proofs of relationships and environment contexts (that can also include accountability information). In order to avoid access problems, the data that is important to authorization process has to be preferable held by the IoT devices themselves. So, the direction tendency of the data flow will always be toward the IoT devices. Finally, owners, users, and IoT devices are the generators of all access control data and also can query, at least, the information related to them.

The decision of including relationships and contexts in the rules was based on the flexibility and the powerful features that it brings to the authorization. With it, authorizations can change based on environment events without owners direct interference. Also, it brings flexibility to the rules, allowing, for example, the definition of a user's authorization level based on its relationships. It is also important to note that, in order to meet the decentralization requirement, the IoT Access Control Data propagation has to be based on mechanisms that allow its decentralization.

Analyzing the guidelines, it is possible to note that the Blockchain (see Chapter 3) has mechanisms that are capable of providing a good part of the IoT requirements (see Sections 2.1). For example, it is a decentralized database that eliminates the necessity of a third party control and there are sub-mechanisms that can be employed to provide resilience to data corruption or attacks trying to remove or replace a published data. However, it is important to highlight that, with the appropriate management of the published data, it is possible to "update" the Blockchain data via the publication of new registers that takes priority over old ones and turn them in historical data only, i.e., with an instruction that mark the old data as canceled and creates a new valid one. Furthermore, the way that its decentralized database works also offer the possibility of off-line authorization.

As mentioned before, another important requirement is the compatibility with the largest number of models, specially the ones adopted for the IoT. This is also possible to be achieved with the Blockchain as it does not require a strict data information or structure. In fact, it is totally flexible and can store any type of information. Therefore, virtually, any type of authorization instruction can be stored in it.

From the guidelines, i.e. based on main access control architectures and IoT requirements, ControlChain [19, 14] made its way to our world. The Figure 4.2 presents an overview of the ControlChain. It is a heavily Blockchain-based IoT access control architecture that, however, also count with an off-Blockchain side channel to communicate real-time information. Furthermore, it was designed with the following principles in mind:

- Decentralization. The expected growth of the IoT unfeasible the centralized solutions and the decentralization helps to bring scalability;

- Resilience. This means the necessity of avoid an architecture with single points of failure and use mechanisms that provide resilience to data corruption;

- Off-line working. The dominant IoT communication media type, i.e. wireless, is known to be instable and could lead to intermittent connections, so the continuous operation in a disconnected environment is a very important feature;

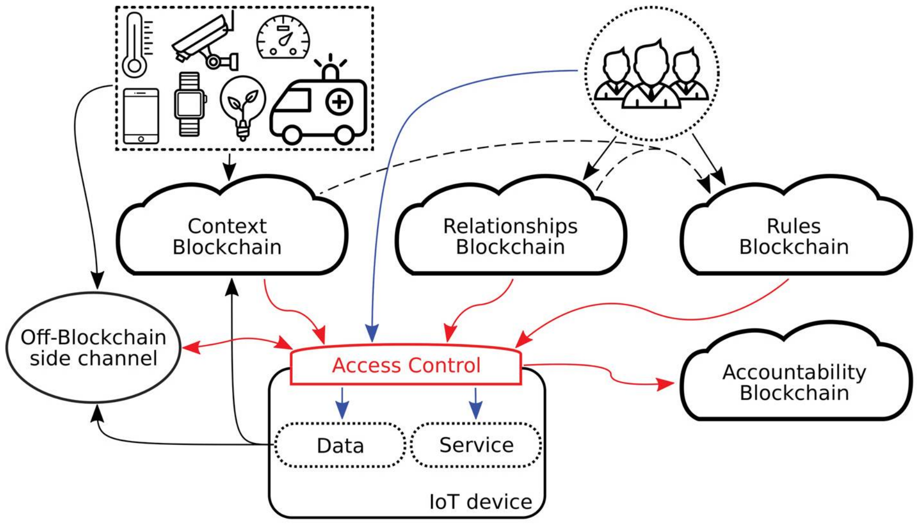
- Low processor usage for authorizations. In the IoT, some devices will be restricted in the processing power capacity and this requires a lightweight processing for the devices;

- Trust. The IoT will collect, generate, use and provide personal and sensitive informations about users and operations in general. The manipulation of these informations requires the trust of its users. A way to acquire this trust is providing an user-friendly environment with transparency, privacy;

- Context and Relationship. Allowing the use of context and relationships bring flexibility and allows better fine-grained control;

- Compatibility. There is no unique model that can achieve the requirements of all IoT environments and, thus, an architecture needs to provide mechanisms that allow the compatibility with the today's main IoT authorization models (see Section 2.3).

Figure 4.2: ControlChain entities interactions



Although nothing prevents all the information to be in one single Blockchain, for organization and to facilitate the explanation, we divided the required data of the ControlChain in four different Blockchains: Context Blockchain, Relationships Blockchain, Rules Blockchain and Accountability Blockchain. The next sections bring more information about each one of them.

4.1 CONTEXT AND ACCOUNTABILITY BLOCKCHAINS

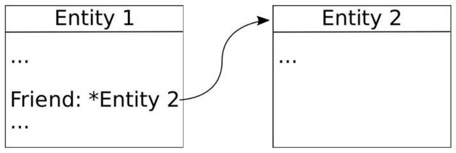
The `Context Blockchain` stores contextual information obtained from sensors, processed data and manual inputs. This information can be used in the authorization decision. For example, suppose there is an access rule with the following statement: "8k resolution videos can only be accessed when the router reports that the network traffic is low". In this situation, the access control will find the report of the router in the Context Blockchain and check its state before allowing the access. In order to allow its reference from a rule, each context information need to be identifiable by an unique identifier, namely context identifier (it could be, for example, an ordered pair "(entity, variable name)"). Also, each time a context needs to be updated, the same context identifier has to be used to ensure that previous defined rules can find the new context value.

Accountability information can also be considered as context, however, in our specification, it has a more specific information than that registered in the Context Blockchain. The `Accountability Blockchain` registers information about permissions or denies of access to object. The information required to be registered is described in the Rules Blockchain (see section 4.3). These information could be used for accountability and auditing of accesses, and for checking the sanity of the system. Furthermore, the information stored in this Blockchain could also be used like the contextual ones.

4.2 RELATIONSHIPS BLOCKCHAIN

The `Relationships Blockchain` stores the public credentials and relationships in a directional graph-similar manner, where the nodes represents the credentials and the edges represents the relationships. An entity can be a user, a device or a group of them. Each entity has a defined owner that has complete control over it. It is important to highlight that the ControlChain does not make any differentiation between the entities. A relationship is an unilateral reference to another entity with an optional set of attributes to it. For example, in the Figure 4.3, the entity 1 gives the attribute of "Friend" to the entity 2. As we will see later, the relationships and their attributes could become part of the authorization decisions.

Figure 4.3: ControlChain reference



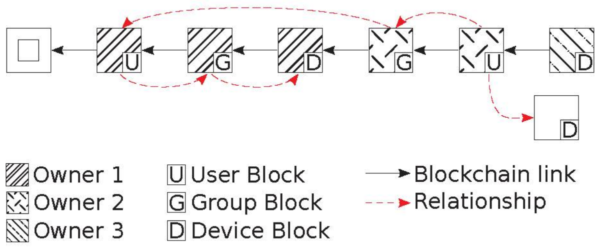
There are two possible types of relationship references: Blockchain-dependent and external. The Blockchain-dependent reference is a link created with identifications tied to the Blockchain registers (for example, a pair composed by the block's and the register's order number inside the block). The external reference is a link to off-Blockchain identification (for example, a public key). The last is a dynamic reference that always is interpreted as a pointer to the most recent update of an entity in the Blockchain, if it exists there. Note that the external reference also allows referencing Blockchain outsider entities. The choice of best type of reference is use case dependent. In Table 4.1 we give some directions to the best choice based on the use case requirements. The "+" signal shows the most indicated type for the requirement.

Table 4.1: Most indicated reference type based on the requirements of the use case.

Requirement	**Identifications Types**	
	Blockchain-dependent	External
If the most recently information inside a referenced entity need to be evaluated		+
If the information inside a referenced entity need to be evaluated as it was in the time of the authorization	+	
If references to entities outside the Blockchain are allowed		+
If it is only allowed references to entities defined on the Blockchain	+	

The Figure 4.4 shows an example of Relationships Blockchain. In this figure, each square is a block, the leftmost square is the genesis block, contiguous line arrows are the default links between the blocks, dashed line arrows are relationships and the hatch represents the block owner. It's important to remember that one block could have more than one entity and not necessarily the blocks are labeled with their type or function inside the use case, like the users, devices and groups in the figure. We choose this methodology only to simplify the explanation.

Figure 4.4: Relationship overview



In the example, the left part shows the relationship between an user, a group and a device (all owned by the Owner 1): the user references the group and the group references the device. In the middle of the example, there is an Owner 2 user with a group (for example, with the attribute “friends”) that links to the Owner 1 user. Finally, could exist entities without references or been referenced like the last entity of the Blockchain. Furthermore, if the entities identities does not depend on the Blockchain network (i.e. they are external identifications), there could be references to off-Blockchain entities, like the Owner 2 user who is referencing a device that does not belong to the Blockchain yet.

4.3 RULES BLOCKCHAIN

The `Rules Blockchain` keeps the authorization rules defined by owners to their objects or by objects to themselves. Although specifying how policy rules will be structured is a task of access control models and not of architectures, it has to give support to them, providing ways to store and retrieve all the models’ required data. We used the previous models information gathering, from section 2.3, to extract the characteristics and specificity of each one and build our architecture compatible with the largest possible set of models. The big challenge faced by this Blockchain is making it generic enough to be compatible with the big variety of access control models and mechanisms used in the IoT: RBAC [25, 26, 27, 28], ABAC [29], UCON [30], CapBAC [31, 32, 33, 34], ACL [15, 16, 20] and others [35, 36, 37]. Each one capable of fulfill different IoT scenarios requirements.

Without loss of generalization, currently, it supports three types of access control mechanisms (based on ACL, Capability and Attribute) that with some minor additions could lead to the compatibility with a lot of models. These minor additions are addition of context conditions, obligations (from UCON) and accountability information (including obligations from XACML). The context conditions are Boolean expressions that are build using context identifiers of the Context Blockchain. The obligations determine routines (for example, accept an agreement) that the subject must accomplish to get the access authorization. Finally, the third addition describe the access information that should be recorded on the Accountability Blockchain by the access control and/or PEP obligations before or after allowing or denying an access. We call these mechanisms as ACL-based rules (allows a list of subjects for each object), capability-based rules (allows a list of objects for each subject) and attribute-based rules (allow a list of subjects’ and objects’ attributes). Each rule needs to be uniquely identified (by an identifier or a set of them) in order to allow their updating or revoking using Blockchain techniques (as smart contracts). The process to transform access control models to these three generic mechanisms is done through the decoder, a mapping process capable of translating them. Section 4.4 gives more detail on how these types of rules can be used to bring compatibility to a large variety of models.

4.4 DECODER: A COMPATIBILITY TOOL SCHEME

The transformation of access control models to more generic mechanisms is illustrated in Figure 4.5. We propose the utilization of a decoder. This decoder receives the access control model and its rules and translate them to mechanisms supported in our architecture, i.e. the three suggested mechanisms-based blocks rules: ACL, Capability and Attribute. In some cases, the IoT owners are required to provide additional information. For example, suppose a rule says that only a requester with the role “manager” can have access to the management system. This

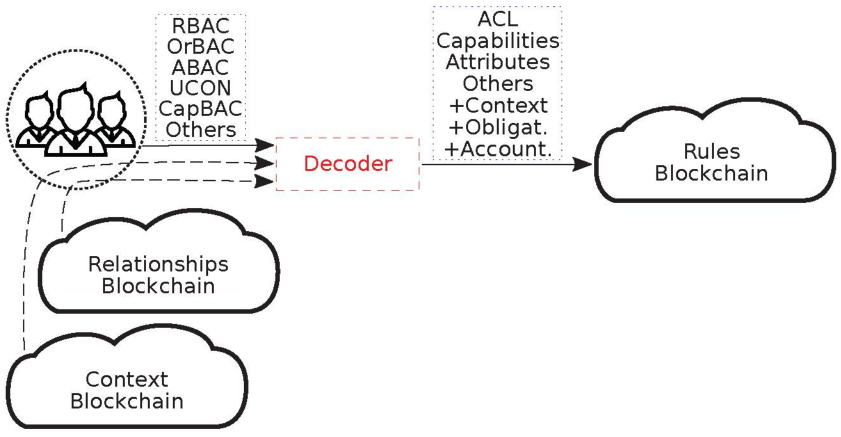
Table 4.2: Mapping of the models to the suggested rule blocks.

ControlChain	**RBAC**	**OrBAC**	**ABAC**	**UCON**	**CapBAC**	**ACL**
Entity	Req.	Req.	Req.	Req.	Req.	Req.
Entity att.	Role	Role	Req. att.	Req. att.	-	-
Entity	Res.	Res.	Res.	Res.	Res.	Res.
Att. Rule *	Rule	Rule	Rule	Auth.	-	-
- **	-	Activity	-	-	-	-
Entity att.	-	View	Res. att.	Res. att.	-	-
Entity	-	Org.	-	-	-	-
Cont. identifiers	-	Cont.	Cont.	Cont.	-	-
Cap. Rule	-	-	-	-	Rule	-
ACL Rule	-	-	-	-	-	Rule

Abbreviations: activity (activ.); attribute (att.); authorization (auth.); capability (cap.); context (cont.); organization (org.); requester (req.); resource (res.).
- The model/architecture does not directly support this element.
* When applicable, it requires the attribute authority, subject and object attribute names and expected values, allowed actions, objects with the allowed actions, contexts and obligations.
** The interpretation of activities needs to be carried by the requester and resource.

role can be seen as an attribute of the requester and, at least, one entity should be pointed as the official attribute authority for it.

Figure 4.5: Transformation of access control models to the mechanisms



The decoder can automate the adaptation of IoT authorization models using a mapping table that converts each element of the RBAC [56], OrBAC [57], ABAC [59], UCON [60] and CapBAC [61] models to those in the ControlChain. Table 4.2 summarizes all the mapping process between models and ControlChain suggested blocks. After, we discuss each model mapping.

RBAC. The adaptation of the RBAC model uses rules in the format of attribute rule block. The required steps for the adaptation are: (1) each RBAC subject and object need to be translated to entities; (2) an entity is required to be the official attribute provider for the created rules; (3) this entity has to refer all the RBAC subjects or groups of them and give an attribute compose by the pair (key,value), where the key could be, for example, "role" and the value is the names of the roles the RBAC entity participate; (4) the attribute-based rule have to contain the official attribute provider, the role attribute name and expected value, and the resource together with the allowed actions for it. Note that, in ControlChain specification, the multi-role relationship [56] needs to be carried by the requester and resource themselves.

OrBAC. The adaptation of OrBAC model also uses the attribute rule block. All the RBAC's three first steps are the same for OrBAC. The third step needs also be executed for the OrBAC objects, changing attribute from "role" to "view". Besides this, the OrBAC organization could be translated as the official attribute provider entity. At this point, all the OrBAC subject and object entities are referenced by the official attribute provider with the role and view attributes, respectively. Unfortunately, we have found no suitable conversion of the activities to actions in our suggested blocks, however, note that the actions in ControlChain can be anything and, thus, the verification if an action belongs to an activity can be carried by the subject and objects themselves. Finally, the OrBAC context should be mapped to context identifiers (see Section 4.1). Thus, the attribute rule block have to contain the official attribute provider, the role and view attribute name and the expected values, and each object has to be together with the allowed actions for it.

ABAC. Like the OrBAC, the adaptation of the ABAC model also uses the attribute rule block and uses the first three steps of the RBAC to translate the entities and establish the relationships for both, ABAC's subjects and objects. The attribute rule block should contain the requester and resource attributes, their respective attribute provider authority, the allowed actions and the context that needs to be true in order to the access be authorized.

UCON. The adaptation of the UCON model is very similar to the ABAC and all the steps required to adapt ABAC are required to the adaptation of UCON. The only exception is the obligations that does not exists in ABAC. As discussed in section 4.3, all the rule blocks have a special field for the obligations. Thus, the UCON obligations can be placed in this obligations field. Furthermore, as the UCON requires the immediately interruption of an access if the authorizations, obligations and conditions are not satisfied at any time, devices need to keep monitoring the Blockchains, searching for changes in the parameters and validating the access for each change.

Finally, note the presented models till here require that entities establish relationships on the Relationships Blockchain. This is required because all these mappings use attributes and, in ControlChain, attributes only can be given by established relationships. The CapBAC and ACL mappings don't need attributes, so it is not required that the entities involved in their rules be in the Relationships Blockchain.

CapBAC. Originally, the capabilities of the CapBAC model is completely transferable. However, we are proposing to use a public Blockchain and the storage of completely transferable capabilities in it could become a security problem. To solve this problem, we add the subject s to the capability pair, turning the tuple (o,a) in to the triple (s,o,a). Therefore, with this rule, only the subject s can access objects o to execute the guaranteed respective actions a on them. Finally, this triple can be adapted to our architecture by: converting s and o to entities and defining a in the capability rule block for each object that s can access. The result is a rule that contains the subject and a list of objects with the respective allowed actions.

ACL. Although ACL is considered as a mechanism [5], we discuss the mapping of the ACL because it can be used as a mechanisms to a wide variety of models. So, all the models that can be mapped to ACL mechanism also can be mapped to the ControlChain rule blocks, increasing the compatibility. The mapping of ACL is similar to CapBAC: ACL subjects and object are converted to entities, and defining, in the ACL rule block, all resources, with their respective requester's allowed actions. The result is a rule that contains the object and a list of subject with the respective allowed actions.

4.5 THEORETICAL ANALYSIS

The theoretical analysis of ControlChain is divided in two parts. First, we compare ControlChain with other architectures and, then, we discuss the ControlChain's usage viability in scenarios with limited resource devices.

4.5.1 Architectures comparison

Table 4.3 shows the comparison of ControlChain with other architectures, specifically XACML, OAuth, UMA and FairAccess based on IoT requirements, dependence of third parties, time and effort to generate new authorization or update the already existent ones and to be authorized. This architectures were chosen based on two reasons: the first three are commonly used in IoT access control proposals; and the last has gained prominence between researches of the area. A "+" sign means a good evaluation, a "-" sign means a bad evaluation and "+-" sign means an average evaluation. Next, we explain the features and evaluations.

Table 4.3: Architectures comparison.

Feature Architecure	**XACML**	**OAuth**	**UMA**	**FairAccess**	**ControlChain**
Scalability	-	-	-	+-	+-
Low object overload	+	+	+	+	+
Transparency	-	-	-	+-	+
Fault tolerant	-	-	-	+-	+
Privacy-friendly	+	+	+	+-	+-
Delegation capability	+	-	-	+	+
Context-aware	+	-	-	-	+
Fine grained	+	-	-	+	+
Integrated relationship	-	-	-	-	+
Compatibility	+	-	-	-	+
No third-parties	-	-	-	+	+
New/Update authorization	+	+	+	-	- * **
Get authorization	+	+	+	- *	+

* Dependent of the type of proof and dissemination speed of blocks.
** If using the side channel for publication of rules, it would be a plus sign.

Scalability. FairAccess and ControlChain use Blockchain as its underline technology. Although it grants scalability in the number of nodes participating in the network, it does not scale in the same manner with the number of published data. Therefore, FairAccess and ControlChain received an average evaluation, while the centralized architectures (XACML, OAUTH and UMA) receives negative evaluation.

Low object overload. This criterion evaluates how much the resource is overloaded by the authorization process. In the centralized architectures, all the authorization process is externalized to a powerful entity by design. However, nothing prevents the FairAccess and the ControlChain to externalize their authorization process too. In fact, the ControlChain can facilitate the automation of this process when it is necessary. A device could search, in the Relationships Blockchain, for other confident devices, for example from same owner, and ask their support in the authorization process. Therefore, all the architectures were positively evaluated.

Transparency. The transparency evaluation tries to identify the level of transparency in the acceptance and denial of access. In centralized architectures, the central server defines to grant or deny access using information in its possession. Therefore, all centralized architectures receive a negative evaluation. FairAccess define a contract that has to be fulfilled before the access is authorized, however, its says nothing about who will grant the proof of fulfillment of the contract. Thus, we give an average evaluation as it could require a third party in the process. In the ControlChain, all the processing is done in the Blockchain and anyone can verify the algorithms and data used in the access granting.

Fault tolerant. This criterion evaluates the impact caused by failures on devices or communication links. Naturally, the decentralized ones are more fault tolerant than the centralized ones. However, the FairAccess requires that the resource owner publishes a token every time an access is requested or need to be renewed. If the owner is not available to provide it, the access cannot be established. This gives ControlChain a slight better evaluation.

Privacy-friendly. Although all access control provide privacy to the data and services held, centralized architectures can keep rules and information, used in the access attempt evaluation, like rules and context, hidden from outside viewers. Therefore they are more privacy-friendly than the Blockchain approaches that keep this information public.

Delegation capability. XACML, FairAccess and ControlChain has at least one level of delegation, thus they received a positive evaluation. OAuth and UMA, although are used to authorized application to execute action in behalf of the user (working as an admission control), do not have a delegation functionality by default.

Context-aware. As only XACML and ControlChain define policies capable of dealing with context, only them receive a positive evaluation.

Fine-grained. The fine-grained evaluation tries to measure how flexible the access control architecture is. The OAuth and UMA are almost statically, thus they receive a negative evaluation. In FairAccess the owner can specify a contract that has to be fulfilled by the requester in order to receive the access authorization. However, they do not present the structure of the contract, making it hard to analyze its flexible, so we gave it a neutral evaluation. On the other hand, XACML and ControlChain handle attributes and context, and thus they receive a positive evaluation.

Integrated relationship. The ControlChain is the only architecture designed to directly allow relationships on rules. Therefore, it was the only one positively evaluated.

Compatibility. This criterion evaluates the compatibility of the architectures with the plenty of models currently employed in the IoT. XACML and ControlChain are compatible with a considerable quantity of models and then they received a positive evaluation. Indirectly, OAuth, UMA and FairAccess can operate with almost any model, however, directly, they only operate with access tokens and, thus, they were negatively evaluated.

No third parties. The dependence on third parties could prevent the detection of censorship, frauds and interference. All the centralized architectures depend on third parties and, thus, only FairAccess and ControlChain receives positive evaluations.

New/Update Authorization. This criterion evaluates the latency to make or change an authorization. The centralized architectures XACML, OAuth and UMA have low latency in these activities because the update of their database is straightforward. Thus, they received a positive evaluation. For FairAccess and ControlChain, the complete analysis of this criterion is dependent of the blocks dissemination speed and, mainly, the type of proof used in the blocks mining. As the most common and known type is the proof-of-work and this type of proof imposes a considerable latency, we give a negative evaluation. It is important to note that, the FairAccess does not specify a way to modify a contract already published on the Blockchain. ControlChain specification defines a way to update the information (see Sections 4.2, 4.1 and 4.3). Also, if the ControlChain side channel is used to publish rules at the same time it goes to be published on Blockchain, the time for share a rule or information could be very similar to the centralized approaches, and thus it would receive a positive evaluation.

Get authorization. This criterion evaluates the latency to get an authorization. XACML, OAuth, UMA and ControlChain have almost real-time authorizations. The FairAccess has a bigger latency because it requires that the requester publish a fulfillment proof of the contract requirements. This requires, at least, one additional block to be mined in order to the access be granted. Thus, it is the only one that receives negative evaluation.

User-friendly. We chose not evaluate this criterion as we do not have a survey with real users.

4.5.2 Viability with limited resources devices

One important factor about an access control approach for the IoT is its viability in scenarios with limited resource devices. In this section, we discuss how the main technology used by ControlChain, the Blockchain, and the evaluation of rules can be compatible with IoT requirements.

Growing of the Blockchain. One of the main concerns when talking about the Blockchain is its growth. The size of an entire Blockchain could be a problem to devices with low storage space. However, devices with very limited resource does not need to store the full Blockchain. For example, the storage could be performed with some replication factor [141]. Besides this, they could also filter all the non-important information and store only the ones it judges to be important. For example, a device could store only rules related to it and both, contexts and relationships that are related to these stored rules.

Speed of new registers. With the arise of many new registers in a short period of time, devices with less resources could not be capable of keep up with the updates. However, the number of registers in a block can be limited and the speed of new blocks can be adjusted by, for example, changing the difficult of the proof-of-work imposed to miners. Besides this, Blockchains and Sidechains could be used for depending on the requirements of the system, avoiding partially the burden caused on the Blockchain.

Finally, as mentioned in Section 4.5.1, restricted devices could also find support in other devices, for example, the ones with the same owner in the Relationships Blockchain.

4.6 CONCLUSION

This chapter presented the ControlChain, an architecture for access control in the IoT. The ControlChain is based on Blockchain and provides protection ways against improper accesses of resources in a decentralized fashion. It also supports the establishment of relationships, context, obligations and accountability in order to bring more flexibility, to allow more powerful controls over the access and to make possible verifying important information about the access control. It is heavily based on Blockchain technology, however it also has an off-Blockchain channel that is used to communicate real-time data. We also presented the Decoder, a mechanism that specifies a way to turn the ControlChain compatible with a wide range of already adopted IoT access control models. Finally, we constructed a comparison between other architectures and our proposal. It showed the benefits of ControlChain over the others in aspects like transparency, fault tolerance, delegation capability, context-awareness, compatibility and others. However, unfortunately, in two aspects it has competitors that have advantage over it, specifically, privacy-friendly and the time to generate new or update authorizations. Finally, we also briefly discussed about some two of the main concerns surrounding Blockchain-based systems in environments with resource-restricted devices.

5 ARCHITECTURE IMPLEMENTATION, TESTS AND RESULTS

The E-ControlChain [14] is a ControlChain's [19] proof-of-concept architecture implementation. It was implemented with the objective of checking ControlChain's viability for usage in the IoT. The E-ControlChain is a smart contract designed to run over the Ethereum platform and developed using the Solidity [142], a widely known and used smart contract language for the Ethereum Blockchain network. In order to make ControlChain compatible with it, we had to deal with Ethereum particularities, for example, all the E-ControlChain interactions are through function calls. Also, as it was implemented as a proof-of-concept, not every aspect and detail of the ControlChain was implemented (more details in Section 5.3). Table 5.1 shows the ControlChain properties present in the current version of E-ControlChain. Some of the not-implemented properties depend on external interactions or dynamic actions, like "obligations", and requires more investigation on how to integrate the solutions to Ethereum contracts. Others are apparently impracticable to implement over the Ethereum contracts because of its purposeful design limitations, like the strict limited number of variables in each function.

Table 5.1: Current E-ControlChain development status.

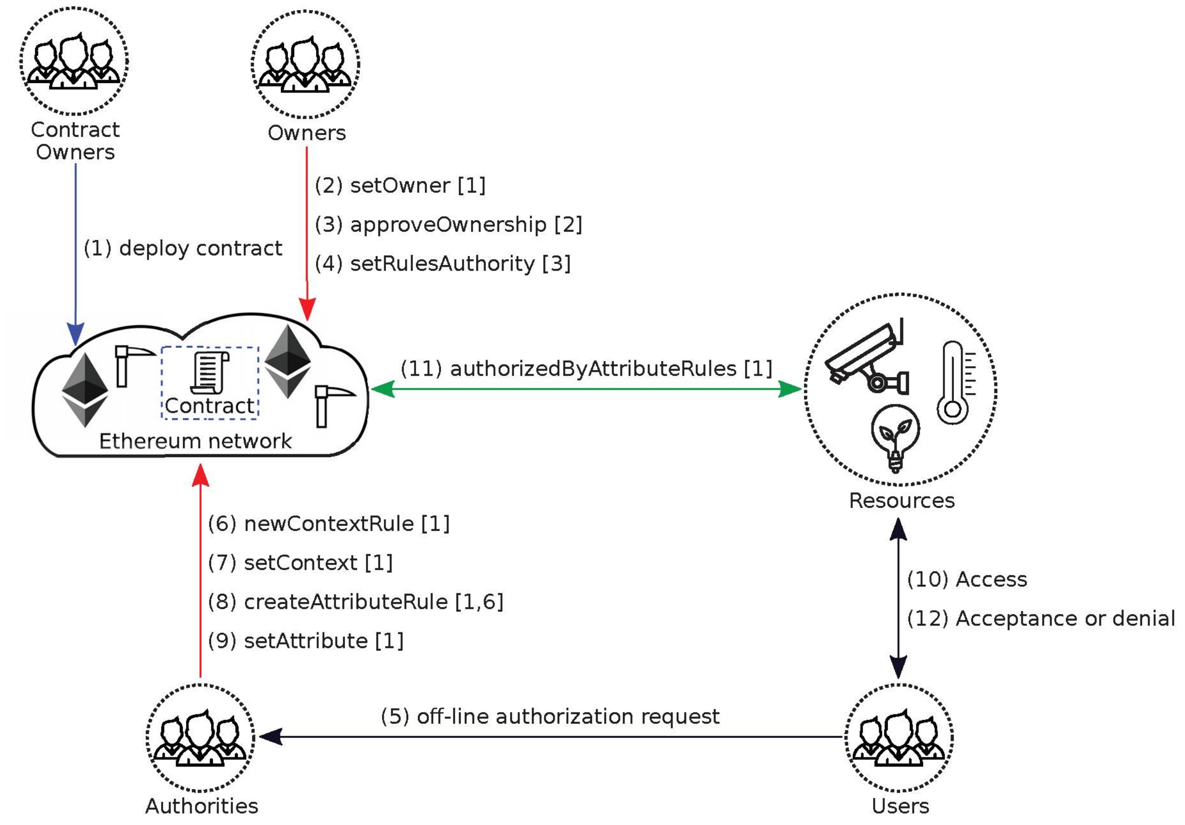
ControlChain properties	Present in E-ControlChain
Context	Yes
Relationship	Yes, through attributes and/or context
Accountability	Yes, through context publishing
Obligations	Yes, through context information
Side channel	No, but it can be accomplished using XMPP or MQTT
Capability-based rule	Yes
ACL-based rule	Yes
Attribute-based rule	Yes
Multi-attribute-based rule	No, at most one per entity
Delegation	Yes
Rules w/ contexts	Yes, but only allows the use of "and"for multiple contexts
Rules w/ relationships	Yes, through attributes and/or context
Possibility of using a gateway	Yes

In its core, the E-ControlChain is composed by four smart contracts. The first one, called `BasicAndCommonControl`, defines variables and functions common to all other contracts. The other three are specialized in each type of authorization mechanism that are defined by the ControlChain architecture, i.e., authorization based on attributes (`AttributeControl`), capabilities (`CapabilityControl`) and ACL (`AclControl`). For brevity and as it is the most complex of them, we will focus only the essential contract parts for the attribute-based authorization, i.e. both `BasicAndCommonControl` and `AttributeControl` contracts. However, the full code is available on Appendix B.

Figure 5.1 presents the main interactions of the attribute authorization type (red and green arrows), together with the contract deployment (blue arrow) and the expected off-line and direct interactions (black arrows). In the figure, each interaction has a number, however, except by the prerequisites (defined in the box brackets), there are no strictly execution order. Thus, one can invoke (7) setContext before it invokes (6) newContextRule because they are independent functions. However it is impossible to invoke (4) setRulesAuthority before (3) approveOwnership because the former depends on the posterior.

Before being used, E-ControlChain needs to be deployed. The deployment is made sending its compiled code to the Ethereum, i.e., by `(1) deploy contract`. The deployment

Figure 5.1: E-ControlChain interactions



has to be done only one time and any other E-ControlChain function call from any user can be directed to the same deployed contract. There are eight fundamental function calls to provide the ControlChain attribute-based access control: (2) setOwner, (3) approveOwnership, (4) setRulesAuthority, (6) newContextRule, (7) setContext, (8) createAttributeRule, (9) setAttribute and (11) authorizedByAttributeRules. Keep in mind that, some functions dedicated to "update" or "remove" published contents were hidden to simplify the explanation, for example, a function called "deleteAttributeRule" that is used to invalidate an access rule. Furthermore, some of the presented functions, like (7) setContext if using a previous used context identifier, can be used to update the context values. Thus, the update of some information does not require a separated function.

The `(2) setOwner` and `(3) approveOwnership` functions define an entity ownership and approve it. If it already has an owner, only the defined owner can define a new one. However, if there is not a defined owner, anyone can became the owner of the address. Letting the call permission of the function "setOwner" be more flexible could ease the recovery of a device if the owner lost his or her keys, however, in the other hand, the effectiveness of thefts can increase. Also, in order to avoid malicious entities from pretending to belong to an entity they do not belong to, only the defined new owner can approve his ownership. After the ownership is approved, the owner can delegate the access control over the resource to another entity (called rules authority) using `(4) setRulesAuthority`.

The rules authority is responsible for creating the access control rules for its delegated resources. It creates the attribute-based rules using the `(8) createAttributeRule` function. Each rule defines the required attributes for the resource and requester, and its respective attribute authorities, *i.e.* the entity that has to provide the attribute. besides this, it also defines the allowed actions and a list of already created context rules that have to hold in order to the access be granted. Thus, before creating it all the used context rules has to be already created.

A new context rule can be created with the `(6) newContextRule` function. Each context rule is an ordered quadruple `(source, identifier, comparator, value)`. It defines that the `source` is the provider of the context identified by `identifier` and that, in order to allow access, the comparison, using the `comparator`, of the current context value with the `value` has to be true. Currently, the `comparator` can be any common relational operator: <, <=, ==, ! =, >= or >, represented by integers ranging from 0 to 5, respectively. Note that a tuple `(source, identifier)` is a context identifier. The context rule always uses the more recently published context and only the `source` itself can publish/update its contexts. A context is published with the `(7) setContext` function and requires the context identifier and a value for it.

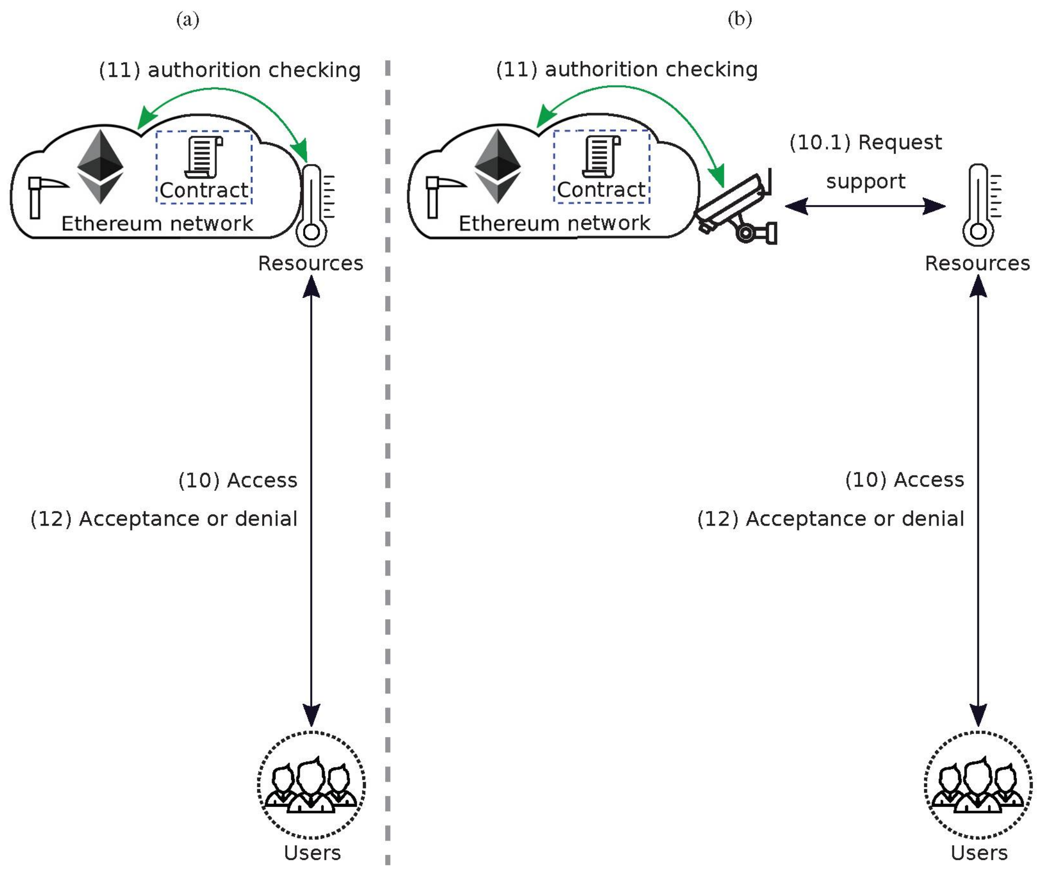
Any entity can give attributes to any other entity using the `(9) setAttribute` function. The given attributes are isolated by authority, *i.e.* the ones given by one authority does not get mixed out with the attributes given by another authority. Note that all the rules that uses information from authorities have to provide the identity of the authority from which the information will be pulled out.

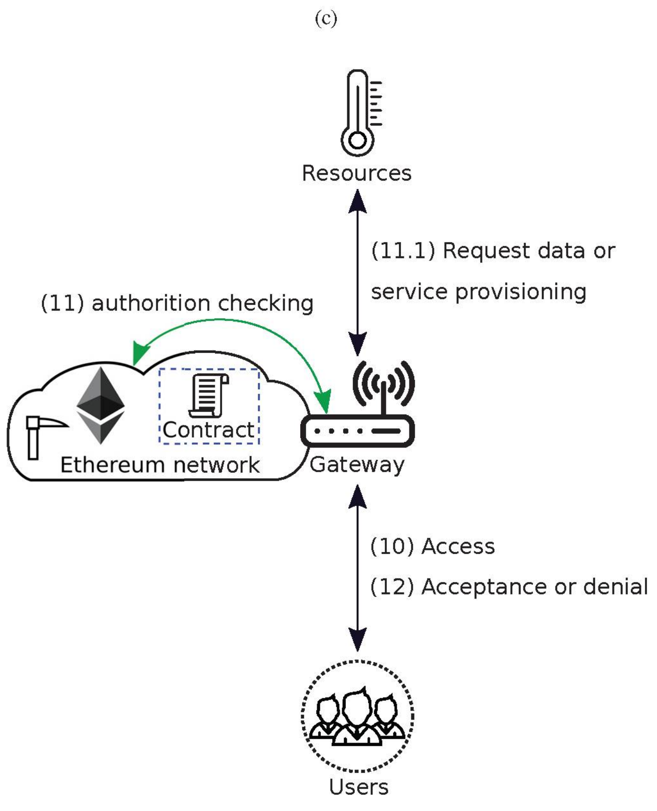
At any time, a resource can verify if a requester can have access to its resource. It can check this information with `(11) authorizedByAttributeRules` function. It only needs to provide the requester and the resource identifiers. As this function does not change the state of the contract it can be executed locally on the Ethereum node, generating no extra cost because its call does not get mined. In addition, as it is intended to be a local query, it also has a similar response time as a common lightweight database query.

In order to avoid centralization, the E-ControlChain advocates the direct `(10) access` to the devices holding the resources whenever it is possible. However, (11) can be called not only by the resource device itself, but also by a gateway or any other device. So, a resource device can choose one or more devices from its confidence circle and ask for their support. Figure 5.2 shows three different approaches of authorization checking. Up receiving an access request, the device can: (a) check the authorization itself; (b) ask for trusted device's help; (c) use a gateway or router that filters all unauthorized access attempts. After the checking, it replies with an `(12) Acceptance or denial`, i.e. the data/service or an unauthorized message. It is important to highlight that the exactly way all the off-line requests, like `(5) Off-line authorization request`, (10) and (11), occurs are out of scope for the E-ControlChain and has to be defined by a protocol defined between the Owners, Users and IoT devices.

Assumptions Our assumptions are: (1) For each resource that requires a different access control policy, there is a different Ethereum network address and, therefore, a public and private key; (2) Only the resource device know the private keys for its resources; (3) If the resource device does not have the minimum power to run a Ethereum client, it has a trusted third party node who can run the client for it; (4) The Ethereum client can synchronize correctly with the network, *i.e.* it is connected to at least one honest participant that is not suffering from attacks; (5) A resource device can verify and extract the address of a requester or has a trusted third party that can do it; (6) A requester know or has a way to obtain the address of a device. As can be seen, all the assumptions are reasonable taking into account the Blockchain mode of operation, its security mechanisms, Ethereum characteristics, and today's existing technology.

Figure 5.2: E-ControlChain Authorization Approaches: (I) direct access; (II) support device access; (III) gateway access





5.1 MATHEMATICAL MODELING

In this section we construct the mathematical modeling of E-ControlChain. We start from the basic structures and go up until reaching the authorities, resources and context. After, we show how this data is updated and how the authorization is verified through it.

Let:

- $\mathbb{Z}$ be the set of integers;
- $\mathscr{S}$ be the set of UTF-8 strings;
- $\mathscr{B}$ be the set of boolean values, *i.e.* "true" and "false";
- $\mathscr{A}$ be the set of Ethereum addresses;
- ACT be the set of actions, *e.g.* read, write and execute;
- and $COMP$ be the set of comparators, *e.g.* >, = and !=.

Then, a context rule is composed by an address $a \in \mathscr{A}$ that will be the source responsible for updating the context, a string $s \in \mathscr{S}$ that uniquely identifies it in the source scope, a comparator $comp \in COMP$ that will be used to evaluate the context and an integer that serve as parameter in the comparison. Thus, we define CR, the set of all possible context rules, as the Cartesian product of the set of $\mathscr{A}$, $\mathscr{S}$, $COMP$ and $\mathbb{Z}$ as shown in (I):

$$CR = \mathscr{A} \times \mathscr{S} \times COMP \times \mathbb{Z} \tag{I}$$

With CR defined, we can define the rules sets, but first we define the capability and ACL set lists. Both are composed by an address $a_1 \in \mathscr{A}$, a subset of context rules $CR' \in \mathscr{P}(CR)$ and a subset of actions $ACT' \in \mathscr{P}(ACT)$. The difference is that the addresses in capabilities correspond to resources, while in ACL to users. Thus, we define CAP and ACL, the sets of all possible capabilities and ACL lists, as an equal Cartesian product of $\mathscr{A}$, $\mathscr{P}(CR)$ and $\mathscr{P}(ACT)$ as shown in (II) and (IV), respectively. Also, to identify to whom they belong to, we need one more address $a_2 \in \mathscr{A}$ that correspond to the user in capabilities (defining to which user the capability belongs) and to resources in ACL (defining to which resource the ACL belongs) address when turning them into rules. This requirement results in a Cartesian product of the $\mathscr{A}$ with CAP, and $\mathscr{A}$ with ACL, originating $CAPRULES$ (III) and $ACLRULES$ (V), respectively. Finally, the attribute-based rules have seven variables. The second and fourth, $s_1, s2 \in \mathscr{S}$, are the attributes required for the resource and user, respectively. The first and third, $a_1, a_2 \in \mathscr{A}$, are the addresses of who needs to provide the attributes to the resource and requester, respectively. The fifth parameter, $b \in \mathscr{B}$ is a Boolean that define if addresses should inherit the attributes of their own owners. The sixth parameter, $CR' \in \mathscr{P}(CR)$, is a subset of CR set. The seventh parameter, $ACT' \in \mathscr{P}(ACT)$, is a subset of ACT set. Thus, we define $ATTRRULES$, the set of all possible attribute rules, as the Cartesian product of the set of $\mathscr{A}$, $\mathscr{S}$, $\mathscr{A}$, $\mathscr{S}$, $\mathscr{B}$, $\mathscr{P}(CR)$ and $\mathscr{P}(ACT)$ as shown in (VI).

$$CAP = \mathscr{A} \times \mathscr{P}(CR) \times \mathscr{P}(ACT) \tag{II}$$
$$CAPRULES = \mathscr{A} \times CAP \tag{III}$$
$$ACL = \mathscr{A} \times \mathscr{P}(CR) \times \mathscr{P}(ACT) \tag{IV}$$
$$ACLRULES = \mathscr{A} \times ACL \tag{V}$$
$$ATTRULES = \mathscr{A} \times \mathscr{S} \times \mathscr{A} \times \mathscr{S} \times \mathscr{B} \times \mathscr{P}(CR) \times \mathscr{P}(ACT) \tag{VI}$$

To define authorities, we have to define the rules and attributes that it holds. Each rules authority can choose between what rules defined above they will make its subordinate devices obey. Thus, the rules used by an authority $AUTHRULES$ are a triple composed by

$CAPRULES' \in \mathscr{P}(CAPRULES)$, $ACLRULES' \in \mathscr{P}(ACLRULES)$ and $ATTRULES' \in \mathscr{P}(ATTRULES)$ as shown in (VII). Attributes are strings that are associated to addresses. They are used to grant accesses in attribute-based rules. A set of all attributes, $ATTRS$, is composed by the Cartesian product of all addresses $a \in \mathscr{A}$ with all strings $s \in \mathscr{S}$ as shown in (VIII). Each attribute authority can choose, between all attributes, which ones they will declare. Thus, each authority chooses a subgroup $ATTRS' \in \mathscr{P}(ATTRS)$ to be its attributes, $AUTHATTRS$, as shown in (IX). Finally, the set $AUTHS$ holds all the information from authorities, *i.e.* rules and attributes. This information is indexed by the address of the authority. Thus, we define the set of all authorities, $AUTHS$, as the union of the triples composed by the authority address $a \in \mathscr{A}$, the attributes and rules chosen, $AUTHATTRS$ and $AUTHRULES$, respectively. Note that all addresses are authorities but they can choose not to create rules or give attributes.

$AUTHRULES = (CAPRULES', ACLRULES', ATTRULES') \mid$ (VII)
$CAPRULES' \in \mathscr{P}(CAPRULES)$, $ACLRULES' \in$
$\mathscr{P}(ACLRULES)$, $ATTRULES' \in \mathscr{P}(ATTRULES)$

$ATTRS = \mathscr{A} \times \mathscr{S}$ (VIII)

$AUTHATTRS = ATTRS' \mid ATTRS' \in \mathscr{P}(ATTRS)$ (IX)

$AUTHS = \bigcup(\forall a \in \mathscr{A}(a, AUTHATTRS_a, AUTHRULES_a))$ (X)

A resource is represented by its address, the address of its owner, a flag of approved ownership and an address of the authority. Thus, we define the set of all resources, RES, as the union of the quadruples composed by the resource address a_1, the address of its owner $a_2 \in \mathscr{A}$, a flag of ownership approving $b \in \mathscr{B}$ and an address of the authority $a_3 \in \mathscr{A}$ for each $a_1 \in \mathscr{A}$ as showed in (XI). Note that all addresses are resources but its owner can choose not to define authorities for it. Furthermore, the definition of an owner for an address is not mandatory, although it is the first recommended action when a new address is created (see Section 5.4.1). Note also that an address can be the owner of itself.

$RES = \bigcup(\forall a_1 \in \mathscr{A}(a_1, a_2, b, a_3) \mid a_2, a_3 \in \mathscr{A}, b \in \mathscr{B})$ (XI)

A context creates a representation of the current environment. To have a meaning it is always bounded to an address. They are used inside authorization rules. We define the set of all possible contexts, $CONTEXTS$, as a the union of the triples composed by an address a, a string s and a value $v \in \mathbb{Z}$ for each $a \in \mathscr{A}$ and $sin\mathscr{S}$.

$CONTEXTS = \bigcup(\forall a \in \mathscr{A}, \forall s \in \mathscr{S}(a, s, v) \mid v \in \mathbb{Z})$ (XII)

5.1.1 Updating the data

To avoid contradiction in the information held by the mathematical model, operations of updating information are composed by an intersection that removes the old information and a union that adds the new one. Furthermore, in order to avoid undesired updates, all of them are restricted by who can perform it. For example, authorities can freely update its given attributes and its rules, but cannot change the ones from others as shown in (XIII).

$AUTHS = AUTHS \bigcap (a, AUTHATTRS_{old}, AUTHRULES_{old}) \bigcup$ (XIII)
$(a, AUTHATTRS_{new}, AUTHRULES_{new})$ IFF the entity updating the data is a,
otherwise $AUTHS = AUTHS$

On the other hand, resources cannot freely change its information. Only owners can execute the operation. Also, the model also impose some restrictions to avoid undesirable behaviors. For example, if the owner is changing the ownership of one of its resources, the flag of approved ownership has to be false and the authority rule be undefined as shown in (XIV). besides this, only the current owner can change the flag of approved ownership to true or define a new rule authority for the resource (after approving the ownership) as shown in (XV) and (XVI), respectively.

$RES = RES \bigcap (a_1, a_{2old}, b, a_{3old}) \bigcup (a_1, a_{2new}, false, 0x0...0)$ (XIV)
$\mid (a_1, a_{2old}, b, a_{3old}) \in RES,\ a_{2new} \in \mathscr{A}$ IFF the entity updating the data is a_{2old} or a_{2old} is $0x0...0$, otherwise $RES = RES$

$RES = RES \bigcap (a_1, a_2, b, a_3) \bigcup (a_1, a_2, true, a_3) \mid (a_1, a_2, b, a_3) \in RES$ IFF the entity updating the data is a_2, otherwise $RES = RES$ (XV)

$RES = RES \bigcap (a_1, a_2, b, a_{3old}) \bigcup (a_1, a_2, b, a_{3new}) \mid (a_1, a_2, b, a_{3old}) \in RES,\ b = true,\ a_{3new} \in \mathscr{A}$ IFF the entity updating the data is a_2, otherwise $RES = RES$ (XVI)

As in rules and attributes update, only the context provider itself that can update the values of a context held by it as shown in (XVII).

$CONTS = CONTS \bigcap (a, s, v_{old}) \bigcup (a, s, v_{new}) \mid (a, s, v_{old}) \in CONTS,\ v_{new} \in \mathbb{Z}$ IFF the entity updating the data is a, otherwise $CONTS = CONTS$ (XVII)

5.1.2 Authorization check

Let:

- a_{req} be the address of the requester;
- a_{res} be the address if the resource;
- act_{req} be the action the requester is trying to perform on the resource.

So, from (XI), (XIV) and (XVI), there is a rules authority a_r that manages the access to the resource on behalf of the owner:

$\exists\ a_r \mid (a_{res}, a_2, b, a_r) \in RES$ (XVIII)

From (VII) and (X), there are three sets of rules, each set for one of the types of rules, *i.e.* capability ($CAPRULES_r$), ACL ($ACLRULES_r$) and attribute ($ATTRULES_r$):

$\exists\ (a_r, AUTHATTRS_r, AUTHRULES_r) \in AUTHS$ (XIX)

$AUTHRULES_r = (CAPRULES_r,\ ACLRULES_r,\ ATTRULES_r)$ (XX)

Each of this sets need to be evaluated before denying an access. If anyone of them has a valid rule, them the return is true. So,

$$authorized = \begin{cases} \text{true, if } \exists\ caprule_i \in CAPRULES_r \mid caprule_i \text{ is valid or} \\ \qquad \exists\ aclrule_i \in ACLRULES_r \mid aclrule_i \text{ is valid or} \\ \qquad \exists\ attrule_i \in ATTRULES_r \mid attrule_i \text{ is valid,} \\ \text{false, otherwise} \end{cases} \qquad \text{(XXI)}$$

Next we show how each one of the three types are evaluated.

CAPRULES. From (III), each $caprule_i \in CAPRULES_r$ is a tuple (a_j, CAP_j) and

$\exists\ (a_{req}, CAP_{req}) \in CAPRULES \mid CAP_{req}$ has all the capability-based authorizations granted to req by the rules authority of resource res (XXII)

From (II), each $cap_i \in CAP_{req}$ is a triple (a_j, cr_j, act_j) and

$\exists\ (a_{res}, CR_{res}, ACT_{res}) \in cap_{req}$ (XXIII)

Therefore, for CAPRULES,

$$authorized = \begin{cases} \text{true, if } act_{req} \in ACT_{res} \text{ and } \forall cr_i \in CR_{res}(cr_i \text{ is valid}) \\ \text{false, otherwise} \end{cases} \qquad \text{(XXIV)}$$

At the end of the session we show how to evaluate a context rule cr_i.

ACLRULES. The evaluation of ACLRULES are very similar to those in CAPRULES, except that the addresses of the requester and resource are used in an exchanged order. From (V), each $aclrule_i \in ACLRULES_r$ is a tuple (a_j, ACL_j) and

$\exists\ (a_{res}, ACL_{res}) \in ACLRULES \mid ACL_{res}$ has all the ACL-based authorizations that grant access to res, according to the rules authority of resource res (XXV)

From (IV), each $acl_i \in ACL_{res}$ is a triple (a_j, cr_j, act_j) and

$\exists\ (a_{req}, CR_{req}, ACT_{req}) \in acl_{res}$ (XXVI)

Therefore, for ACLRULES,

$$authorized = \begin{cases} \text{true, if } act_{req} \in ACT_{req} \text{ and } \forall cr_i \in CR_{req}(cr_i \text{ is valid}) \\ \text{false, otherwise} \end{cases} \quad \text{(XXVII)}$$

At the end of the session we show how to evaluate a context rule cr_i.

ATTRULES. From (VI), each $attrule_i \in ATTRULES_r$ is a 7-tuple $(a_{resa}, s_{res}, a_{reqa}, s_{req}, inherit_{req}, CR_{attrule}, ACT_{attrule})$ and has to pass through the process of validation until one grants the access or all the rules from the authority rules were verified. Ignoring, for now, the $inherit_{req}$ (explained later), there are four things to validate on an $attrule_i$: if $act_{req} \in ACT_{attrule}$, if the context rules $CR_{attrule}$ hold (explained later) and if the s_{res} and s_{req} are attributes given to res and req by a_{resa} and a_{reqa}, respectively.

The attributes validation uses the definitions (VIII), (IX) and (X). From (X),

$\exists\ (a_{resa}, AUTHATTRS_{resa}, AUTHRULES_{resa}) \in AUTHS$ (XXVIII)

$\exists\ (a_{reqa}, AUTHATTRS_{reqa}, AUTHRULES_{reqa}) \in AUTHS$ (XXIX)

and, from (VIII) and (IX),

$$valid\_attributes = \begin{cases} \text{true, if } s_{res} \in AUTHATTRS_{resa} \text{ and} \\ \qquad s_{req} \in AUTHATTRS_{reqa} \\ \text{false, } otherwise \end{cases} \quad \text{(XXX)}$$

If the $inherit_{req}$ is true and the ownership of the requester's address is approved, then the rule also has to be tested against the address of the requester's owner. FROM (XIII), (XIV), (XV):

$\exists\ (a_{req}, owner_{req}, approved_{req}, authority_{req}) \in RES$ (XXXI)

The owner of req is $owner_{req}$. Thus, checking the $approved_{req}$ will lead to if the ownership of the requester has been approved. Thus, if both, $inherit_{req}$ and $approved_{req}$, are true then the rule has to evaluated with $owner_{req}$ as the requester.

Therefore, for ATTRULES,

$$authorized = \begin{cases} \text{true, if } req\text{'s and } res\text{' attributes are valid,} \\ \qquad act_{req} \in ACT_{req} \text{ and } \forall cr_i \in CR_{attrule}(cr_i \text{ is valid) or} \\ \quad \text{if } inherit_{req} \text{ and } approved_{req} \text{ are true,} \\ \qquad owner_{req} \text{ as the requester makes} \\ \qquad req\text{' and } res\text{' attributes valid,} \\ \qquad act_{req} \in ACT_{req} \text{ and } \forall cr_i \in CR_{attrule}(cr_i \text{ is valid)} \\ \text{false, otherwise} \end{cases} \quad \text{(XXXII)}$$

At the end of the session we show how to evaluate a context rule cr_i.

CR.

From (I), the context $cr \in CR$ is a 4-tuple, $(a, s, comp, v)$, composed by the address a of the context's source, a string s that identifies the context in source's scope, a comparator $comp$ to evaluate the context and an integer v that serves as parameter to the comparison.

To verify if the cr holds, the current value of the context is required. From (XII),

$\exists\ (a, s, v_{current}) \in CONTEXTS$ (XXXIII)

where $v_{current}$ has the newer value for context (a, s). Thus,

$$cr\_holds = \begin{cases} \text{true, if } v_{current}\ .comp.\ v \text{ is } true \\ \text{false, otherwise} \end{cases} \quad \text{(XXXIV)}$$

5.2 EXPERIMENTAL EVALUATION

We divided the experimental evaluation section in three parts. Firstly, we present our experimental environment (Section 5.2.1). Secondly, we expose the E-ControlChain usage cost (Section 5.2.3). Finally, we show the burden it causes on constrained resource devices (Section 5.2.4).

5.2.1 Experimentation environment

For these evaluations, we used the environment depicted in the Figure 5.3. We have a Raspberry Pi 3 B+ that has some simulated sensors and actuators plugged to it. It also runs an Ethereum client and uses a Decentralized Application (DApp) interface, implemented in JavaScript, to interact with E-ControlChain contract published on Ethereum. This Ethereum client does not mine, it only synchronizes with the Ethereum network (for more details see Section 3.2).

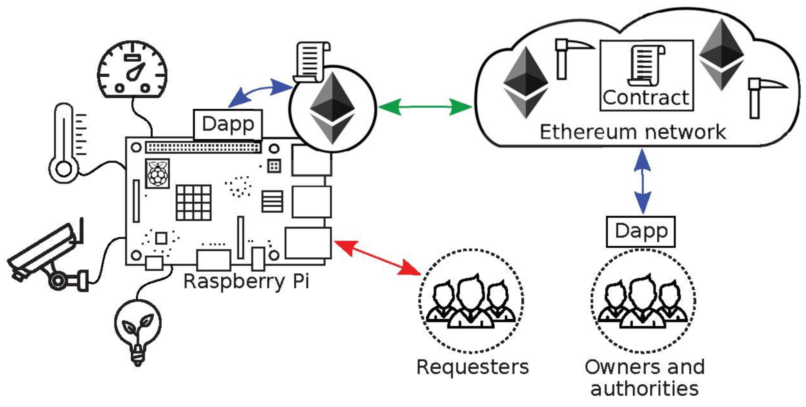


Figure 5.3: Experimentation environment

The resource owners, authorities and requesters also interacts with the E-ControlChain contract on Ethereum network trough a DApp interface. Like the resource devices, with this interface, they can call functions of E-ControlChain in order to publish access rules, attributes, context information and so on. The attempt of access resources can be done using IoT communication protocols, like MQTT or CoAP. Note that, for MQTT, a broker is required to intermediate the communication between requesters and resource devices. Therefore, the DApp interface and the Ethereum node client is required to be on the broker too. In our experimentation, we used the direct access.

The E-ControlChain contract for all experiments, unless otherwise specified, were implemented in Solidity version 0.4.23 and the version of running Ethereum clients and miners was 1.8.17. The E-ControlChain pseudocode is available in the Appendix A and the full code in the Appendix B. The Raspberry Pi was running Raspbian released on 2018-06-27 with kernel version 4.14. Furthermore, to emulate the Ethereum network, we used an Ubuntu 18.04 with kernel 4.15 over a MacBook Pro with core i7, 16GB of RAM and SSD storage. Finally, we also developed a framework to easy and automate the configuration, initialization, deployment and starting of the network on multiple hosts and/or multiple Ethereum clients in a same host. Thus, the nodes in the network were all created through this framework.

5.2.2 Use case tests

This section shows some use cases and how they can take advantage of the E-ControlChain. In particular, we based our use cases in the main steps in device's life-cycle [143]: (1) distribution after its fabrication (Section 5.2.2.1) and the use of devices by the final user (Section 5.2.2.2). The first use case shows the transferring of the control over a device. The second use case show how the controlling of a device. However we also present its usage in other scenarios like public vehicles tenancy and the protection of generic resources outside the IoT. All the tests on this section were generated using the Ethereum Go version 1.9.5 and Solidity 0.5.11.

5.2.2.1 Tracking of assets' ownership

During tests, we noted that the E-ControlChain can also deal with other important topic in the IoT's study area, the tracking of assets [144, 145]. It can be done through the transfer of control over objects. It provides easily verification of ownership and origin of an asset using ownership history. Besides this, who detains the ownership over an object also detains the control over it and can create access policies. As an extra feature, it can be used to identify and avoid stolen assets and unofficial products offered as official ones.

Some of the tracking scenarios requires complex systems, like the tracking of which raw materials were used to fabricate a specific product, i.e., with composition functions. In this case, a complex chaining of materials and products are required and E-ControlChain does not comply with it. However, without this composition, it can provide the tracking of the assets' ownership.

The ownership tracking can be made using the contract "BasicAndCommonControl" from E-ControlChain, specifically, using the functions "setOwner" and "approveOwnership". The very first owner has the capability of defining him or herself as the owner of the asset. After he or she can create a transfer transaction indicating the new owner and so on. It is important to remind that, after each ownership transfer, the new owner has to accept the ownership. As the new ownership information does not erase the old one and only turn it into historic information, the history of owners can be obtained and verified.

Next, we present one example of how this functionality can be used. Suppose a fabric that assemblies a device and delivers it to customers, through distributors and retail stores. In this example, the fabric will ship the device to a distributor. Then it will ship to a retail store, where customers buy the smartphone.

For the following function calls and results, we will use the correspondence between entities and addresses showed in Table 5.2.

Table 5.2: Tracking of assets' ownership - correspondence between entities and addresses.

Entity	Address
Device	`0x3779a449784Dd73379f88888079564529F5Ae752`
Owner 1 (Fabric)	`0x1156976F5Eb86cbB225240BC1B1470FBE0ac1BC7`
Owner 2 (Distributor)	`0x4259F68a405c008591E1816Ad4Da789E2E3791aB`
Owner 3 (Retail store)	`0x200DD168e935cad1f7df7886d3Dd727F42F8a318`
Retail store 2	`0x7fb7b2342Cf25779e29b5279BC90F8ad0870C98E`

The process starts with the fabric generating an address for the next device, through the generation of a new Ethereum public key. The fabric generate it in its own servers. After its generation, the fabric own the address calling the function

```
setOwner(0x3779a449784Dd73379f88888079564529F5Ae752, 0
    ↪ x1156976F5Eb86cbB225240BC1B1470FBE0ac1BC7)
```

and embed the keys and the address of the E-ControlChain in the device, protecting the private key from external access and all of them from modification.

The call to the function “setOwner” generates a transaction hash (tx hash), like

```
b'"B\x11v#\x91\xfb9f&T\x01\xae\xe6\xcbFN\x03Q\xed<un\xae\xab\x1aRQ\x96\
    ↪ x15\xda\xdb'
```

After the transaction gets mined the tx hash can exchanged by a receipt, like

```
(AttributeDict({'args': AttributeDict({'addr': '0
    ↪ x3779a449784Dd73379f88888079564529F5Ae752', 'new_owner': '0
    ↪ x1156976F5Eb86cbB225240BC1B1470FBE0ac1BC7'}), 'event': '
    ↪ setOwnerEvent', 'logIndex': 0, 'transactionIndex': 0, '
    ↪ transactionHash': HexBytes('0
    ↪ x224211762391fb3966265401aee6cb464e0351ed3c756eaeab1a52519615dadb'
    ↪ ), 'address': '0x9F94f29EF95E45ec07621d8a755A8055691bc14c', '
    ↪ blockHash': HexBytes('0
    ↪ x88ee91e851b418f2e94b8a986e9ff555a1693b9ae28a94ee5e2dc9c88d8739ca'
    ↪ ), 'blockNumber': 1417}),)
```

The receipt contain information about the function execution, like parameters, the success of its execution and events that were triggered. In this case, the event triggered was “setOwnerEvent”. Extracting the information from the event, we have

```
AttributeDict({'addr': '0x3779a449784Dd73379f88888079564529F5Ae752', '
    ↪ new_owner': '0x1156976F5Eb86cbB225240BC1B1470FBE0ac1BC7'})
```

Although, to manipulate the access control of the device, the fabric is required to approve the ownership using the function “approveOnwerhip”, this is not necessary for transfer it to the next owner, in this case, the distributor. Thus, right after the last ownership was mined (for more information, see Section 3.2), the fabric can transfer the ownership of the device to the distributor calling the function

```
setOwner(0x3779a449784Dd73379f88888079564529F5Ae752, 0
    ↪ x4259F68a405c008591E1816Ad4Da789E2E3791aB)
```

After this function call is mined the distributor can transfer the device to a new onwer, in this case, the retail store using the function call

```
setOwner(0x3779a449784Dd73379f88888079564529F5Ae752, 0
    ↪ x200DD168e935cad1f7df7886d3Dd727F42F8a318)
```

Anyone can see the triggered events and filter then based on their contract and name. Therefore, after the execution of this functions, the history of the event “setOnwerEvent” show us three changes of owners for the devices’ address, where the last one is the current owner:

```
AttributeDict({'addr': '0x3779a449784Dd73379f88888079564529F5Ae752', '
    ↪ owner': '0x1156976F5Eb86cbB225240BC1B1470FBE0ac1BC7'})
AttributeDict({'addr': '0x3779a449784Dd73379f88888079564529F5Ae752', '
    ↪ owner': '0x4259F68a405c008591E1816Ad4Da789E2E3791aB'})
AttributeDict({'addr': '0x3779a449784Dd73379f88888079564529F5Ae752', '
    ↪ owner': '0x200DD168e935cad1f7df7886d3Dd727F42F8a318'})
```

Note that, although it is not necessary the approve of the ownership to transfer a device, it is useless, *i.e.* it cannot be used, without the ownership be approved . This occurs because no rule can be added to the device, as no rule authority can be set to it and the device cannot access another device with its address unless the resources rules define it explicitly.

Also, lets say the distributor (owner 2) wants to fool two different retail stores, sending them the same device on E-ControlChain. In order to do this it has to generate a second transaction calling the function “setOwner” similar to the one that it made to the retail store (owner 3), but with a different address as new owner:

```
setOwner(0x3779a449784Dd73379f88888079564529F5Ae752, 0
    ↪ x7fb7b2342Cf25779e29b5279BC90F8ad0870C98E)
```

The receipt of this call returns an empty tuple:

```
()
```

This means that the execution failed and no changes to the data was made. The reason behind its failure is that the malicious distributor cannot forge the signature of the current owner (the retail store in the first transaction). This can be proved listing all events triggered when a new owner is defined and seem that it remains the same as before:

```
AttributeDict({'addr': '0x3779a449784Dd73379f88888079564529F5Ae752', '
    ↪ owner': '0x1156976F5Eb86cbB225240BC1B1470FBE0ac1BC7'})
AttributeDict({'addr': '0x3779a449784Dd73379f88888079564529F5Ae752', '
    ↪ owner': '0x4259F68a405c008591E1816Ad4Da789E2E3791aB'})
AttributeDict({'addr': '0x3779a449784Dd73379f88888079564529F5Ae752', '
    ↪ owner': '0x200DD168e935cad1f7df7886d3Dd727F42F8a318'})
```

It occurs because the function “setOwner” has a require statement that authorizes the change only if the current calling user is the current owner or there is no owner for the address. Of course, if the two transactions are created at the same time and the second one gets mined before, the result would be reversed.

In a similar way, if someone decides to stole a truck full of devices, the product will have no or little value as the thief cannot transfer their ownership. This could be different if the thief also acquires the current owner’s private key. Therefore, it has to be kept in a secure environment.

5.2.2.2 Guests in a smart home

Smart homes are complex environments because its diversity of devices. While some of them are more processing powerful, like televisions and computers, others are more limited, like smart locks and temperature sensors. However, in the good side, as it is a structured environment, energy is easily provided to all devices. Thus, there is no limitation on the usage of the devices’ communication equipment (commonly, a radio).

In most cases, the operator, i.e. who needs to configure the devices of a smart home, will be the owner of the objects in the house and will have only basic knowledge in computation. Thus, keep in mind that many of the steps described below can be automatized or have a pretty and simple interface with the user.

All the following function calls were executed with the real address, however, to simplify the explanation we substitute them by fictitious and smaller ones as shown in table 5.3.

Let’s say that Alice (address 0x2), bought many cool smart devices for her house, between them a special one (address 0x1). As Alice is a concerned person, she verified that (1) the device is original, verifying if the first owner address is equal to the address of the official fabric; (2) if the device is new through verifying if no one created access policies for the device; and (3) if it was not stolen, automatically done as only the current owner can transfer the ownership of a device (using the premise that the current owner’s key weren’t stolen). This is possible using the techniques shown in Section 5.2.2.1.

Table 5.3: Guests in a smart home - mapping between entities, real and fictitious addresses.

Entity	Fictitious address	Real address
Device	0x1	`0x3779a449784Dd73379f88888079564529F5Ae752`
Alice (Host/Owner)	0x2	`0x1156976F5Eb86cbB225240BC1B1470FBE0ac1BC7`
Alice's friend (Housemate)	0x3	`0x4259F68a405c008591E1816Ad4Da789E2E3791aB`
Bob (Guest)	0x4	`0x200DD168e935cad1f7df7886d3Dd727F42F8a318`
Some other guy/girl (guest)	0x5	`0x309D72826Be336db6c45ee1763f1F7B999c2CAf2`

To use the device, Alice has to follow two steps: (1) approve her ownership through the function "approveOwnership"; and (2) create access policies granting permissions for her. For the second step, she could use either access methods of the E-ControlChain: ACL, capability or ABAC. In this example, she chose to use ABAC as it fits better in her future necessities. The first step can be done calling:

```
approveOwnership("0x1"); // only the current owner of 0x1 is authorized
    ↪ to call it.
```

The second step requires at least four function calls:

```
1 setRulesAuthority("0x1", "0x2"); // define who will define access
      ↪ policies to the device
2 createAttributeRule("0x2", "home_device", "0x2", "resident", true, [],
      ↪ [0,1,2]);
3 setAttribute("0x1", "home_device", true);
4 setAttribute("0x2", "resident", true);
```

In line 1, Alice defined herself as the rules authority, i.e. who can define the access policies for the device. The line 2 creates a attribute-based rule. This rule defines that any address that have the attribute "resident" (4th parameter), defined by 0x2 (3rd parameter), can access any address that has the attribute "home_device" (2nd parameter), also defined by 0x2 (1st parameter), to read, write and execute (7th parameter, respectively 0, 1, 2) independently of the context (6th parameter). Also, as the flag "inherit_owner_attributes" (5th parameter) is true, any address owned by an authorized address will also has access to the home devices. Finally, in lines 3-4, she gives the attributes "home_device" to the device and "resident" to herself. Of course, this permission only are valid for those devices that are using Alice's address as a rule authority.

Note that both functions "setRulesAuthority" and "createAttributeRule" allows the delegation of the access control. The first (setRulesAuthority) can be used to delegate full control over the device (except the ownership). The second (createAttributeRule) allows a second level of delegation where any entity can be defined as an attribute authority, *i.e.* the entity who choose others to receive attributes that grants the access.

For now on, she can access the device. Any other entity that she did not grant the attribute "resident" cannot. This can be confirmed by the response of the function calls:

```
authorizedByAttributeRules("0x1", "0x2", 0);
  <True>
authorizedByAttributeRules("0x1", "0x3", 0);
  <False>
```

The function "authorizedByAttributeRules" receives three parameters, the first is the address of the resource, the second is the address of the requester, and the third is the action that the requester is trying to execute on the resource. Also, this function can be executed by any participant and, thus, a gateway device can help devices in this task.

If she wants to grant the same access privileges to let's say, a housemate (0x03), she has only to call:

```
setAttribute("0x3", "resident", true);
authorizedByAttributeRules("0x1", "0x3", 0);
 <True>
```

If the Alice's housemate moves out of their house, Alice can remove her permissions removing the attribute "resident" of the housemate through calling:

```
setAttribute("0x3", "resident", false);
authorizedByAttributeRules("0x1", "0x3", 0);
 <False>
```

Alice also is a party girl and frequently invite her colleges for parties on her home. As a good party host, she always wants to make her guests comfortable and feel like they were in their home, so she wants to provide them with access to devices she bought. However, as parties in her home are very frequently, she does not want to keep giving and revoking access permissions in every party to every guest, as this would be laborious. So she prefer to maintain the access permission. The only problem is that between the guests is Bob (address 0x4). She knows that he is not a reliable person and that sooner or later other unreliable persons will become her guests too. Thus, she wants to provide the authorization access only while the party is going on.

To achieve her goals, she needs to create a rule that authorize guest under the described circumstances. This rule requires a context rule capable of defining when there is a party going on. To create this context rule, she can call the function:

```
newContextRule("0x2", "giving_a_party", 2, 1);
```

The created context rule states that, in order to it be considered a valid context rule, the value obtained from context "giving_a_party" (2nd parameter) provided by the Alice's address (1st parameter) has to be equal (3rd parameter, the number 2 represents "=") to the value presented in the 4th parameter.

Each created context rule has an unique identifier in the E-ControlChain that is its index in the array of contexts. Alice can obtain her context rule index monitoring the event "newContextRuleEvent" triggered when the newly created context rule gets mined. In her case, the index is 0 as revealed by the field "id" in the event:

```
AttributeDict({'source': '0x1156976F5Eb86cbB225240BC1B1470FBE0ac1BC7', '
    ↪ id': 0, 'identifier': 'giving_a_party', 'comparator': 2, 'value':
    ↪ 1})
```

After discovering the new context identification, she can create a rule using it as follows:

```
createAttributeRule("0x2", "home_device", "0x2", "guest", true, [0], [0,
    ↪  1]);
```

This new rule authorizes any address that received from Alice's address (3rd parameter) the attribute "guest" (4th parameter) has the permission to read and write (7th parameter) content for any address that also received from Alice's address (1st parameter) the attribute "home_device" if the context of index 0 hold (6th parameter). As the 5th parameter is true, any guest's owned device also have his or her permission. Again, this permission only are valid for those devices that uses Alice's address as a rule authority.

As Alice already defined an attribute "home_device" for her device previously, she only has to define who are the guests of her party and, in this case, is Bob:

```
setAttribute("0x4", "guest", true);
```

As can be seen in the access attempt, Bob still cannot access the device:

```
authorizedByAttributeRules("0x1", "0x4", 0);
  <False>
authorizedByAttributeRules("0x1", "0x4", 1);
  <False>
```

This occurs because Alice has not yet set the context "giving_a_party" to 1.

When Alice's party is ready to begin, she sets the context and Bob receives the permission to read and write but not to execute (as defined in the attribute rule):

```
setContext("0x2", "giving_a_party", 1)
authorizedByAttributeRules("0x1", "0x4", 0);
  <True>
authorizedByAttributeRules("0x1", "0x4", 1);
  <True>
authorizedByAttributeRules("0x1", "0x4", 2);
  <False>
```

Also, to grant the same permission to any other guest, for now on, she only has to give him or her the attribute "guest":

```
authorizedByAttributeRules("0x1", "0x5", 0);
  <False>
setAttribute("0x5", "guest", true);
authorizedByAttributeRules("0x1", "0x5", 0);
  <True>
```

To revoke a guest's permission, Alice's just need to remove the attribute "guest" from entity:

```
setAttribute("0x4", "guest", false);
authorizedByAttributeRules("0x1", "0x4", 0);
  <False>
```

And, for revoking all guest's accesses, she only need to set the context "giving_a_party" to 0 as this causes the context rule to not hold and, therefore, the attribute rule deny the access attempt:

```
setContext("0x2", "giving_a_party", 0);
authorizedByAttributeRules("0x1", "0x5", 0);
  <False>
```

As explained in the Section 5.2.2.1, Bob will fail if he tries to steal device's ownership or Alice's device itself as it cannot forge the Alice's signature. However, Bob can try to gain unauthorized access in many different ways, like: (1) trying to approve the ownership of an entity pretending to be as Alice; (2) setting a new rule for Alice's device; (3) giving himself an attribute that could grant access to a device; and (4) changing a value from a context.

The same protection that keeps Bob away from stealing the ownership of Alice's devices is used here. The ownership approval can only be invoked by the current defined owner. To verify it, the given public key is validated against the transaction signature and the address of the sender is extracted from this public key. Therefore, the (1) will fail by the same reason the stealing of an ownership does not work. In a similar way, only the device itself can change the value from its created context. Therefore, only if Bob find a way to forge the signature, (4) will fail.

Each created rule is coupled with the entity who made the transaction. Also, before a device be accessible, its owner has to define a rules authority. Devices only obey to rules created by its defined authority. Therefore, any other entity creating rules does not interfere with Alice's devices. So, attempt (2) will fail. Also, Bob cannot create a rule pretending to be a specific rules authority because he cannot forge the signature. In a similar way, attributes also does not have the value if not received from the attribute authority defined in the rule and (3) will also fail.

Finally, if Alice wants to sell or give one of her devices to another person, she invokes the function "setOwner" as discussed in Section 5.2.2.1. Calling this function clears the rules authority from the device's address. Therefore, it will be available only when the new owner approve his or her ownership and define a new rules authority for it.

5.2.2.3 Public vehicles tenancy

Public vehicles, like bicycles and scooters, also are becoming very popular in many cities. They remain locked until authorized users unlock them, normally, through smartphone applications. In these systems, there are different user profiles, some of them are constant users, like people that lives in the city and makes daily use of them, and other user profiles are only casual, like tourists.

Constant users could prefer to join a subscription plan while the casual ones could prefer to pay per use or day use. In such scenario, the company behind the vehicles could maintain rules for the different profiles and give attributes for each user according to its subscription plan. besides this, context could be used, for example, to limiting the user usage or define the vehicles that are currently unavailable for maintenance.

We will not show the full list of commands required for this use case as it is similar to those presented before.

5.2.2.4 Protection of resources outside the ioT

In systems with continuous usage, it is very common the usage of roles, like in Google Team Drive from G Suite [146] or Open Journal Systems (OJS) [147]. Although they have completely different purposes, in short, both of them deals with the protection of files, and control the access to them. Note that the files can be seen as equivalent to sensors reads and actuators from the IoT. Furthermore, the roles can be translated to E-ControlChain as explained in Section 4.4.

In order to make the access control role-based access control compatible with E-ControlChain, it needs to be converted to attributes-based mechanism: the users that needs access to files are the requesters, *i.e.* entities; the service provider can be the owner and attribute authority; roles are attributes given by the attribute authority. Thus, using the example of the Google Team Drive. The Google would be the owner of the files (in access control scope) and the attribute authority. besides this, each user receives one or more roles (through attributes) from it. Note that, for example, Google could allow users, with the attribute "administrator" for the drive, to change or include new users as this operations also can be seen as a resource and can exists a rule that authorizes administrators to do it.

Therefore, E-ControlChain can also be applied to similar environments that resides outside the IoT. Basically, a good part of the use cases that uses ACL, Capability, role, attribute and UCON-based access control systems can also benefit from E-ControlChain.

5.2.3 Usage cost

The approximated cost of executing each E-ControlChain function, using geth version 1.9.5 and solidity 0.5.11, is presented on the table 5.4. Note that this cost can be drastically reduced adopting other mechanisms of "mining" (see Section 3.2.2) instead the proof-of-work. For example, it could be used the proof-of-authority, or approaches based on Directed Acyclic Graph (DAG) (like IoT Chain and IOTA [135, 90]) or based on Byzantium algorithms (like the Algorand [134]).

Table 5.4: Cost of executing the functions with a gas price of 20 gwei.

Function	**Cost in Ether**	**Cost in USD****
E-Controlchain contract deployment	0.09024	18.048
setOwner	0.00113	0.226
approveOwnership	0.00088	0.176
setRulesAuthority	0.00092	0.184
newContextRule	0.00228	0.456
createAttributeRule	0.00751	1.350
setAttribute	0.00092	0.184
setContext	0.00093	0.186
authorizedByAttributeRules*	0.00000	0.000

* Although it is executed over the contract, it is executed locally on the Ethereum client and does not generate extra cost. Only functions that change the state of the network are charged by the extra cost.
** Based on the price of Ether on 29 Oct. 2018 (around USD 200.00 for 1 ether).

5.2.4 Burden on constrained devices

One important factor about an access control architecture for the IoT is its viability in scenarios with constrained resource devices. In this section, we analyze the consumption of CPU, disk, network and others when running the E-ControlChain over the Raspberry Pi and discuss how ControlChain can be compatible with even more constrained devices.

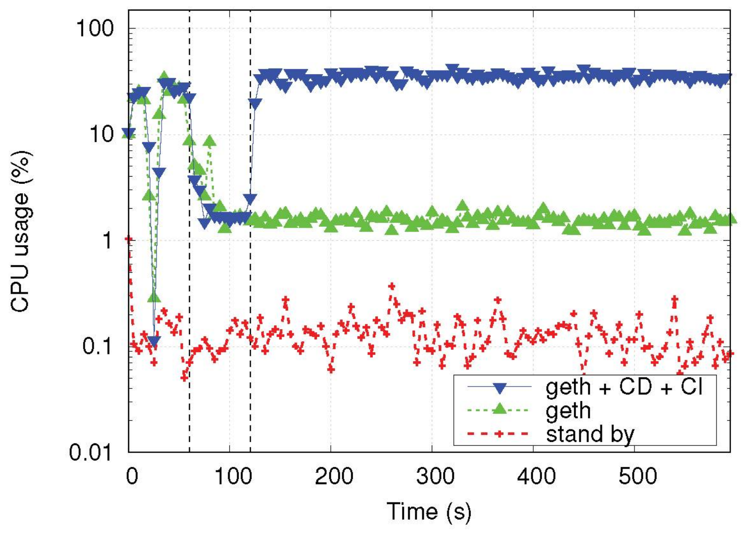
In order to measure the impact of E-ControlChain over the Raspberry Pi, we collected its resource usage in three different scenarios. In the first, "stand by", it was in a resting state, i.e., only with its fundamental activities running (basic operational system tasks). In the second one, "geth", a Ethereum client (geth) was running and synchronizing with a private Ethereum network, however with no contract interaction. In the third one, "geth + CD + CI", a Ethereum client was running with contract interactions, in a way that it saturate the capacity of the Ethereum mining process (see Section 3.2). In order to do this, we generated 1000 transactions and, as soon as all of them was mined, we generated another 1000 transactions and so on. Using this methodology, we achieve an average of 28 mined transactions per second. It is important to highlight that the mining methodology adopted was the Proof-of-Work with an average block production interval of 10 seconds and that the mining process was taken only by the computer (see Section 5.2.1), *i.e.* the Raspberry Pi only synchronizes with it.

The generated transactions was a mix of `setRulesAuthority`, `newContextRule`, `createAttributeRule`, `setAttribute`, `setContext` functions. The `setOwner`, `approveOwnership` and `authorizedByAttributeRules` functions weren't used because the first two are security dependencies for the `setRulesAuthority` and the last is executed locally in the geth node and, consequently, are not mined into the Blockchain. The second and third scenario was used to reveal how much of the burden caused by E-ControlChain is actually caused by the Ethereum fundamental activities. Finally, the measurements were obtained using the collectd v5.7.1 tool with the collection interval set to five seconds and all measurement graphs are an average of 10 repetitions.

The "geth" and "geth + CD + CI" scenarios can be divided in periods. The "geth" can be divided in two periods: before geth complete initialization (BG) and after geth complete initialization (AG). The "geth + CD + CI" can be divided in three periods: BG, contract deployment (CD) and contract interactions (CI). The approximated time in which the division between them occur is represented in the result graphs as two vertical dashed lines. The first one divides the BG from AG and BG from CD. The second one divides the CD from CI.

Figures 5.4 and 5.5 presents the Raspberry Pi CPU and memory usage, respectively, for the three scenarios. The CPU results shows that the E-ControlChain causes a CPU consumption peak of 35% in the BG and CI period. The result also show that the Ethereum client is only responsible for 2% of the CPU usage after geth initialization and synchronization and that the operational system CPU usage in stand by is around 0.1%. On the other hand, the memory usage results shows the same order of magnitude on geth scenarios (350MB versus 500MB in the CI period). Their memory usage drop around the 485 seconds could be the release of memory used to store unused geth code or information that after a while in inactivity gets freed, however, a more deep investigation is necessary to reveal the real reasons behind it. In the stand by scenario, the memory usage was around 20MB.

Figure 5.4: CPU impact



The measurements obtained from disk reading, writing and total usage is presented in Figures 5.6, 5.7 and 5.8, respectively. The disk read measurements shows very few readings in the AG, CD and CI periods with the more overloading activity being the reading of 10KB in an interval of five seconds. In the BG period, there was peaks of 200KB. In counterpart, the write measurements shows a slight increase in "geth" scenario when compared to "stand by" one. The "geth + CD + CI" scenario shows peaks 10x higher than the "geth scenario", however this peaks are about 100KB. In the BG period there are write peaks of 2MB. The stored information on disk shows an increase of 30MB in the BG period and a linear use of the storage space in CI period. In almost 500 seconds of contract interactions in the CI period, it used approximately 16MB. This measure up to 2.7GB of storage requirement per day. If a real E-ControlChain was heavily

Figure 5.5: Memory impact

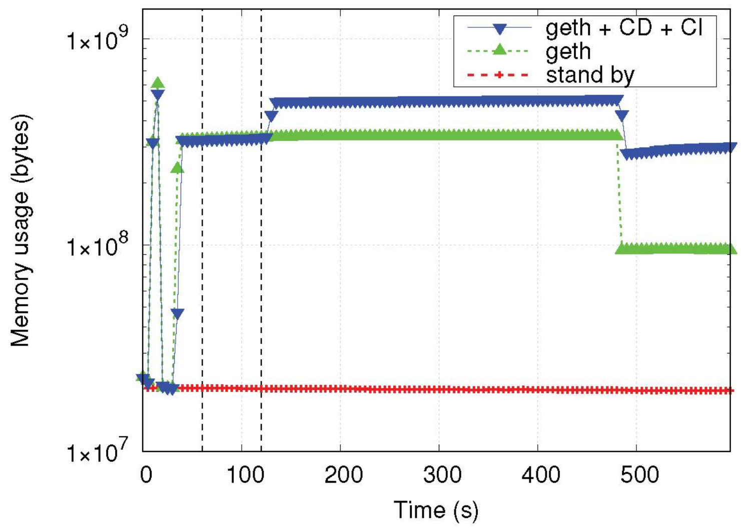
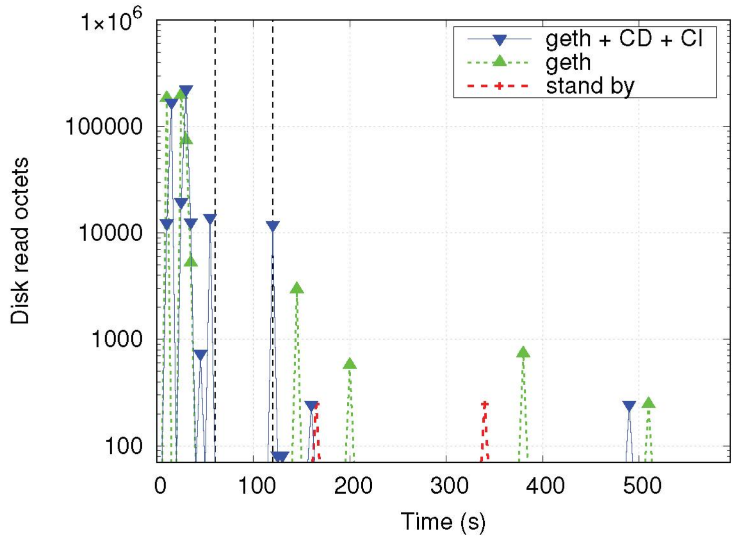


Figure 5.6: Disk read impact



used as in this experimentation, perhaps this could be the main problem of using it over the IoT network. Fortunately, as discussed after, there are some ways to reduce or avoid this problems.

Figures 5.9 and 5.10 presents the quantity of received and transmitted octets, respectively, of the network interface. As expected, there was an increase in the network traffic, that is necessary to keep the network synchronized. The "geth" scenario received and transmitted, in average, 1KB and 750B, respectively, for each interval of five seconds. The "geth + CD + CI" scenario received and transmitted a bigger quantity of information in the CI period (received 30KB and transmitted 10 KB of information approximately for each interval of five seconds). The "stand by" scenario keep the information transmission and receipt lower than 20B for each interval of five seconds.

The results obtained in these experiments shows the viability of running E-ControlChain in even more resource constrained devices than the used one. However, even so, for devices with a severe storage restriction, it can be applied a replication factor to the stored data, *i.e.* each devices keeps a part of the data with a probability of $x\%$, instead of it in totality. It is possible to apply selectively information storage also, making them keep only information related to themselves and, of course, the ones necessary to keep updating with the E-ControlChain (like

Figure 5.7: Disk write impact

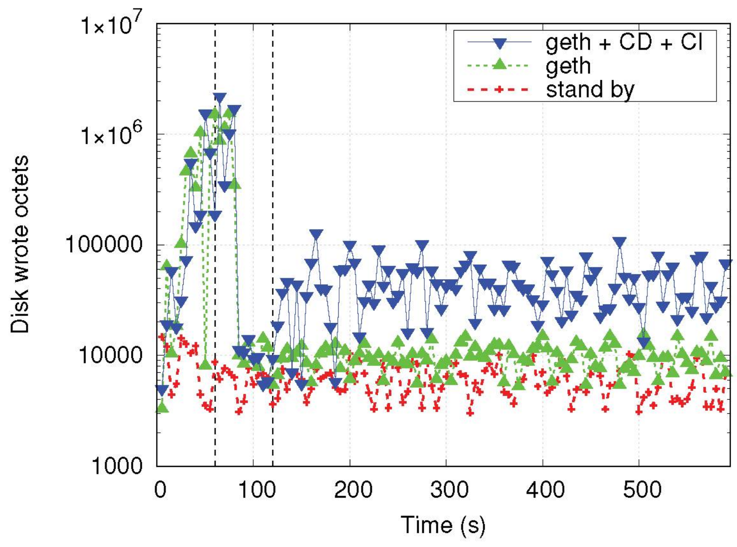
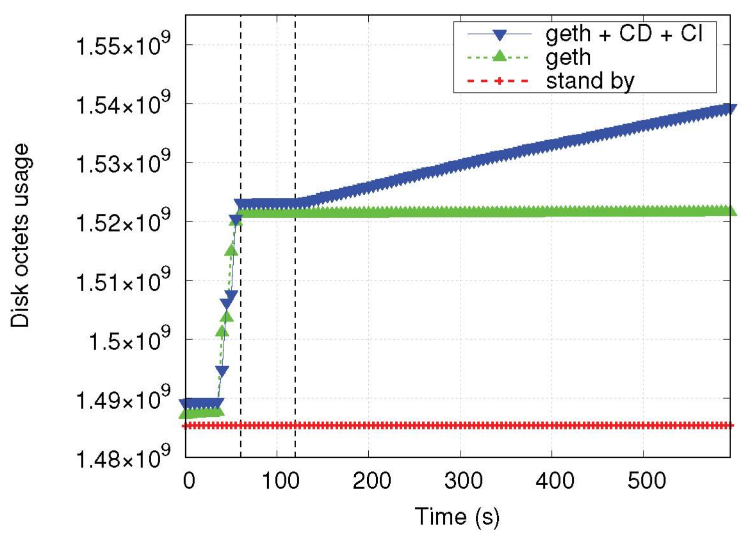


Figure 5.8: Disk usage impact



some of the most recent mined blocks). Finally, supporting devices (see Section 5.2) can be used for helping those even more constrained devices (that cannot handle CPU, memory, storage or network overload imposed by E-ControlChain). So, when they receive an access request, they forward it to a supporting device that will check the authorization for them. Of course, the process of selecting these supporting devices requires caution, otherwise malicious ones can end up being selected, compromising the device depending on it. We suggest that, when possible, the constrained device always select devices owned by its own owner or from its confidence circle.

To demonstrate the potential of a single Raspberry Pi as a supporting device, we made a stress authorization test over three scenarios (Figure 5.11). In the first, `Without CI`, the E-ControlChain was deployed but there were no transactions towards it. In the second, `During CI`, the Raspberry Pi was under the scenario described for CI period, where the Ethereum network was saturated with transactions towards the contract. In the third, `After 600s of CI`, there were no contract transactions, however, it occurred after 600s of CI period. The results shows that the number of transactions made to the contract has no or little influence over the number of authorizations the Raspberry can check. Both, "Without CI" and "After 600s of CI"

Figure 5.9: Network receipt impact

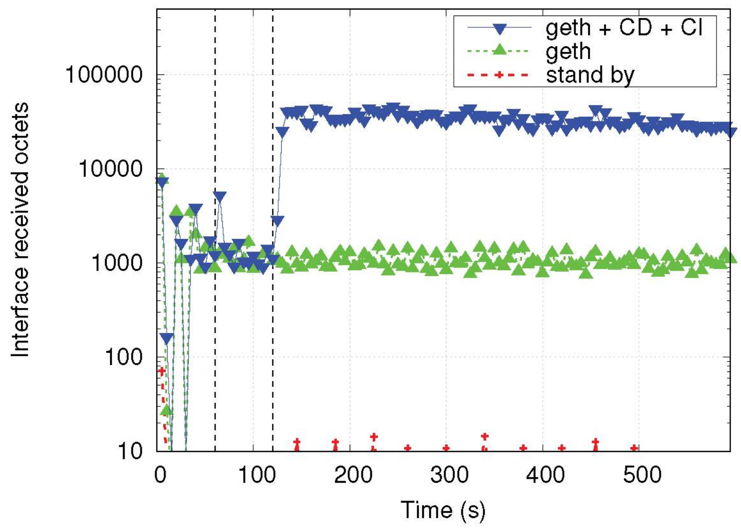
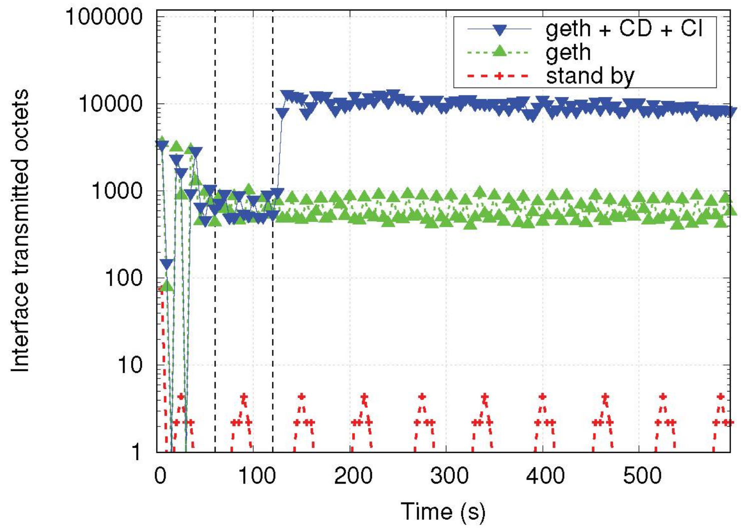


Figure 5.10: Network transmission impact

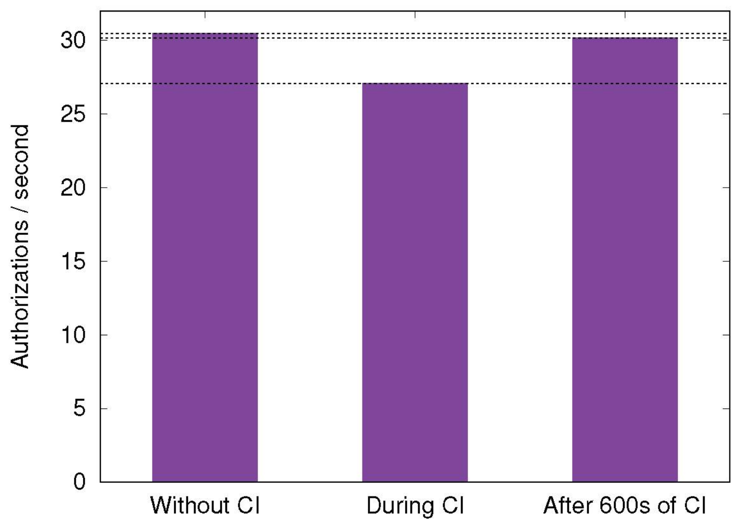


scenarios presented a similar result of approximately 30 authorization per second. However, in "During CI" scenario, the average of authorizations dropped to approximately 27, showing a reduction of about 10% with the intense transaction mining. This reduction is, probably, caused by mechanisms that deal with the concurrency of reading from and writing to the contract at the same time. Furthermore, for big networks, multiple Raspberry Pi devices can be used, and is expected that the number of authorizations per second will increase arithmetically with the number of devices.

5.3 LIMITATIONS

The limitations of E-ControlChain can be divided in three main decision axes: design, base components and algorithm. The design axis is limitations that involve the basic concepts behind E-ControlChain. The base components axis is limitations imposed by the adopted platforms, libraries or other software components used in the development. Finally, the algorithm axis is limitations taken in the development stage in favor of other benefits.

Figure 5.11: Number of authorizations per second



A design decision limitation is the adoption of a public Blockchain and the implementation of all algorithm decisions in the contract. The drawback of this approach is that it lacks of a strong privacy protection. Although, all the devices and users are behind addresses, that are known to provide a certain level of anonymity, profiling tools can be applied and, in some cases, could end up revealing the real identity behind them [148]. As we chose to implement all the algorithm decisions in the contract, most of the stored information cannot be really encrypted. However, some data can be replaced by another one using a bijective function and, therefore, difficult the deduction of what the data represent and what are the real value behind the masked data. Furthermore, in future works we are planning to create a version of E-ControlChain over private Blockchains or other more privacy-friendly Blockchain solutions. Another possible future work is the application of cryptography to hide the information published on Blockchain (maybe using shared keys cryptography algorithms). However, this solution also leads to the problem of how to share efficiently these shared keys.

We chose Ethereum as a platform and Solidity as a programming language for developing E-ControlChain. Naturally, each platform and language has its own limitations (whether intentional or not) that are forwarded to solutions that adopt them. In this sense, the E-ControlChain also inherited all the base components limitations from them. Probably, the most explicit of them is the requirement of an Ethereum client on the device itself or on its selected supporting device. Currently, there are compiled version of geth (one of the official implementations of the Ethereum clients) for Android, iOS, macOS, Windows and for Linux using the architectures 32-bit, 64-bit, ARM64, ARMv5, ARMv6, ARMv7, MIPS32 and MIPS64. Thus, out-of-the-box, a wide range of devices are supported. The geth source code is also available and can facilitate its provision for those that aren't supported yet. However, there are also others limitations that we discuss next.

One base component limitation is the choosing of a proof-of-work-based Blockchain as a central tool for E-ControlChain. These Blockchains uses network processing power to stays safe and avoid attacks or collusion. The lower the processing power the easier is to deploy successful attacks through, for example, miners collusion or overcoming of the network processing power. Although these attacks alone does not give new unauthorized access to devices, it can prevent changes in authorizations or management of devices, for example, preventing a context change registration. As much as it is a possible scenario, at least two things hardens its occurrence.

First, it would require more than 50% of the mining processing power to be in the control of the colluding miners or attackers. Second, if it happens the network will loss reputation and people will start to abandon it, making the miners loose their incoming source. Therefore, there are incentives to miners keep honest and protect themselves from these attacks.

Another problem with choosing the proof-of-work is the time required time to allow a new information to be part of the Blockchain. Of course, this is mainly dependent of the type of proof, its level of difficult, the quantity of information that could be mined in one block, the complexity of consensus establishment and the established reward fee. However, it is very common to take some time between a few seconds and minutes.

The dependency on public key pairs can also be considered as a base component limitation. The loss of a key requires the generation of a new one and the reconfiguration of all parameters (define the new devices owner, attributes, access rules, and so on). Also, if an attacker have access to the private key, it can hijack, control and create fake messages of a device behind the address (if there is one) and address's owned devices.

The base component and development language, Solidity, also brought some limitations to E-ControlChain. With the objective of keeping the contracts clean and avoid code bugs, solidity defines a limit to the number of variables defined in each function. Also, the only way to pass/return objects to/from functions in the used version of the Solidity compiler is using the ABIEncoderV2, however when we were making the tests, it was a little unstable and we decided not to use it. So, the lack of a suitable solution to work with objects in the experimenting time worsened the limitation imposed to the number of variables in a function. Therefore, implementing more complex and complete access control became a hard task. Because of this, as exposed earlier in Table 5.1, the current E-ControlChain's version does not implement obligations and accountability instructions. Furthermore, the owner can define only one context set for each rule and cannot create more complex allowed contexts, for example, using logical operators. Also, the unique way to edit an authorization is through is removal and addition. Although there is no big mystery in the implementation of these functions, it became a future work that could come true with a stable ABIEncoderV2.

Finally but not least there are also algorithm limitations, *i.e.* the ones imposed by implementation decisions. Two of them are: (1) only one rule for each pair resource-requester in the capability and ACL access control can be defined; (2) in order to remove an attribute-based rule, the context indexes and the allowed actions have to be passed in the same order they were passed in the rule creation. All this limitations could be avoided with more code instructions, however this would turn the code more complex, difficult to inspect and would increase its deployment and execution cost. Therefore, we choose to keep this limitations in the current version of E-ControlChain.

Another algorithm limitation is generating by choosing between letting a device define its own owner and allow stolen devices to be used again or loose the device if the owner loose its key. Although, we implemented the first solution, making the physical access prevails over any other control, it is required the change of only a single line of code to modify this behavior to the later option.

5.4 SECURITY ANALYSIS

To analyze the E-ControlChain in a security perspective, we divided this section in two parts. First, we use the framework STRIDE to analyze many different threat levels, ranging from physical and social attacks to network ones. Then we give an special attention to smart contract vulnerabilities.

We start our security analysis using the STRIDE framework [149]. It helps in building threat models and, in our opinion, are one of the best frameworks to analyze our architecture. It explores threats involving Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege. Table 5.5 shows what could be affected by them.

Table 5.5: Summarized security analyze of the E-ControlChain components.

Threat	What could be affected
Spoofing	Entities, network participants and miners
Tampering	Devices, Blockchain information
Repudiation	Entities
Information disclosure	Entities
Denial of service	Entities, network participants and miners
Elevation of privilege	Contracts

Spoofing. Spoofing attacks have the objective of impersonate someone else identity. If successfully launched, the attacker can execute unauthorized actions in behalf of the impersonated identity. Some of the attacks that can have spoofing characteristics are Sybil, Man In The Middle (MITM) and replay.

In Sybil attacks, the malicious entity forge multiple identities. It is specially effective against unprotected voting systems where it can generate an infinite number of new identities to manipulate the results. In E-ControlChain, a Sybil attacker can try to forge fake identities that could affect different layers, like Ethereum peers [150] and E-ControlChain entities (owners, resources and requesters). Ethereum peers communicate using Peer-to-Peer protocols, a successful Sybil attack can make an Ethereum peer connect only to forged peers, thinking that it is connecting to honest peers. Then, it can execute an eclipse network attack [150] over the peer. In this case, the Ethereum peer, instead of receiving the real Ethereum data, receives only the data generated by the attacker. Fortunately, if not combined with other attacks, the data generated by the attacker in an eclipse attack is, at most, garbage. In the other hand, if the attacker forges an identity of an E-ControlChain owner, resource or requester he or she can control the access policies, generate fake context information and make unauthorized access, respectively. besides this, it can also stole all the funds from accounts behind the entities. Ethereum protects itself from this attacks using public keys. In the Ethereum P2P network, the nodes are identified using the enode, public key in hexadecimal format. In the platform, including in the E-ControlChain, the entities are identified using the address, the first 160 bits extracted from hash KECCAK-256 of the public key.

A Sybil attacker could also try to overpass the Ethereum's mechanism of transaction ordering by forging multiple identities. As the identity occurs in an off-line mode, the attacker achieves this with ease. However, the currently used method for mining process is the Proof-of-Work (PoW), i.e. it depends only of the computing power. Thus, from the Ethereum's transaction ordering perspective, creating multiple addresses (derived from public keys) and dividing the processing power between them brings no advantage to the attacker.

The MITM attacks intercept the connection between two hosts and, being capable of changing the content of the messages, pretend to be the other to each one of them. It is effective against systems that fails to check the message signature or to validate the public key. As the Sybil attack, it can also be launched against many different layers, like the Ethereum network, E-ControlChain transactions and requests for devices. All messages exchanged by the Ethereum nodes or E-ControlChain transactions have to be signed with the private key of the requester in order to be a valid message or transaction. Thus, a MITM attacker cannot modify the message or transaction content without invalidating the signature and cannot produce a new valid signature unless the original sender's private key was compromised. The last case, MITM in requests for devices are out of E-ControlChain's scope, however it can be solved by the device taking four security steps when receiving a request. First, check if the request is correctly signed by the pointed public key. Second, extract the address from public key. Third, check the permissions in E-ControlChain using this address. Also, when using a support device, verify if the response from it is signed and check the validity of this keys against some service of validation, for example, an certification authority.

It is important to highlight that, although there is no spoofing in nodes and entities in normal conditions, many E-ControlChain transactions use addresses in policies definitions. Thus, if the attacker finds a way to swap an address by its own (for example), it can obtain improper advantages. The process of acquiring and informing addresses is also outside of the E-ControlChain scope, however, a good start could be establishing reliable communication links.

Replay attacks consists of recording exchanged messages and replaying them when convenient for the attacker, *i.e.* re-sending them in the communication channel to cause abnormal behaviors. The Replay attack achieves success when re-sending a copy of the recorded message triggers an operation that was not supposed to occur. Thus, it is specially effective against systems that does not employ the one-time message policy. Fortunately, Ethereum transactions [151] use nonce in transactions and each transaction generated by a specific address has to employ a unique nonce. Being so, if an address generate more than one transaction with a single nonce, when one of them appear in the Blockchain, the others are automatically invalidated. Also, Ethereum developers already fortified the protection against replay attacks between parallel chains with the EIP 155 [130]. However, it seems that replay attacks can still occur in specific scenarios as demonstrated in [129], where messages directed to one contract can be replayed in another one.

In short, to avoid spoofing attacks, the E-ControlChain makes use of Ethereum public keys, fingerprints and nonce. Each E-ControlChain's entity has a public key and all transactions interacting with it have to be signed using it and have a one-time nonce. Furthermore, all E-ControlChain functions verifies the requester identity and uses it to avoid unauthorized policies modifications before making these changes. besides this, all network participants, including miners, have a fingerprint known as "enode" that is the hexadecimal format of its public key. These enode identities can compose a list of static and trusted nodes [152]. Static nodes are the ones to always maintain a connection. Trusted nodes are the ones that connections can be established at any time even if above the peer limit (maximum quantity of connected peers in a given moment). Furthermore, all communication packages between peers are signed using its keys using the RLPx encryption [153]. Therefore, there are two scenarios where an spoof of identify could occur. The first is if the entity's private key gets disclosed (E-ControlChain assumes that it is kept safe). The second is breaking the Ethereum's cryptography algorithms and fabricate a private key that makes a pair with the target public key. This is an extremely improbable achievement for the current technology and, to the best of our knowledge, never happened for the adopted cryptography algorithms. Also, if that happen, the majority, if not all, of cryptography-dependent systems are condemned.

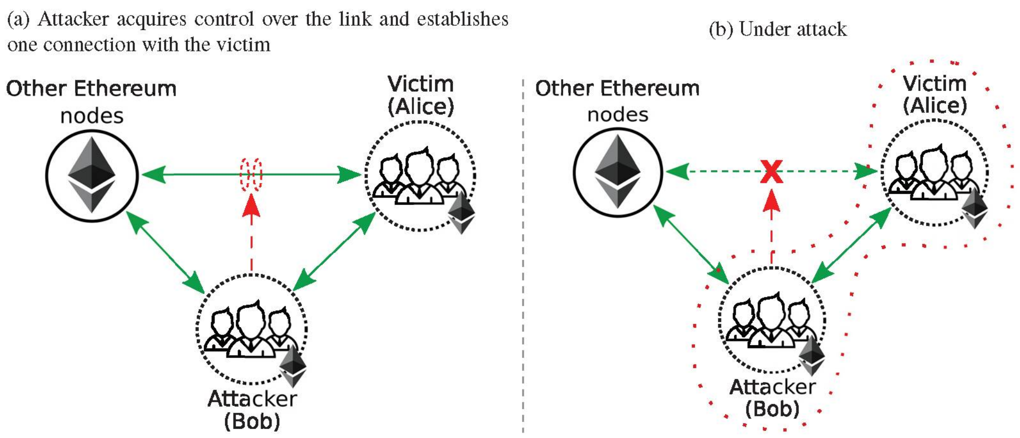
Tampering. Taking E-ControlChain context, the tampering can be the creation, change or removal of policies, its information or, even, devices. As the E-ControlChain runs over the Ethereum, all information is stored in its Blockchain and also inherits all its limitations.

A general Blockchain flaw is its Internet dependency. This turns it vulnerable to eclipse attacks [150] and Border Gateway Protocol (BGP) hijackings. Theses attacks can isolate one or more participants from the rest of the network and make the communication between the isolated part and the rest of the network vulnerable, by controlling which messages can be sent to and received from the connected portion. It gives the attacker the power to trick the attacked participants, for example, it can keep them only with outdated data. besides this, if new data gets published on the isolated portion, it will probably be erased as soon as the Internet connection is reestablished as the connected portion commonly spent more computing power in its branch than the isolated one. Also, this is one of the conditions where double spending problems could arise. The use of redundant Internet links can reduce the probability of this attacks occurring. The potential of this attack can be extended if running it together with the next attack.

The Blockchain is known to keep immutable data. However, under certain circumstances, like under 51% attack, its data become vulnerable. This attack consists of controlling more than 50% of the network hash power to define what new transactions will be part of the Blockchain or to replace past "immutable" blocks with chosen transactions. This can be used to, for example, remove data authorizing or denying access to entities. It is important to highlight that this does not mean that the attacker can forge transactions, it only has the power to select the ones that will be stored in the Blockchain. This attack can be very hard to achieve, specially when trying to replace blocks. According to the Blockchain nature, to replace a Block, the attacker needs to mine all the Blocks after it sequentially and make his or her branch have more computing power employed than the original one. Keep in mind that, in this meantime, the honest network miners keep working and growing its official branch. This makes this attack very expensive, specially for wide used network [154], like Bitcoin and Ethereum, however it is still possible [155].

Other type of tampering involves physical attacks. It occurs when someone physically interferes with device operation, either through vandalism, thievery, inserting/removing components, or extracting/forging information from/to it. Increasing the surveillance or the protection of devices can reduce the chance of its occurrence.

Figure 5.12: Eclipse attack scenario



To exemplify a tampering in the Ethereum, and consequently on E-ControlChain data, we created the scenario depicted in Figure 5.12. There are three entities: the attacker (Bob), the victim (Alice) and a the rest of the Ethereum network. In this scenario, Bob has the capability of disrupting the communication channel between Alice and all the other Ethereum through an Eclipse attack. As previously explained, triggering this attack, Bob can make Alice loose her connectivity with any other external nodes immediately. Although, through this attack, Bob cannot create fictitious message on behalf of others as he cannot forge the required signature, Bob can control any message from the network to Alice and vice versa. This is enough to make a lot of mess in Alice's life. First, preventing Alice from receiving updates from the Ethereum network. Second, preventing the Alice's updates from reaching the Ethereum network. Third, launch a miner in the isolated network to make Alice believe that "It's all god, man". Note that, while the first two only causes connectivity problems between the two isolated network, the last causes synchronization problems and tries to trick the victim. Next, we show the connectivity and synchronization problems occurring.

To simplify the demonstration of the problems, we added a global integer variable and two functions (a getter and a setter) to our contract that can be called by anyone:

```
...
int public important_value;

function setValue(int value) public {
  important_value = value;
}

function getValue() public view returns(int) {
  return important_value;
}
...
```

After setting a value for the variable and waiting for the resulting transaction to be mined we can verify that the synchronization occurs as expected:

```
// A generic node of the network executes
setValue(20);
// Waiting until the transaction gets mined
// Any node, including Alice, executes
getValue();
  <20>
```

When Bob starts the eclipse attack, the communication between Alice and other Ethereum nodes are lost. Being so, Alice will fail if she try to change any information from the contract:

```
// Bob starts the eclipse attack
// Alice executes
setValue(3);
// Waiting a long time expecting to get the transaction mined
// Any node, including Alice, executes
getValue();
  <20>
```

When Alice tries to set a new value, she creates a transaction, however, as she is under Bob's eclipse attack, her transactions does not reach any miner. They get trapped in her Ethereum client waiting for the reestablishment of the connection. The transaction is received by miners as soon as Bob ceases the attack:

```
// Bob ceases the eclipse attack
// Waiting until the transaction gets mined
// Any node, including Alice, executes
getValue();
  <3>
```

Note that if any other node changed the value trough the presented function while Alice was under attack, the Alice's values will replace it after Bob ceases the attack.

If Bob starts the eclipse attack again, Alice will return to the isolated condition. At this time, lets show what happens if a generic node in the set "other Ethereum nodes", tries to change the value under the attack situation:

```
// Any node, including Alice, executes
getValue();
```

```
  <3>
// Bob starts the eclipse attack
// Any node, but Alice, executes
setValue(89);
// Waiting until the transaction gets mined
// Any node, but Alice, executes
getValue();
  <89>
// Alice executes
getValue();
  <3>
```

As can be seen, in the same way the network does not see Alice's operations, Alice also does not see the network ones, unless Bob let they. Furthermore, Bob can also choose what operations each one will see. However, it is important to note that, if Bob wants Alice to take notice of an Ethereum network's operation, he will need let she synchronizes her Blockchain, at least, until the block where the operation was mined. besides this, as each account has a strictly incremental nonce that increases one by one and that has to be present in the transaction to make it valid, in order to Bob let a transaction from Alice gets mined, he has to let all the transactions with a smaller nonce also reach network's miners.

The last problem we will discuss is if Bob creates a private miner, starts the eclipse attack, establish the connection with Alice and then tricks her to think all her transactions are being mined by the Ethereum network. The demonstration of what happens in this case is exposed in the following listing:

```
// Any node, including Alice, executes
getValue();
  <89>
// Bob starts the eclipse attack, creates the private miner and
    ↪ establish an connection with Alice
// Alice executes
setValue(30);
// Waiting until the transaction gets mined (by the private miner)
// Alice executes
getValue();
  <30>
// Any node, but Alice, executes
getValue();
  <89>
// Bob ceases the attack
// Any node, including Alice, executes
getValue();
  <89>
```

Note that, in this case, all transactions mined by the private miner will, normally, be replaced by the network blocks and, by default, will be lost. This occurs because the blocks of the network would have more effort employed in its construction. Thus, all honest nodes will choose it. We believe that this is worse than the synchronization problem as it can trick Alice to think that all its transactions are being executed when it is not. This could lead to security issues and vulnerabilities and can make the attack harder to detect. In the majority of cases, the monitoring of changes in hash difficult can help detect it as the attacker, normally, will not have an equal quantity of hash power as the network available for tricking the victim for long periods of time.

Repudiation. The repudiation is when someone can claim that he or she did not performed an action although the system he or she as its author. A system with such flaw is a non-reliable source of information and is doomed to loose all its credibility.

One of the most efficient way to avoid and fight against this type of flaw is using public key signature algorithms. It involves three phases: (1) initialization of keys; (2) signing of messages; (3) checking of signatures. In the first phase, a secret key (SK) and a public key (PK) are generated. Commonly, the PK is create from the SK. The PK has to be public while the SK has to be kept safe. The second phase occurs after the message (transaction, in our case) is generated. Using the SK and a criptographic algorithm, a signature is created and attached to the message. The third phase occurs when a signed message is received by an interested party (IP). The IP can use the criptographic algorithm and the PK to check if the message was officially created by its sender. By the principles of public signatures, only who has the SK can create a signature that can be proved using the corresponding PK.

The Ethereum network protects itself from repudiation requiring each transaction to be signed using Elliptic Curve Digital Signature Algorithm (ECDSA) [156]. As all interactions with E-ControlChain are via Ethereum transactions, it is also protected by the same security mechanism. Thus, if the secret key is safe, no one is capable of signing a message as another participant or forge a signature or, at least, no security issue with the signature mechanism were made public. Furthermore, many E-ControlChain functions depend on the sender's address. It is computed only after the signature is verified. Thus, according to the principle of public signatures, we can trust the computed address is in fact the transaction sender's one. Finally, it is extremely improbable (although possible as the address is a hash), that any other PK generates an equal address.

Information disclosure. In Ethereum, the addresses provide only a pseudo anonymity and all transactions are public, *i.e.* readable by anyone. Thus, any participant can follow all the actions taken by an address in the network, like called functions and its parameters, although there are exceptions like queries and the execution of local functions (as checking for access authorizations). Bijective functions can be applied to values stored in contracts, like the context values in E-ControlChain, to make the deduction by third-party more difficult. However, in this case, the bijective functions have to be also mirrored into access policies.

Although no one knows for sure who or which device is behind an address in the Ethereum, transactions involving this addresses are traceable and profiling tools could be used to try to identify them [157]. One approach is the Dusting attack [158]. It consists of sending a small quantity of coins to a big quantity of addresses. Then, wait for their moving, to try to group them by their wallets and identify the users or companies behind these wallets and, then, executing for example directed phishing attacks against the identified target. To reduce the potential of the profiling tools, two common techniques are changing the own address every time a payment transaction is executed and use chaff coins (called mixins) in transactions. Because of the specificity of E-ControlChain, the use of different addresses for each operation is near impracticable as it, normally, would require many function calls to configure the new address and this action also would be easily identified by profiling tools. However, mixins-similar strategies are possible and could be easily employed to confuse profiling tools. This can done by creating fake entities, interacting with them as if they were real and making them generate realistic transactions.

Finally but not least, users are always susceptible to phishing and social engineering. They can be executed through e-mail, calls, text messages or in person. Although can be difficult to identify who is behind an account, other attacks like the Dusting Attack [158] can help to reduce the possibilities. To avoid them, specialized tools and training can be applied to identify and block it. Also, the training of users could help to identify and ignore these attacks.

Denial of service (DoS). The objective of DoS attacks is to deny or to degrade service provision to legitimate users, spoiling the system availability. It does that by exhausting all the resources available and, consequently, let the legitimate requests starving. However, as the Ethereum is a decentralized platform, conventional DoS attacks to completely disrupt the network would require the stop of Internet with unthinkable levels of traffic generation or serious flaws in the Internet's core equipment. Of course, small groups of users or devices are still vulnerable. Thus, for E-ControlChain, there are two perspectives of DoS attacks we think is worth mentioning: the network's and the contracts'.

From the `perspective of the network`, the E-ControlChain runs over the Ethereum and, therefore, it inherits all limitations from lower layers and middlewares. In this analysis, we explore internet dependency, outperforming of hash power and storage requirement.

A fundamental aspect of conventional Blockchains is the necessity of Internet connectivity to synchronize the data and keep all participants with an equal snapshot. If it is faulty for some reason like eclipse attacks [150] or Border Gateway Protocol (BGP) hijackings, the data holded by the isolated portion of the internet will become obsolete because changes in the connected part will not be reflected in the isolated part. Also, similar to what happens on 51% attack, if new data gets published on the isolated portion, it is probable that it will be erased as soon as the connection between the to parts is reestablished. This occurs because the Ethereum does not support Blockchain branches and will choose the one with more effort employed in its construction as the official one. Normally, this represent the constructed in the Internet portion because it has a bigger power processing capability. Also, this is one of the conditions where double spending problems could arise. Therefore, for critical services, we recommended the use of redundant Internet links.

Although Ethereum is trying to move from proof-of-work to proof-of-stake [124, 159], it is, until the current time, based on proof-of-work. The main weakness known in proof-of-work protection is the 51% attack. It defines that if an attacker succeed to control over 50% of the network hash power it can undo transactions constructing an alternative branch that has more processing employed in it than the original. A successful attack of this type against the Ethereum was launched on beginning of the year 2019 [155], allowing the so called double-spend problem. However, it is a very cost attack, for example, one hour of 51% attack against Ethereum cost approximately USD 98,572 [160].

The adoption of private or permissioned Blockchains could, in one hand, minimize the attack surface, however these networks have less processing power or are managed by only a few participants. Therefore, it is much more easy for an attacker to take control over the network. In the official Ethereum, the attacker has to overcome the all miners processing power or create a collusion between them. However, miners has the incentive to avoid collusion between themselves as it could drop the Ethereum cryptocurrency value and, consequently, their incoming.

Devices that directly use the E-ControlChain also have to deal with the increasing Ethereum's storage demand. [14] achieved a maximum increase of 2.7GB per day in its experiment while the current (oct. 28, 2019) increase of the Ethereum is around 0.5GB per day [161]. Although there are synchronizations less space-hungry, some constrained devices will end up depending on storage strategies (like distributed storage) or supporting devices. In its experiment, [14] also showed that a Raspberry Pi 3 B+ working as a supporting device is capable of serving around 30 authorizations consults per second using a query software implemented in JavaScript. Thus, for demanding scenarios, more powerful machines serving as authorization support are required. Finally, to reduce DoS attacks attempt over the storage, the Ethereum network sets a hard blocks size and a variable minimum hash difficult for each Block. These settings limit the data rate production to a constant (in average) velocity.

From the `perspective of contracts`, they are vulnerable to at least two types of Denial of Service [162]: "DoS with (Unexpected) Throw" and "DoS with Block Gas Limit".

The first occurs when the continuity of the contract requires the execution of an instruction that always causes a throw and, thus, never gets executed. The second occurs when the processing of an call requires more gas than can be provided for it. These types of DoS are caused by flaws in the code of the contracts and not always are triggered by attackers.

Elevation of privilege. The elevation of privilege occurs when someone manage to achieve authorization beyond those initially granted [163]. This attack can occur, for example, in contracts vulnerable to re-entrancy attack. The re-entrancy attack uses external contracts to trick the original ones to re-execute parts of the code they weren't supposed to, at least, not in the order that they are executed. In another words, this occurs when the called contract makes a call to an external one and it manages to take over the control flow [164].

5.4.1 Application vulnerabilities

Blockchain applications (called smart contracts) are stored in a similar way as the transactions and being so there is not an easy way to update them [155]. Therefore, it is recommended that it passes a carefully security analysis. For access control smart contracts, algorithmic vulnerabilities exploitation can lead to the insertion of unauthorized data into the authorization system, making it take wrong decisions in the authorization process. This section, at the same time, analyzes the E-ControlChain and presents some directions on how to avoid vulnerabilities in smart contracts.

First, all entry points of interaction has to be protected by clear permission limits. In Ethereum, all interactions are done through function calls. Thus, each function, mainly the public ones (that can be called from outside of the contract), have to define who can call them according to its parameters. The most common way of function call protection is through the function "require". It can verify inputs and test them against any Boolean validation. Furthermore, a function can be protected through modifiers that define commonly used require statements, *i.e.* when the same protection has to be applied to many functions.

E-ControlChain protects each one of its public functions that can be used to modify information policy that does not belong to the caller. The function caller is identified through the address extracted from the public key used in the message signature. This address is, then, used in the verification of the function call permission. For example, in the current version (as showed in line 12 of the algorithm on Listing 5.1), when the entity e_1 tries to change the entity e_2's owner, E-ControlChain only accept the request if e_1 is the current owner of e_2, or if e_1 signed the message and e_2 don't have an owner yet.

Another example of function call permission verification occurs when the entity tries to set an context. E-ControlChain `setContext` function has a modifier (line 18) called `isTheAddreessOrItsOwner`. This modifier verifies if the function caller entity is the context source itself or its owner (line 7). This avoids the forgery of contexts by malicious devices. As the `setOwner` and `setContext` functions, all the other E-ControlChain's ones have a clear limitation of who can execute it.

Improper function calls can also interfere with the system normal operation. In this aspect, the E-ControlChain `setOwners` function has what can be considered a flaw. It allows anyone to generate numerous addresses and take control of them, becoming the owner of them. However, it requires one call per address and transactions has a cost. As there are 2^{160} available addresses, the cost for someone become owner of a significant quantity of addresses is prohibitive.

There are also many other vulnerabilities that are generated by mistakes in the development. Some of them are: (1) integer underflow, (2) integer overflow, (3) parity multisig bug, (4) call stack depth attack, (5) transaction ordering dependency, (6) timestamp dependency, (7) re-entracy. Some of them can appear using simple features like arithmetic operations, like (1)

Listing 5.1: E-ControlChain - setOwner function

```
Contract BasicAndCommonControl {
  mapping(address => address) public owners;
  mapping(address => bool) public approved_ownerships;
  mapping(address => mapping (string => int)) context;
  ...
  modifier isTheAddressOrItsOwner(address addr) {
    require(msg.sender == addr || msg.sender == owners[addr]);
    _;
  }
  ...
  function setOwner(address addr, address new_owner) public {
    require((owners[addr] == address(0) && msg.sender == addr) || (owners[addr] != address(0)
        ↪ && msg.sender == owners[addr]));
    approved_ownerships[addr] = false;
     rules_authorities [addr] = address(0);
    owners[addr] = new_owner;
  }
  ...
  function setContext(address source, string identifier, int value) public
      ↪ isTheAddressOrItsOwner(source) {
    context[source][ identifier ] = value;
  }
}
```

and (2), while others require more complex features, for example, nesting of contracts or function recursion, like (3), (4) and (7).

`Integer underflow` and `integer overflow` occurs when an arithmetic operation tries to create a number outside of the representation range of the variable [165]. For example, the uint8, in solidity, can represent a number ranging from zero to 255. So, if a variable *x* of the type uint8 receives a value of 255 and, after, we add one to it, instead of containing a value of 256 as expected, it will contain a value of zero. This problem is called integer overflow. The integer underflow occurs in a similar way, if you have the same variable containing the value zero and tries to subtract one from it, it will contain 255. This vulnerability can be avoided by verifying if the result is inside the variable representation range before executing the operation. An example of insecure and secure algorithm against these vulnerabilities is presented in Listing 5.2.

The `parity multisig bug` can occur when a contract uses another one as a library and has a fall back function that redirects any unknown call to it. Depending on the functions available in the library contract and how they affect the main one, the fall back function can be used to execute unauthorized actions or take control over the main contract if they (main and library contracts) aren't protected. [167] demonstrated how a contract can be affected by this vulnerability. Suppose the contract "Library" (Listing 5.3) is used as a library in contract "Main" (Listing 5.4). If someone call the function "initLibrary" on contract "Main" it will end up in the fall back function that, in turn, will call the "initLibrary" from the contract "Library", changing the owner of the contract "Main". Therefore, exploiting the parity multisig bug present in this example, anyone can change the owner of the contract and, then, execute any other function that is only available for the owner.

There are many ways to avoid the parity multisig bug attack in the example. First, the function "initLibrary" could be marked as internal, *i.e.* it would only be called by the code running

Listing 5.2: Example of insecure and secure algorithm against integer overflow (adapted from [166])

```
mapping (address => uint256) public balanceOf;

// INSECURE
function transfer (address _to, uint256 _value) {
  // Checks for underflow only
  require (balanceOf[msg.sender] >= _value);
  // Add and subtract new balances
  balanceOf[msg.sender] −= _value;
  balanceOf[_to] += _value;
}

// SECURE
function transfer (address _to, uint256 _value) {
  // Checks for underflow and for overflows
  require (balanceOf[msg.sender] >= _value && balanceOf[_to] + _value >= balanceOf[_to]);
  // Add and subtract new balances
  balanceOf[msg.sender] −= _value;
  balanceOf[_to] += _value;
}
```

Listing 5.3: Example of library that allows the parity multisig bug (adapted from [167])

```
Contract Library {
  address owner;

  // called by the constructor
  function initLibrary (address _owner) {
    owner = _owner
  }
  ...
}
```

Listing 5.4: Example of contract that can suffer from the parity multisig bug (adapted from [167])

```
Contract Main {
  address owner;
  address _library;

  // called by the constructor
  function Main(address _owner) {
    _library = <address of contract Library>;
    _library . delegatecall (bytes4(sha3("initLibrary(address)")), _owner);
  }
  ... // that does not contain a function called initLibrary
  function () payable {
    _library . delegatecall (msg.data);
  }
}
```

Listing 5.5: Example of call stack depth attack (example taken from [170])

```
function CallstackExploit (int counter) {
  if (counter < 1023) {
    if (counter > 0) {
      self.CallstackExploit.gas(msg.gas-2000)(counter+1);
    } else {
      self.CallstackExploit(counter+1);
    }
  } else {
    // finally call a function in another contract after calling self.CallstackExploit 1023
        ↪ times
  }
}
```

at the current address (contract) or contracts derived from it. Second, the function "initLibrary" could have a checking for double initialization, *i.e.* verify if the owner weren't defined yet before defining it. Furthermore, as the E-ControlChain is formed solely by a derivation of contracts and all interactions are to be made directly to the composing contracts, this vulnerability does not affect it.

The `call stack depth attack` exploited the big limit of the call stack. Its hard limit is 1024 and this means that could exist up until 1023 nested function at any algorithm execution point. Taking advantage of that and using computationally difficult and inexpensive operations, attackers achieved to create DDoS-like attacks [168]. This type of attack filled the network with pending transactions which caused users delays in processing their transactions.

An example of how it worked is presented in Listing 5.5. Fortunately, this vulnerability already was solved at the Ethereum Improvement Proposal (EIP) 150 [169]. This EIP increased the cost for some operations and limited the maximum allowed amount of gas for a child call to 63/64 of the parent.

The `transaction ordering dependency` can cause a racing condition attack [171]. It occurs when someone can take advantage based on the order that the transactions are mined. For example, suppose the contract presented in the Listing 5.6, suppose that a buyer can the function "buy" when the price is at 100. If the contract owner manage to mine a transaction calling the function "setPrice" with a higher price before the buyer's transaction, the buyer will pay a higher price than the one at the transaction generation.

A solution for this example could be setting a versioning for the price. Each time the owner changes de price the a counter (version number) increase. When a buyer buy the asset it determines the current price version. The purchase only gets performed if the version determined by the buyer is the current version. Otherwise, it will not be performed. Finally, we did not find any way to perform this attack in E-ControlChain.

The `timestamp dependency` attack is similar to the transaction ordering dependency attack, however, instead of using the order to affect the result, it uses the block timestamp. Although a new block must have a bigger timestamp than the last mined block and blocks with a timestamp more than 15 seconds in the future are not accepted, the miners has some manipulation malleability over it [172]. Therefore, any contract that uses the timestamp in critical functionalities, like using it as seed for a lottery, can be affected by this attack as the miner can manipulate it to favor its earnings. E-ControlChain does not use timestamp in any of its functions.

The `re-entrancy` attack uses external contracts to trick smart contracts to re-execute parts of the code that weren't supposed to execute, at least, not in the order that they are executed.

Listing 5.6: Example of transaction ordering dependency attack (example taken from [171])

```
contract TransactionOrdering {
  uint256 price;
  address owner;
  event Purchase(address _buyer, uint256 _price);
  event PriceChange(address _owner, uint256 _price);
  modifier ownerOnly() {
    require(msg.sender == owner);
    _;
  }
  function TransactionOrdering() {
    owner = msg.sender;
    price = 100;
  }
  function buy() returns (uint256) {
    Purchase(msg.sender, price);
    return price;
  }
  function setPrice(uint256 _price) ownerOnly() {
    price = _price;
    PriceChange(owner, price);
  }
}
```

This occurs when an external contract take over the control flow [164]. Take as example the contract "HoneyPot" (HP) presented in Listing 5.7. It exposes two main functions, one to store a value in an account ("put") and other to withdraw this value ("get"). However, the function to withdraw this value has a vulnerability. It first sends the amount to the external account and only after it, removes the balance for the account. This vulnerability can be exploited by the contract "HoneyPotCollect" (HPC) presented in the Listing 5.8, as it can iterate through this function without letting it update the balance.

The HPC has two main functions, one to trigger the attack, "collect()", and the other to maintain the attack, the fallback function. The function "collect" starts putting some value in the HP where the credited account is the calling one, *i.e.* the HPC address. Its value defines the amount that will be withdrawed in each iteration of the attack. After this, it triggers the attack calling the function "get" of the HP. Note that, again, the caller address is the HPC's. Up receiving the call the HP sends the amount to the HPC. However, before letting HP updates the balance, this triggers the fallback function of the HPC. Note that at this moment the external contract, *i.e.* HPC, takes the control of the execution flow. The fallback function call again the function "get" of the HP, that sends the HPC's account balance to it, triggering again the fallback function. In this example, this only ends when the the HP total balance is less than the value withdrawed in it attack's iteration.

As the E-ControlChain does not have any function that can trigger functions from other contracts, it is not affected by this type of attack. Here, we discussed some of the most DApps famous attacks. [118, 166] presents many others. Furthermore, many other Blockchain generic threats are discussed in [23, 155, 173, 174] and there also a classification of weaknesses that could exist in smart contracts [175].

Listing 5.7: Example of contract with the re-entrancy vulnerability (adapted from [164])

```
contract HoneyPot {
  mapping (address => uint) public balances;
   ...
  function put() payable {
    balances[msg.sender] = msg.value;
  }
  function get() {
    if (!msg.sender.call.value(balances[msg.sender])()) { // At this point, the caller can call
         ↪ get() function again
      throw;
    }
    balances[msg.sender] = 0; // In the example re-entrancy attack, it will be executed only
         ↪ when all the funds were moved from the current contract
  }
   ...
}
```

Listing 5.8: Example of contract that exploit the re-entrancy vulnerability (adapted from [164])

```
contract HoneyPotCollect {
   ...
  function collect() payable {
    honeypot.put.value(msg.value)(); // Define the amount that will be withdrawed in each
         ↪ executed loop
    honeypot.get(); // Trigers the withdraw loop
  }
  function () payable {
    if (honeypot.balance >= msg.value) {
      honeypot.get(); // When receives a payment and the target contract has balance, re-
           ↪ executes get() function
    }
  }
}
```

5.4.1.1 Auditing tools

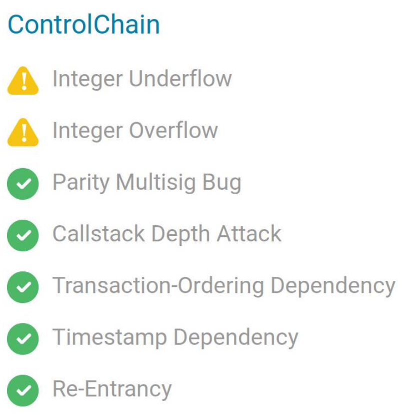
Contract vulnerabilities are not always straight forward to detect. Thus, many new companies and startups are getting their space in the market as smart contract analyzers and advisers. They mainly use artificial intelligence (AI) in the process as it makes the analyzes more cheaper and faster. However, as some times the AI cannot achieve the same level of precision from a specialist, many of these companies also offer the option of hiring specialists in smart contracts to analyze them, like AnChain.AI [176] and ChainSecurity [177].

We tried to analyzed the E-ControlChain with the tools AnChain.AI [176], Securify [178], ContractGuard [179], Smartcheck [180], Slither [181], Oyente [182], Manticore [183], ContractFuzzer [184], Mythril [185] and Octopus [186]. Next, we discuss the faced problems and the reports. We also present the full report on Appendix C.

AnChain.AI. The AnChain.AI identified two flaws[1] as shown by Figure 5.13: integer underflow and integer overflow. After a manual analysis, we found that it exists, however, it only affects the user who passed the wrong value to the function as the information from it is linked to user's address. The only real problem we identified is if someone wants to work with context values outside the range of the integer representation, *i.e.* outside the range $[-2^{31}, 2^{31} - 1]$.

As users can insert information into the arrays, another theoretical problem is the possibility of growing them bigger than 2^{256} elements. However, this is almost impossible to occur as it would lead to two bigger problems first: the higher cost involved in the process and the required storage capacity (note that if each element is 1B size, it would be necessary around 10^{53} yottabytes to store all elements).

Figure 5.13: E-ControlChain analysis by AnChain.AI


ControlChain

⚠ Integer Underflow

⚠ Integer Overflow

✓ Parity Multisig Bug

✓ Callstack Depth Attack

✓ Transaction-Ordering Dependency

✓ Timestamp Dependency

✓ Re-Entrancy

Securify. Securify pointed out two types of issues: "unrestricted write to storage" and "Locked Ether". The first seems to be only false-positives as one part of the pointed functions use modifiers or statements "require" to protect its execution and the other part use mappings that are initially mapped to "msg.sender", and therefore, affecting only the user itself. The second pointed vulnerability, Locked Ether, means that once E-ControlChain receive Ethers transferred by the network, it does not has a function to withdraw this Ethers, thus they will be locked on the contract address. As there is not payable methods in E-ControlChain, it only receive Ethers in

[1]Unfortunately, AnChain.AI is not working anymore to test E-ControlChain. After we tested it, we updated the it to Solidity v0.5. So we believe that perhaps AnChain.AI cannot deal correctly with the new version although it is listed as compatible. We verified and executed all the items from the troubleshooting list message, but contacting the helpdesk (as it is a paid feature), however none of it makes it work. Being so, we don't have the full report, however we present the results we still have from when we tested the E-ControlChain with Solidity v0.4.23.

very rare and special conditions. Furthermore, working with Ethers is not one of its function, thus, in the worst scenario, it would only lock Ethers sent by the network.

ContractGuard. The current version of ContractGuard only supports Solidity v0.5.10 and, therefore, is unable to handle v0.5.11. So, we modified E-ControlChain code to use Solidity v0.5.10. After it, the ContractGuard was capable of making its analyzes. The result was that it found no errors in E-ControlChain.

Smartcheck. The Smartcheck issued the following warnings: "Costly loop" and "Prefer external to public visibility". The first is caused by the iteration loops inside some functions, mainly to work with the rules and its contexts. This flaw can lead to a problem in some functions, for example if an address register a big quantity of rules or a big quantity of context or actions inside a rule. However, as this data is separated by entity's addresses, in the worst scenario, only the entity who did it will be affected by the problem. The second, prefer external to public visibility, warning suggest the use of external visibility whenever it is possible. We did not used it in one function where it, in theory, was possible, the "deleteAttributeRule". The reason is that the visibility external requires the method's parameter be located in "calldata", however this makes it use more stack variables and this overflow the maximum variables allowed for a function in Ethereum. Furthermore, in the worst scenario, it only causes a little increase in the cost of the function execution as instead of pulling the data from the calldata, it starts copying it into the memory.

Slither. The current version of Slither only supports Solidity v0.5.9 and, therefore, is unable to handle v0.5.11. So, we modified E-ControlChain code to use Solidity v0.5.9. After it, the Slither was capable of making its analyzes. The result was three warning types: unused return values, too recent pragma version, and public function that could be declared as external. The first was triggered by a function pop called to remove an item from the array. This item is useless after its removal, so this warning can be seem as a false-positive. The second warn type, too recent pragma version, suggest the use of a prior and possibly more trusty version of Solidity, the v0.5.3. The third warn type, declare as external, is caused by the reason explained for the Smarcheck's warn "Prefer external to public visibility".

Unfortunately, the tools Oyente, Manticore, ContractFuzzer, Mythril and Octopus did not run successfully. Oyente and Manticore was not compatible with Solidity v0.5. Oyente, for example, is compatible only with v0.4.19, while manticore seems to be compatible only with v0.4.23. We could not achieve the docker image from ContractFuzzer. Mythril exceed the 6GB of RAM we have in our disposal and did not finish its execution.

In Octopus, we tried to execute both tasks, the control flow graph (CFG) and the SSA. The CFG generated a PDF with a list of more than 300 rectangle blocks side by side each one with the writing "block_H", where "H" is an apparently unique hexadecimal number. Also, there was no link between the rectangles. The SSA did not finished its execution as it exceeded our limit of 6GB of RAM. The scenario repeated itself even setting the options "–simplify –onlystatic", that would make a more superficial analysis.

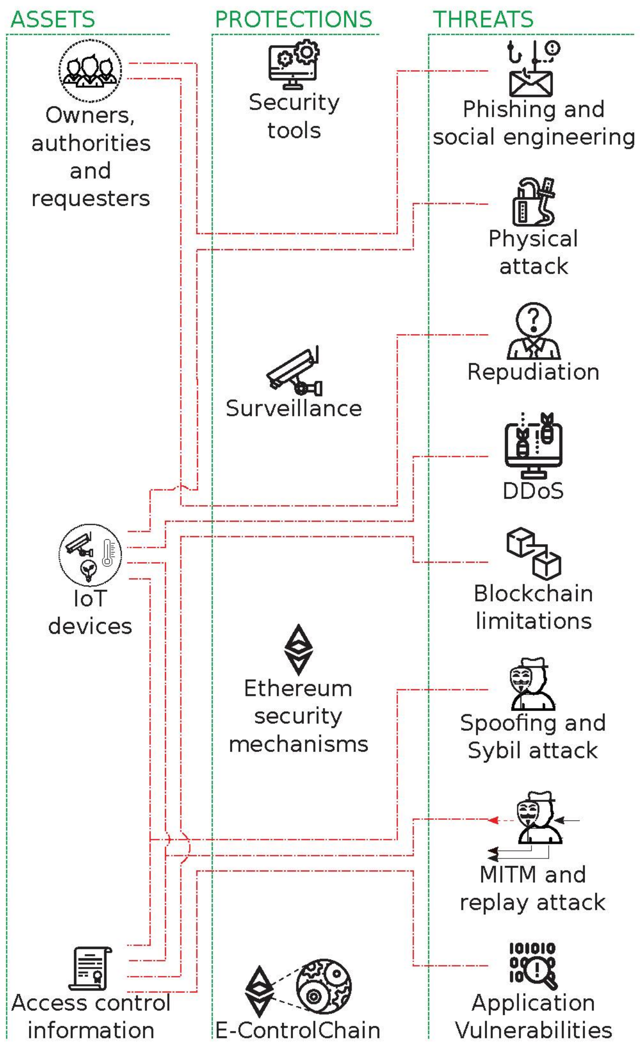
Finally, it is important to highlight that we had encountered no bigger vulnerability does not mean they do not exist. As in any system, new and exotic ways of exploiting vulnerabilities in smart contract can be discovered and all contracts are susceptible to them. Therefore, security analysis is a continuous task.

5.4.2 Overview

For the overview of threats, we created a high-level threat model, presented in the Figure 5.14, that cover, in categories, the threats discussed in this section. It is composed by eight vector of threats: phishing and social engineering; physical attack; Repudiation; DDoS;

Blockchain limitations; Spoofing and Sybil attack; Man In The Middle (MITM) and replay attack; and application vulnerabilities. We summarize the discussion in Table 5.6, showing what gets affected by each threats and techniques that can be used to avoid them. Furthermore, we also summarize the possible vulnerabilities and warning reported by the auditing tools in Table 5.7.

Figure 5.14: Threat model


ASSETS
PROTECTIONS
THREATS
Owners, authorities and requesters
IoT devices
Access control information
Security tools
Surveillance
Ethereum security mechanisms
E-ControlChain
Phishing and social engineering
Physical attack
Repudiation
DDoS
Blockchain limitations
Spoofing and Sybil attack
MITM and replay attack
Application Vulnerabilities

5.5 CONCLUSION

In this chapter, we presented the E-ControlChain, a proof-of-concept for the ControlChain architecture. It was design to run over the Ethereum platform, a Blockchain network that is capable of running Turing-complete code. The E-ControlChain is a contract that exposes many functions that allows users to interact with access control information. After presenting the main possible E-ControlChain interactions, we described some application scenarios, discussed the E-ControlChain cost official Ethereum chain, tested it against an IoT environment with IoT devices, discussed its limitations, and, finally, presented a security analysis.

The experiment scenario counted with a Raspberry Pi and evaluated the E-ControlChain usage of processing, memory, disk and network. From the data obtained on the experiment, it can be considered overwhelming to a severely limited device. However, it is possible to ensure its viability in IoT's scenarios as a simple Raspberry Pi could deal with it and also give support to many more restricted devices that cannot.

Table 5.6: Summarized security analysis of the E-ControlChain components.

Threat	**What would be affected**	**Avoidance or mitigation techniques and methods**
Phishing and social engineering	users in general (owners, authorities, requesters)	Train and instruct users; use specialized tools to detect such attacks
Physical attack	IoT devices	Protect and use surveillance monitoring of the IoT device location
Repudiation	The trust of users in general	Require recognizable cryptography signature in all transactions and messages
DDoS	Access control data synchronization, CPU and network usage, slowness in operations on the DDoS attack local	Adopt firewall, routing and other DDoS filtering techniques
Blockchain limitations	Access control data synchronization and storage	Multiple network links; employ devices with large storage capacity to store access control information
Sybil attack	Access control policies in general, privacy and communication security	Protect the private keys; adopt strong cryptography algorithms
MITM	Access control policies in general, privacy and communication security	Use certification authorities or any other reliable mechanism to validate public keys
replay attack	Access control policies in general, privacy	Adopt signed transactions with a single-usage nonce, and identify the chain and the contract in each transaction
Application vulnerabilities	Access control policies in general	Clear and well defined restrictions in contract functions; avoid the usage of new structures or technologies that has not yet been extensively tested

Table 5.7: Summarized possible vulnerabilities and warning reported by the auditing tools.

Vulnerability\Tool	AnChain.AI*	Securify	ContractGuard**	Smartcheck	Slither
Integer overflow	+				
Integer underflow	+				
Unrestricted write to storage		+			
Locked Ether		+			
Costly loop				+	
Prefer external to public visibility				+	+
Unused return value					+
Too recent pragma version					+

* Previous results for E-ControlChain with Solidity 0.4.23.
** Did not report any problem.

The limitation discussion is about showing and justifying most of the E-ControlChain limitations. It revealed three main axes that generated them. The axes are based on design, base components and algorithm decision.

Finally, we made a gathering of a wide range of E-ControlChain threats that can affect it through a global analysis, ranging from physical to the exploitation of application vulnerabilities. In application vulnerabilities, we searched for some of the most common ones and also used the AnChain.AI tool to verify our code. Furthermore, we also give a list of techniques that could prevent or avoid the threats. Finally, we also analyzed the E-ControlChain solution using the STRIDE framework and we found that although it is a very system, mainly because the Blockchain protection mechanisms, some attacks in specific scenarios can disturb its functioning.

6 CONCLUSION

Intelligent services are surrounding us as never imagined before. To achieve their full potential, they use information collected by IoT devices scattered all over the world, including parks, buildings, transportation systems, homes, bodies and many others. However, the great services that the IoT can provide came together with privacy and security concerns as many of them require the handling of private and confidential information. besides this, the recent wave of successful attacks targeting the IoT environment only increase this concerns and also reveal the vulnerability and inefficiency of the currently used access control systems. Finally, the literature review only revealed many unsuccessful attempts to fulfill this lack of a suitable access control approach.

Believing in the Blockchain unexplored potential, in this work, we joined the on going effort to change the IoT access control scenario and proposed a new access control architecture, namely ControlChain. The ControlChain is heavily based on the Blockchain technology and absorbs its decentralization capability. Besides this, it was initially designed to support at least three types of authorization (based on attributes, capabilities and ACL). Together, they are capable of providing compatibility with a wide range of already adopted IoT access control models and, therefore, can smooth the migration from older and outdated access control mechanisms. Our architecture also includes a secure way of creating relationships, assigning attributes for them and using them in the access control.

The main idea behind the ControlChain architecture was the provision of basic pillars for a simple and, at the same time, powerful access control system. This could give to IoT devices the freedom for achieving its tasks efficiently, without letting the users loose control over its owned devices and information. Furthermore, its comparison with other architectures revealed gains in many aspects, like scalability, transparency, fault tolerance, time to authorize, compatibility and others. In order to verify and demonstrate this and its viability of adoption for the IoT environment, we developed a proof-of-concept, namely E-ControlChain.

The E-ControlChain was developed as a smart contract to run over the Ethereum platform. The tests performed using a Raspberry Pi showed a relatively low resource usage and corroborates with its viability for the IoT. However, even so, E-ControlChain also gives support for more resource constrained devices through assistance of other more powerful ones. besides this, we also evaluated the security of the E-ControlChain. To this end, we created a threat model and discussed the threats and how they can became security issues for it. This initial analysis showed us that the E-ControlChain is a very robust solution being resilient to a wide variety of attacks.

Finally, part of the contributions of this work were published at conference GLOBECOM 2017 [19] and journal Concurrency and Computation: Practice and Experience [14].

REFERENCES

[1] Gartner IT Glossary. Internet of things. http://www.gartner.com/it-glossary/internet-of-things/, 2017. Visited on 25 Abr 2017.

[2] Amy Nordrum. Popular internet of things forecast of 50 billion devices by 2020 is outdated. http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated, Aug 2016. Visited on 16 Mar 2017.

[3] James Manyika and Michael Chui. By 2025, internet of things applications could have $11 trillion impact. https://www.mckinsey.com/mgi/overview/in-the-news/by-2025-internet-of-things-applications-could-have-11-trillion-impact, Jul 2015. Visited on 23 Aug 2018.

[4] Aafaf Ouaddah, Anas Abou Elkalam, and Abdellah Ait Ouahman. Fairaccess: a new blockchain-based access control framework for the internet of things. *Security and Communication Networks*, 2017. SCN-16-0184.

[5] Aafaf Ouaddah, Hajar Mousannif, Anas Abou Elkalam, and Abdellah Ait Ouahman. Access control in the internet of things: Big challenges and new opportunities. *Computer Networks*, 112:237–262, 2017.

[6] Vijayaraghavan Varadharajan and Shruti Bansal. *Data Security and Privacy in the Internet of Things (IoT) Environment*, pages 261–281. Springer International Publishing, 2016.

[7] Andy Greenberg. Hackers remotely kill a jeep on the highway-with me in it. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/, Jul 2015. Visited on 24 Aug 2017.

[8] Andy Greenberg and Kim Zetter. How the internet of things got hacked. https://www.wired.com/2015/12/2015-the-year-the-internet-of-things-got-hacked/, Dec 2015. Visited on 24 Aug 2017.

[9] Brian Krebs. Hacked cameras, dvrs powered today's massive internet outage. https://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/, Oct 2016. Visited on 3 Nov 2016.

[10] Symantec Security Response. Vpnfilter: New router malware with destructive capabilities. https://www.symantec.com/blogs/threat-intelligence/vpnfilter-iot-malware, Jun 2018. Visited on 13 Jun 2018.

[11] R. S. Sandhu and P. Samarati. Access control: Principle and practice. *Comm. Mag.*, 32(9):40–48, September 1994.

[12] Yun Zhou, Yanchao Zhang, and Yuguang Fang. Access control in wireless sensor networks. *Ad Hoc Netw*, 5(1):3–13, 1 2007.

[13] L. Seitz, G. Selander, and C. Gehrmann. Authorization framework for the internet-of-things. In *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 1–6, June 2013.

[14] O. J. A. Pinno, A. R. A. Gregio, and L. C. E. De Bona. Controlchain: a new stage on the iot access control authorization. *Concurrency and Computation: Practice and Experience*, pages 1–23, 2019.

[15] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. Medrec: Using blockchain for medical data access and permission management. In *2016 2nd International Conference on Open and Big Data (OBD)*, pages 25–30, Aug 2016.

[16] S. Fujimura, H. Watanabe, A. Nakadaira, T. Yamada, A. Akutsu, and J. J. Kishigami. Bright: A concept for a decentralized rights management system based on blockchain. In *2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pages 345–346, Sept 2015.

[17] IBM. Watson internet of things: The internet of things becomes the internet that thinks with watson iot. `https://www.ibm.com/internet-of-things/`, Aug 2017. Visited on 13 Aug 2017.

[18] Aafaf Ouaddah, Anas Abou Elkalam, and Abdellah Ait Ouahman. *Towards a Novel Privacy-Preserving Access Control Model Based on Blockchain Technology in IoT*, pages 523–533. Springer International Publishing, 2017.

[19] O. J. A. Pinno, A. R. A. Gregio, and L. C. E. De Bona. Controlchain: Blockchain as a central enabler for access control authorizations in the iot. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–6, Dec 2017.

[20] G. Zyskind, O. Nathan, and A. '. Pentland. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops*, pages 180–184, May 2015.

[21] Bitcoin. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, 2008. Visited on 11 Apr 2017.

[22] M. Swan. *Blockchain: Blueprint for a New Economy*. O'Reilly Media, 2015.

[23] ethereum. A next-generation smart contract and decentralized application platform. `https://github.com/ethereum/wiki/wiki/White-Paper`, Mar 2017. Visited on 14 Mar 2017.

[24] Georgi Georgiev. South korea to invest $230m in blockchain technology development. `https://bitcoinist.com/south-korea-230b-invest-blockchain`, Jun 2018. Visited on 25 Jun 2018.

[25] Ezedine Barka, Sujith Samuel Mathew, and Yacine Atif. *Securing the Web of Things with Role-Based Access Control*, pages 14–26. Springer International Publishing, 2015.

[26] J. Jindou, Q. Xiaofeng, and C. Cheng. Access control method for web of things based on role and sns. In *2012 IEEE 12th International Conference on Computer and Information Technology*, pages 316–321, Oct 2012.

[27] J. Liu, Y. Xiao, and C. L. P. Chen. Authentication and access control in the internet of things. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, pages 588–592, June 2012.

[28] Guoping Zhang and Jiazheng Tian. An extended role based access control model for the internet of things. In *2010 International Conference on Information, Networking and Automation (ICINA)*, pages 319–323, Oct 2010.

[29] Ning Ye, Yan Zhu, Ru chuan Wang, Reza Malekian, and Lin Qiao-min. An efficient authentication and access control scheme for perception layer of internet of things. *Applied Mathematics & Information Sciences*, 8(4):1617–1624, jul 2014.

[30] Guoping Zhang and Wentao Gong. The research of access control based on UCON in the internet of things. *J Softw*, 6(4), apr 2011.

[31] José L. Hernández-Ramos, Antonio J. Jara, Leandro Marín, and Antonio F. Skarmeta Gómez. Dcapbac: embedding authorization logic into smart things through ecc optimizations. *Math Comput Model*, 93(2):345–366, 2016.

[32] Parikshit N. Mahalle, Bayu Anggorojati, Neeli R. Prasad, and Ramjee Prasad. Identity authentication and capability based access control (iacac) for the internet of things. *Journal of Cyber Security and Mobility*, 1(4):309–348, 3 2013.

[33] Bayu Anggorojati, Parikshit N. Mahalle, Neeli R. Prasad, and Ramjee Prasad. *Secure Access Control and Authority Delegation Based on Capability and Context Awareness for Federated IoT*, pages 135–160. River Publishers, 5 2013.

[34] Sergio Gusmeroli, Salvatore Piccione, and Domenico Rotondi. A capability-based security approach to manage access control in the internet of things. *Math Comput Model*, 58(5-6):1189 – 1205, 2013.

[35] Jorge Bernal Bernabe, Jose Luis Hernandez Ramos, and Antonio F. Skarmeta Gomez. Taciot: multidimensional trust-aware access control system for the internet of things. *Soft Comput*, 20(5):1763–1779, 2016.

[36] R. Neisse, I. N. Fovino, G. Baldini, V. Stavroulaki, P. Vlacheas, and R. Giaffreda. A model-based security toolkit for the internet of things. In *2014 Ninth International Conference on Availability, Reliability and Security*, pages 78–87, Sept 2014.

[37] P. N. Mahalle, P. A. Thakre, N. R. Prasad, and R. Prasad. A fuzzy approach to trust based access control in internet of things. In *Wireless VITAE 2013*, pages 1–5, June 2013.

[38] Aafaf Ouaddah, Anas Abou Elkalam, and Abdellah Ait Ouahman. Harnessing the power of blockchain technology to solve iot security &#38; privacy issues. In *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, ICC '17, pages 7:1–7:10, New York, NY, USA, 2017. ACM.

[39] Guangdong Bai, Lin Yan, Liang Gu, Yao Guo, and Xiangqun Chen. Context-aware usage control for web of things. *Security and Communication Networks*, 7(12):2696–2712, 2014.

[40] Yi Liu, Yinghui Zhang, Jie Ling, and Zhusong Liu. Secure and fine-grained access control on e-healthcare records in mobile cloud computing. *Future Gener Comput Syst*, 2017.

[41] K. Yang, Q. Han, H. Li, K. Zheng, Z. Su, and X. Shen. An efficient and fine-grained big data access control scheme with privacy- preserving policy. *IEEE Internet of Things Journal*, 4(2):563–571, April 2017.

[42] Qian Zhou, Mohammed Elbadry, Fan Ye, and Yuanyuan Yang. Flexible, fine grained access control for internet of things: Poster abstract. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, IoTDI '17, pages 333–334, New York, NY, USA, 2017. ACM.

[43] Mia Consalvo. Using your friends: Social mechanics in social games. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, FDG '11, pages 188–195, New York, NY, USA, 2011. ACM.

[44] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. Rethinking access control and authentication for the home internet of things (iot). In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, pages 255–272, Berkeley, CA, USA, 2018. USENIX Association.

[45] A. A. Abd El-Aziz and A. Kannan. A comprehensive presentation to xacml. In *Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)*, pages 155–161, Oct 2013.

[46] OASIS. extensible access control markup language (xacml) version 3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`, Jan 2013. Visited on 1 May 2017.

[47] D. Hardt. The oauth 2.0 authorization framework. RFC 6749, RFC Editor, October 2012.

[48] Kantara Initiative, Inc. User-managed access (uma). `https://kantarainitiative.org/confluence/display/uma/Home`, Apr 2017. Visited on 5 Apr 2017.

[49] J. E. Kim, G. Boulos, J. Yackovich, T. Barth, C. Beckel, and D. Mosse. Seamless integration of heterogeneous devices and access control in smart homes. In *2012 Eighth International Conference on Intelligent Environments*, pages 206–213, Jun 2012.

[50] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, and G. Ferrari. Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios. *IEEE Sens J*, 15(2):1224–1234, Feb 2015.

[51] S. Cirani and M. Picone. Effective authorization for the web of things. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 316–320, Dec 2015.

[52] Göran Selander, Ludwig Seitz, Hannes Tschofenig, Samuel Erdtman, and Erik Wahlstroem. Authentication and Authorization for Constrained Environments (ACE). Internet-Draft draft-ietf-ace-oauth-authz-06, Internet Engineering Task Force, March 2017.

[53] Dragon Cabarkapa. Authorization architecture for swot. Master's thesis, Aalto University, 2013.

[54] D. Rivera, L. Cruz-Piris, G. Lopez-Civera, E. de la Hoz, and I. Marsa-Maestre. Applying an unified access control for iot-based intelligent agent systems. In *2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 247–251, Oct 2015.

[55] Hannes Tschofenig, Eve Maler, Erik Wahlstroem, and Samuel Erdtman. Authentication and Authorization for Constrained Environments Using OAuth and UMA. Internet-Draft draft-maler-ace-oauth-uma-00, Internet Engineering Task Force, March 2015. Work in Progress.

[56] David Ferraiolo and Richard Kuhn. Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[57] A. A. E. Kalam, R. E. 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*, pages 120–131, June 2003.

[58] Fabien. Orbac: Organization based access control. `http://orbac.org/?page_id=21`, 2017. Visited on 20 May 2017.

[59] Vincent C. Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (ABAC) definition and considerations. Technical report, National Institute of Standards and Technology (NIST), jan 2014.

[60] Jaehong Park and Ravi Sandhu. The uconabc usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, February 2004.

[61] B. Anggorojati, N. R. Prasad, and R. Prasad. Secure capability-based access control in the m2m local cloud platform. In *2014 4th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE)*, pages 1–5, May 2014.

[62] Iuon-Chang Lin and Tzu-Chun Liao. A survey of blockchain security issues and challenges. *I. J. Network Security*, 19(5):653–659, 2017.

[63] Jesse Yli-Huumo, Deokyoon Ko, Sujin Choi, Sooyong Park, and Kari Smolander. Where is current research on blockchain technology? a systematic review. *PLoS One*, 11(10):1–27, 10 2016.

[64] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. Technical report, Peercoin, Aug 2012.

[65] Litecoin Project. Litecoin: global decentralized currency. `https://litecoin.org/`, Aug 2017. Visited on 3 Aug 2017.

[66] Andreas Loibl. Namecoin. In *Network Architectures and Services*, Aug 2014.

[67] Jackson Palmer and Shibetoshi Nakamoto. Dogecoin. `http://dogecoin.com/`, Aug 2017. Visited on 3 Aug 2017.

[68] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, and Huaimin Wang. Blockchain challenges and opportunities: A survey, 2016.

[69] Y. Yuan and F. Y. Wang. Towards blockchain-based intelligent transportation systems. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2663–2668, Nov 2016.

[70] Golem project. Golem: Worldwide supercomputer. https://golem.network/, Mar 2017. Visited on 24 Mar 2017.

[71] Augur Project. Welcome to the future of forecasting. https://augur.net/, Mar 2017. Visited on 24 Mar 2017.

[72] S. Huh, S. Cho, and S. Kim. Managing iot devices using blockchain platform. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pages 464–467, Feb 2017.

[73] K. Biswas and V. Muthukkumarasamy. Securing smart cities using blockchain technology. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1392–1393, Dec 2016.

[74] P. Kianmajd, J. Rowe, and K. Levitt. Privacy-preserving coordination for smart communities. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1045–1046, Apr 2016.

[75] Talita Moreira. Bancos vão compartilhar informação via blockchain. https://www.valor.com.br/financas/6304285/bancos-vao-compartilhar-informacao-blockchain, Jun 2019. Visited on 14 Jun 2019.

[76] Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, November 2016.

[77] Information Technology Laboratory. Digital signature standard (DSS). Technical report, National Institute of Standards and Technology, jul 2013.

[78] Donald E. Eastlake 3rd and Tony Hansen. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234, May 2011.

[79] Khanacademy. Bitcoin: what is it? https://www.khanacademy.org/economics-finance-domain/core-finance/money-and-banking/bitcoin/v/bitcoin-what-is-it, sep 2017. Visited on 3 Sep 2017.

[80] F. Tschorsch and B. Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys Tutorials*, 18(3):2084–2123, 2016.

[81] Cointelegraph. Guide on ethereum wallets: Mobile, web, desktop, hardware. https://cointelegraph.com/ethereum-for-beginners/ethereum-wallets, Fev 2020. Visited on 17 Fev 2020.

[82] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[83] Etherscan: The Ethereum Block Explorer. Ethereum unique address growth chart. https://etherscan.io/chart/address, Jul 2018. Visited on 26 Jul 2018.

[84] Josiah Wilmoth. More than 1,000 ethereum dapps have launched since 2017. `https://www.ccn.com/more-than-1000-ethereum-dapps-have-launched-since-2017/`, May 2018. Visited on 25 Jun 2018.

[85] K. Christidis and M. Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016.

[86] O. Alphand, M. Amoretti, T. Claeys, S. Dall'Asta, A. Duda, G. Ferrari, F. Rousseau, B. Tourancheau, L. Veltri, and F. Zanichelli. Iotchain: A blockchain security architecture for the internet of things. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, April 2018.

[87] 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*, 6(2):1594–1605, April 2019.

[88] Oscar Novo. Blockchain meets iot: An architecture for scalable access management in iot. *IEEE Internet of Things Journal*, 5(2):1184–1195, April 2018.

[89] Sehrish Shafeeq, Masoom Alam, and Abid Khan. Privacy aware decentralized access control system. *Future Generation Computer Systems*, 101:420 – 433, 2019.

[90] IOTA Foundation. What is iota? a permissionless distributed ledger for a new economy. `https://www.iota.org/get-started/what-is-iota`, Nov 2019. Visited on 10 Nov 2019.

[91] Serguei Popov, Olivia Saa, and Paulo Finardi. Equilibria in the tangle. *Computers & Industrial Engineering*, 136:160–172, Oct 2019.

[92] The Linux Foundation. Hyperledger: Sawtooth. `https://www.hyperledger.org/projects/sawtooth`, Nov 2019. Visited on 10 Nov 2019.

[93] Stefano Bistarelli, Claudio Pannacci, and Francesco Santini. Capbac in hyperledger sawtooth. In José Pereira and Laura Ricci, editors, *Distributed Applications and Interoperable Systems*, pages 152–169, Cham, 2019. Springer International Publishing.

[94] Kappaneo. Sawtooth-capbac. `https://github.com/kappanneo/sawtooth-capbac`, Nov 2019. Visited on 10 Nov 2019.

[95] Nallapaneni Manoj Kumar and Pradeep Kumar Mallick. Blockchain technology for security issues and challenges in iot. *Procedia Computer Science*, 132:1815 – 1823, 2018. International Conference on Computational Intelligence and Data Science.

[96] Martin Nuss, Alexander Puchta, and Michael Kunz. Towards blockchain-based identity and access management for internet of things in enterprises. In Steven Furnell, Haralambos Mouratidis, and Günther Pernul, editors, *Trust, Privacy and Security in Digital Business*, pages 167–181, Cham, 2018. Springer International Publishing.

[97] Alfonso Panarello, Nachiket Tapas, Giovanni Merlino, Francesco Longo, and Antonio Puliafito. Blockchain and iot integration: A systematic survey. *Sensors*, 18(8), 2018.

[98] Chethana Dukkipati, Yunpeng Zhang, and Liang Chieh Cheng. Decentralized, blockchain based access control framework for the heterogeneous internet of things. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, ABAC'18, pages 61–69, New York, NY, USA, 2018. ACM.

[99] Y. Zhu, Y. Qin, Z. Zhou, X. Song, G. Liu, and W. C. Chu. Digital asset management with distributed permission over blockchain and attribute-based access control. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 193–200, July 2018.

[100] Ronghua Xu, Yu Chen, Erik Blasch, and Genshe Chen. A federated capability-based access control mechanism for internet of things (iots). *CoRR*, abs/1805.00825, 2018.

[101] Ronghua Xu, Yu Chen, Erik Blasch, and Genshe Chen. Blendcac: A blockchain-enabled decentralized capability-based access control for iots. *CoRR*, abs/1804.09267, 2018.

[102] Fabiola Hazel Pohrmen, Rohit Kumar Das, Wanbanker Khongbuh, and Goutam Saha. Blockchain-based security aspects in internet of things network. In Ashish Kumar Luhach, Dharm Singh, Pao-Ann Hsiung, Kamarul Bin Ghazali Hawari, Pawan Lingras, and Pradeep Kumar Singh, editors, *Advanced Informatics for Computing Research*, pages 346–357, Singapore, 2019. Springer Singapore.

[103] N. Klaokliang, P. Teawtim, P. Aimtongkham, C. So-In, and A. Niruntasukrat. A novel iot authorization architecture on hyperledger fabric with optimal consensus using genetic algorithm. In *2018 Seventh ICT International Student Project Conference (ICT-ISPC)*, pages 1–5, July 2018.

[104] Linux Foundation. Hyperledger fabric. `https://www.hyperledger.org/projects/fabric`, Jan 2019. Visited on 12 Jan 2019.

[105] Longfei Wu, Xiaojiang Du, Wei Wang, and Bin Lin. An out-of-band authentication scheme for internet of things using blockchain technology. In *International Conference on Computing, Networking and Communications (ICNC)*, 03 2018.

[106] Mohamed Tahar Hammi, Badis Hammi, Patrick Bellot, and Ahmed Serhrouchni. Bubbles of trust: A decentralized blockchain-based authentication system for iot. *Computers & Security*, 78:126 – 142, 2018.

[107] Chao Lin, Debiao He, Xinyi Huang, Kim-Kwang Raymond Choo, and Athanasios V. Vasilakos. Bsein: A blockchain-based secure mutual authentication with fine-grained access control system for industry 4.0. *Journal of Network and Computer Applications*, 116:42 – 52, 2018.

[108] C. Molina-Jimenez, E. Solaiman, I. Sfyrakis, I. Ng, and J. Crowcroft. On and Off-Blockchain Enforcement Of Smart Contracts. *ArXiv e-prints*, May 2018.

[109] Michael J. Casey. 'layer 2' blockchain tech is an even bigger deal than you think. `https://www.coindesk.com/layer-2-blockchain-tech-even-bigger-deal-think`, May 2018. Visited on 25 Jun 2018.

[110] S. Pal, T. Rabehaja, M. Hitchens, V. Varadharajan, and A. Hill. On the design of a flexible delegation model for the internet of things using blockchain. *IEEE Transactions on Industrial Informatics*, pages 1–1, 2019.

[111] Sara Rouhani and Ralph Deters. Blockchain based access control systems: State of the art and challenges. In *IEEE/WIC/ACM International Conference on Web Intelligence*, WI '19, pages 423–428, New York, NY, USA, 2019. ACM.

[112] X. Liang, S. Shetty, and D. Tosh. Exploring the attack surfaces in blockchain enabled smart cities. In *2018 IEEE International Smart Cities Conference (ISC2)*, pages 1–8, Sep. 2018.

[113] Fabiola Hazel Pohrmen, Rohit Kumar Das, and Goutam Saha. Blockchain-based security aspects in heterogeneous internet-of-things networks: A survey. *Transactions on Emerging Telecommunications Technologies*, 30(10):e3741, 2019. e3741 ett.3741.

[114] Yu Feng, Emina Torlak, and Rastislav Bodík. Precise attack synthesis for smart contracts. *CoRR*, abs/1902.06067, 2019.

[115] O. B. Mora, R. Rivera, V. M. Larios, J. R. Beltrán-Ramírez, R. Maciel, and A. Ochoa. A use case in cybersecurity based in blockchain to deal with the security and privacy of citizens and smart cities cyberinfrastructures. In *2018 IEEE International Smart Cities Conference (ISC2)*, pages 1–4, Sep. 2018.

[116] Win Raymaekers. Cryptocurrency bitcoin: Disruption, challenges and opportunities. *Journal of Payments Strategy & Systems*, 9(1):30–46, 2015.

[117] Jordi Herrera-Joancomartí. *Research and Challenges on Bitcoin Anonymity*, pages 3–16. Springer International Publishing, 2015.

[118] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts sok. In *Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204*, pages 164–186, New York, NY, USA, 2017. Springer-Verlag New York, Inc.

[119] Iuon-Chang Lin and Tzu-Chun Liao. A survey of blockchain security issues and challenges. *International Journal of Network Security*, 19(5):653–659, Sep 2017.

[120] Ghassan O. Karame. Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. In *In Proc. of Conference on Computer and Communication Security*, 2012.

[121] Meni Rosenfeld. Analysis of hashrate-based double spending. *CoRR*, abs/1402.2009, 2014.

[122] Bitcoin. Double-spending. `https://en.bitcoin.it/wiki/Double-spending`, 2017. Visited on 2 May 2017.

[123] Joel Hruska. One bitcoin group now controls 51entire currency's safety. `https://www.extremetech.com/extreme/184427-one-bitcoin-group-now-controls-51-of-total-mining-power-threatening-entire-currencys-safety`, Jun 2014. Visited on 31 Jul 2017.

[124] Alyssa Hertig. Ethereum's big switch: The new roadmap to proof-of-stake. `https://www.coindesk.com/ethereums-big-switch-the-new-roadmap-to-proof-of-stake/`, May 2017. Visited on 25 Jul 2017.

[125] Jamie Redman. Bitcoin community members and developers propose changing bitcoin's proof-of-work. `https://news.bitcoin.com/bitcoin-developers-changing-proof-work-algorithm/`, Mar 2017. Visited on 25 Jul 2017.

[126] Helen Partz. Ethereum 2.0's phase zero scheduled to launch on january 3, 2020: Devs. https://cointelegraph.com/news/ethereum-20s-phase-zero-scheduled-to-launch-on-january-3-2020-devs, Jun 2019. Visited on 8 Jul 2019.

[127] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 692–705, New York, NY, USA, 2015. ACM.

[128] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 129–144, Berkeley, CA, USA, 2015. USENIX Association.

[129] Zhenxuan Bai, Yuwei Zheng, Kunzhe Chai, and Senhua Wang. You may have paid more than you imagine. https://media.defcon.org/DEF%20CON%2026/DEF%20CON%2026%20presentations/DEFCON-26-Bai-Zheng-Chai-Wang-You-May-Have-Paid-more-than-You-Imagine.pdf, Aug 2018. Visited on 13 Jun 2019.

[130] Vitalik Buterin. Eip 155: Simple replay attack protection. https://eips.ethereum.org/EIPS/eip-155, Oct 2016. Visited on 13 Jun 2019.

[131] Alyssa Hertig. Explainer: What is segwit2x and what does it mean for bitcoin? http://www.coindesk.com/explainer-what-is-segwit2x-and-what-does-it-mean-for-bitcoin/, Jul 2017. Visited on 23 Jul 2017.

[132] Bisola Asolo. Transaction malleability explained. https://www.mycryptopedia.com/transaction-malleability-explained/, Oct 2018. Visited on 9 Jul 2018.

[133] Edan Yago. Bitcoin's bogeyman cometh: Why segwit2x is a 51 https://www.coindesk.com/bitcoins-bogeyman-cometh-segwit2x-51-attack, Nov 2017. Visited on 9 Jul 2019.

[134] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, New York, NY, USA, 2017. ACM.

[135] IoT Chain. Iot chain: A high-security lite iot os. https://iotchain.io/whitepaper/ITCWHITEPAPER.pdf, Set 2018. Visited on 9 Sep 2018.

[136] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: Characterizing payments among men with no names. *Commun ACM*, 59(4):86–93, March 2016.

[137] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *Technical Report (draft)*, 2015.

[138] Ghassan Karame. On the security and scalability of bitcoin's blockchain. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1861–1862, New York, NY, USA, 2016. ACM.

[139] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. *Technical Report*, 2014.

[140] Lightning Network. Lightning network: Scalable, instant bitcoin/blockchain transactions. `https://lightning.network/`, Jul 2017. Visited on 30 Jul 2017.

[141] Mark Mueller-Eberstein. The next radical internet transformation: How blockchain technology is transforming business, governments, computing, and security models. ACM Learning Webinars, Apr 2017.

[142] Solidity. Solidity: Introduction to smart contracts. `http://solidity.readthedocs.io/en/latest/introduction-to-smart-contracts.html`, Jun 2018. Visited on 13 Jun 2018.

[143] Antonio L. Maia Neto, Artur L. F. Souza, Italo Cunha, Michele Nogueira, Ivan Oliveira Nunes, Leonardo Cotta, Nicolas Gentille, Antonio A. F. Loureiro, Diego F. Aranha, Harsh Kupwade Patil, and Leonardo B. Oliveira. Aot: Authentication and access control for the entire iot device life-cycle. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, SenSys '16, pages 1–15, New York, NY, USA, 2016. ACM.

[144] Hokey Min. Blockchain technology for enhancing supply chain resilience. *Business Horizons*, 62(1):35 – 45, 2019.

[145] Edvard Tijan, Saša Aksentijević, Katarina Ivanić, and Mladen Jardas. Blockchain technology implementation in logistics. *Sustainability*, 11(4), 2019.

[146] Google. G suite drive. `https://gsuite.google.com/products/drive/`, Jun 2019. Visited on 08 Jun 2019.

[147] Public Knownledge Project. Open journal systems. `https://pkp.sfu.ca/ojs/`, Jun 2019. Visited on 8 Jun 2019.

[148] Butian Huang, Zhenguang Liu, Jianhai Chen, Anan Liu, Qi Liu, and Qinming He. Behavior pattern clustering in blockchain networks. *Multimed Tools Appl*, 76(19):20099–20110, Oct 2017.

[149] Adam Shostack. *Threat Modeling: Designing for Security*. Wiley Publishing, 1st edition, 2014.

[150] Matt. Bitcoin's attack vectors: Sybil & eclipse attacks. `https://medium.com/chainrift-research/bitcoins-attack-vectors-sybil-eclipse-attacks-d1b6679963e5`, Nov 2018. Visited on 17 Aug 2019.

[151] CodeTract. Inside an ethereum transaction. `https://medium.com/@codetractio/inside-an-ethereum-transaction-fa94ffca912f`, Feb 2017. Visited on 04 Sep 2019.

[152] ethereum. go-ethereum/p2p/server.go. `https://github.com/ethereum/go-ethereum/blob/master/p2p/server.go`, Jul 2019. Visited on 27 Aug 2019.

[153] Ethereum Classic. Rlpx encryption. https://github.com/ethereumproject/go-ethereum/wiki/RLPx-Encryption, Mar 2018. Visited on 27 Aug 2019.

[154] Dominiek Ter Heide. Ethereum is becoming increasingly vulnerable to attack. https://www.longhash.com/news/ethereum-is-becoming-increasingly-vulnerable-to-attack, Jul 2019. Visited on 27 Aug 2019.

[155] Mike Orcutt. Once hailed as unhackable, blockchains are now getting hacked. https://www.technologyreview.com/s/612974/once-hailed-as-unhackable-blockchains-are-now-getting-hacked/, Aug 2019. Visited on 13 Aug 2019.

[156] Dominiek Ter Heide. A closer look at ethereum signatures. https://hackernoon.com/a-closer-look-at-ethereum-signatures-5784c14abecc, Feb 2018. Visited on 27 Aug 2019.

[157] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. A survey on the security of blockchain systems. *Future Generation Computer Systems*, 2017.

[158] Binance Academy. What is a dusting attack? https://www.binance.vision/security/what-is-a-dusting-attack, 2019. Visited on 17 Aug 2019.

[159] Christine Kim. Code for ethereum's proof-of-stake blockchain to be finalized next month. https://www.coindesk.com/code-for-ethereums-proof-of-stake-blockchain-to-be-finalized-next-month, May 2019. Visited on 14 Aug 2019.

[160] Atomic Wallet. What is a 51https://atomicwallet.io/51-attack, Sep 2019. Visited on 28 Out 2019.

[161] Etherscan. Full node sync with default settings. https://etherscan.io/chartsync/chaindefault, Oct 2019. Visited on 28 Oct 2019.

[162] Bernhard Mueller. Ethereum contract security techniques and tips. https://github.com/ethereum/wiki/wiki/Safety, Oct 2018. Visited on 28 Aug 2019.

[163] Cyber Security. All you know about stride - elevation of privilege threat (eop). https://securitycommunity.tcs.com/infosecsoapbox/articles/2017/06/07/all-you-know-about-stride-elevation-privilege-threat-eop, Jun 2017. Visited on 28 Aug 2019.

[164] Gustavo Guimaraes. Reentrancy attack on smart contracts: How to identify the exploitable and an example of an attack contract. https://medium.com/@gus_tavo_guim/reentrancy-attack-on-smart-contracts-how-to-identify-the-exploitable-and-an-example-of-an-attack-4470a2d8dfe4, May 2017. Visited on 22 Aug 2019.

[165] Alber Erre. Understanding uint overflows and underflows — solidity (ethereum). https://medium.com/3-min-blockchain/understanding-uint-overflows-and-underflows-solidity-ethereum-8603339259e6, Apr 2018. Visited on 21 Aug 2019.

[166] ConsenSys Diligence. Known attacks. https://consensys.github.io/smart-contract-best-practices/known_attacks, Aug 2019. Visited on 22 Aug 2019.

[167] Lorenz Breidenbach, Phil Daian, Ari Juels, and Emin Gün Sirer. An in-depth look at the parity multisig bug. http://hackingdistributed.com/2017/07/22/deep-dive-parity-bug/, 2017. Visited on 21 Aug 2019.

[168] Hudson Jameson. Faq: Upcoming ethereum hard fork. https://blog.ethereum.org/2016/10/18/faq-upcoming-ethereum-hard-fork/, Oct 2016. Visited on 22 Aug 2019.

[169] Vitalik Buterin. Gas cost changes for io-heavy operations. https://eips.ethereum.org/EIPS/eip-150, Sep 2016. Visited on 22 Aug 2019.

[170] ZhouW. Are solidity contracts still vulnerable to callstack exploits? https://ethereum.stackexchange.com/questions/26105/are-solidity-contracts-still-vulnerable-to-callstack-exploits/26106, Sep 2017. Visited on 22 Aug 2019.

[171] Chris Coverdale. Solidity: Transaction-ordering attacks. https://medium.com/coinmonks/solidity-transaction-ordering-attacks-1193a014884e, Mar 2018. Visited on 22 Aug 2019.

[172] ConsenSys Diligence. Known attacks. https://consensys.github.io/smart-contract-best-practices/recommendations/, Aug 2019. Visited on 22 Aug 2019.

[173] Perkins Coie. Blockchain attacks and the fight for immutability. https://www.jdsupra.com/legalnews/blockchain-attacks-and-the-fight-for-87600/, Feb 2019. Visited on 16 Aug 2019.

[174] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles A. Kamhoua, Sachin Shetty, DaeHun Nyang, and Aziz Mohaisen. Exploring the attack surface of blockchain: A systematic overview. *CoRR*, abs/1904.03487, 2019.

[175] SmartContractSecurity. Swc registry: Smart contract weakness classification and test cases. https://swcregistry.io/, Nov 2019. Visited on 17 Nov 2019.

[176] AnChain.AI Inc. Secure and grow your blockchain business. https://www.anchain.ai/, Aug 2019. Visited on 14 Aug 2019.

[177] ChainSecurity. Securify: security scanner for ethereum smart contracts. https://securify.chainsecurity.com/, Nov 2019. Visited on 08 Nov 2019.

[178] Petar Tsankov, Andrei Dan, Dana Drachsler Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. Securify: Practical security analysis of smart contracts, 2018.

[179] GuardStrike. Contractguard white paper. http://guard-strike.oss-cn-shanghai.aliyuncs.com/contract-guard/ContractGuard-WhitePaper-V1.0.pdf, Nov 2019. Visited on 09 Nov 2019.

[180] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov. Smartcheck: Static analysis of ethereum smart contracts. In *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 9–16, May 2018.

[181] crytic. Slither. `https://github.com/crytic/slither`, Nov 2019. Visited on 09 Nov 2019.

[182] melonproject. Oyente: An analysis tool for smart contracts. `https://github.com/melonproject/oyente`, Nov 2019. Visited on 08 Nov 2019.

[183] trailofbits. Manticore. `https://github.com/trailofbits/manticore`, Nov 2019. Visited on 09 Nov 2019.

[184] Bo Jiang, Ye Liu, and W. K. Chan. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, pages 259–269, New York, NY, USA, 2018. ACM.

[185] ConsenSys. Mythril. `https://github.com/ConsenSys/mythril`, Nov 2019. Visited on 08 Nov 2019.

[186] quoscient. Octopus. `https://github.com/quoscient/octopus`, Nov 2019. Visited on 09 Nov 2019.

APPENDIX A – E-CONTROLCHAIN ATTRIBUTE-BASED ACCESS CONTROL PSEUDOCODE

Next, we expose, as a pseudocode, the crucial parts of the E-ControlChain attribute-based authorization. The Listings A.1 and A.2 shows the pseudocode of the basic and common control, and the attribute-based access control of E-ControlChain, respectively. The "BasicAndCommon-Control" define variables and functions that are common to all the three types of rules defined by the ControlChain: attribute, capability and ACL. The "AttributeControl" define the variables and functions that are particularly related to the attribute-based access control. Also, note that the "AttributeControl" inherits the "BasicAndCommonControl" and, thus, all its variables and functions.

Although, in Ethereum, it works more similar to a hash table with infinite entries and where we cannot iterate over the hash keys, when the pseudocode shows a variable type declaration as, for example, "bool[address]", it can be understood as an array of bool (booleans) that requires an index of type "address", and when its declaration is "someType[uint]", it is an array of "someType" that requires an uint (unsigned int) index.

Listing A.1: E-ControlChain - BasicAndCommonControl pseudocode

```
Contract BasicAndCommonControl {
  address[address] owners;
  bool[address] approved_ownerships;
  address[address] rules_authorities;
  int[address][string] context;
  ContextRule[uint] context_rules;
  struct ContextRule {
    address source;
    string identifier;
    Comparators comparator;
    int value;
  }
  function setOwner(address addr, address new_owner) {
    REQUIREMENT: be the current owner or be the entity behind "addr" if
        it has no owner yet
    approved_ownerships[addr] = false;
    rules_authorities[addr] = 0x00;
    owners[addr] = new_owner;
  }
  function approveOwnership(address addr) {
    REQUIREMENT: be the new owner of "addr"
    approved_ownerships[addr] = true;
  }
  function setRulesAuthority(address resource, address addr) {
    REQUIREMENT: be the "resource" itlsef or its current owner
    REQUIREMENT 2: "resource" ownership is with approved status
    rules_authorities[resource] = addr;
  }
  function setContext(address source, string identifier, int value) {
    REQUIREMENT: be the entity behind "addr" or the current owner of it
    context[source][identifier] = value;
  }
  function newContextRule(address source, string identifier, Comparators
      comparator, int value) {
```

```
    ContextRule context_rule = ContextRule(source, identifier, comparator,
        value);
    context_rules.push(context_rule);
  }
  function isContextSatisfied(uint[uint] context_rules_ indexes) {
    foreach (i in context_rules_indexes) {
      if (context_rule[i] does not hold) {
        return false;
      }
    }
    return true;
  }
}
```

Listing A.2: E-ControlChain - AttributeControl pseudocode

```
Contract AttributeControl extends BasicAndCommonControl {
  bool[address][address][string] attributes;
  AttributeRule[address][uint] attribute_rules;
  Struct AttributeRule {
    address resource_authority;
    string resource_attr;
    address requester_authority;
    string requester_attr;
    bool inherit_owner_attr;
    uint[uint] context;
    Actions[uint] actions;
  }
  Function setAttribute(address addr, string attr, bool assigned) {
    attributes[msg.sender][addr][attr] = assigned;
  }
  Function createAttributeRule(address resource_authority, string
      resource_attr, address requester_authority, string requester_attr,
      bool inherit_owner_attr, uint[uint] context, Actions[uint] actions)
      {
    AttributeRule attribute_rule = AttributeRule(resource_authority,
        resource_attr, requester_authority, requester_attr,
        inherit_owner_attr, context, actions);
    attribute_rules[msg.sender].push(attribute_rule);
  }
  Function authorizedByAttributeRules(address resource, address requester
      , Actions action) {
    foreach (ar in attribute_rules[rules_authorities[resource]]) {
      if (the ar.resource_authority defined the ar.resource_attr for the
          resource) {
        if (the ar.requester_authority defined the ar.requester_attr for
            the requester or (the inherit of requester owner attributes by
            the requester is allowed by the ar and the ownership of the
            resource was approved and the ar.requester_authority defined
            the ar.requester_attr for the owner of the requester)) {
          if{isContextSatisfied(ar.context) \textbf{and} ar.actions
              contains action} {
            return true;
          }
        }
      }
    }
    return false;
31    }
32  }
```

The Listing A.1 presents the pseudo algorithm of `BasicAndCommonControl` contract. As mentioned before, it defines the common variables, structures and functions used by authorization contracts. In total, there are three variables defined by it: `owners`, `context` and `context_rules` (lines 2-6). The `owners` and `context` variables are mappings (variables similar to hash tables) and store the owners of addresses and context values generated by the addresses, respectively. This means that each network address, representing a device or not, can belong to another address that will have special control over managed addresses. Furthermore, each address can publish an undefined number of context values under its domain. The context values are indexed by the address of the source and a string identifier.

The `ContextRule` structure (lines 7-12) can keep rules of context. These rules define a context source address, a context string identifier, a value and a comparator. The comparator can be any of the main mathematical comparative mechanisms, *i.e.* »", »=", "==<="or «", and will be used to compare the value defined in the context rule with the current context, defined in the variable `context`. The defined `ContextRule` objects are stored in the `context_rules` array variable and its indexes can be used in the authorization rules (see Listing A.2).

Finally, we also define five functions in `BasicAndCommonControl` contract: `setOwner`, `approveOwnership`, `setContext`, `newContextRule` and `isContextSatisfied` (lines 13-43). The `setOwner` is used to define a new owner to an address. This function can be invoked only by the current owner of the address or by the address itself if it has not an owner defined yet. Note that it always set to false the flag of approved ownership. This flag only can be set to true by the new owner using the `approveOnwership` function. Belonging to an address has no real value if this flag is false. The `setContext` function allows the address itself or its owner to specify values to its context identifiers. The `newContextRule` function is used to define new rules of context that can be further used in the access rules. The last function, `isContextSatisfied`, allows to check if the context rules related to the receive indexes are true for the current context values.

Unfortunately, the current Application Binary Interface does not support objects to be passed trough function arguments, so, as the case of `newContextRule` and other functions, we have to receive all its members separately instead of a single object. This is planned to be surpassed with the version 2 of ABIencoder, however it was still too unstable when we tried to use it.

The Listing A.2 presents the `AttributeControl` contract. This contract describe the main idea behind the attribute-based access control of ControlChain. It defines three mappings variables: `attributes`, `attribute_rules` and `attribute_rules_authorities` (lines 2-3). The `attributes` keeps all attributes given by any address to any other address. For example, the address “0x01” can give the address “0x03” an attribute “friend”. The variable `attribute_rules` keeps the attribute access control rules, defined using objects of type `AttributeRule`. Each address, in this case namely authority, can define its own rules. The last variable, `attribute_rules_authorities`, defines for each address who is its authority, i.e., defines the address that has the access control rules that it has to follow. Thus, “attribute_rules_authorities[0x02]” returning “0x03” means that the access should be granted if one of the rules defined by address “0x03” gives the permission, even if the owner of the device is “0x01”. This allows the delegation of the access control without requiring the owner to give up of its ownership.

The structure `AttributeRule` (lines 4-12) stores the attribute based access control rules. It defines the attributes of resources and requesters (`resource_attr` and `requester_attr`), the respective attribute authorities for them (`resource_authority` and `requester_authority`), a boolean variable to permit that addresses owned by allowed owners also receive the authorization, a context index array that is required to be satisfied before the access is granted and, finally, an array `actions` containing the rule authorized interactions (for example, read, write, execute, etc.). The attribute authorities are the addresses to check for the attributes. For example, the requester attribute be “friend”, its attribute authority be “0x01” means that, if the requester is “0x03”, the return of “attributes[0x01][0x03][friend]” should be true. If not, the access attempt should be denied.

The Listing A.2 also defines four functions: setAttributeRulesAuthority, setAttribute, createAttributeRule and authorizedByAttributeRules (lines 13-31). The functions `setAttributeRulesAuthority` and `SetAttribute` define the attribute authority for a resource address and give an attribute for an address, respectively. Note that the attribute given is bounded to the attribute authority represented by the message sender (`msg.sender`). The function `createAttributeRule` creates an object `AttributeRule`, that defines an access rule, and store this rule under the domain of the message sender. Finally, the function `authorizedByAttributeRules` defines if the requester can access the resource. In order to do this, it recovers the AttributeRule objects defined by attribute rule authority of the resource and for each one of them it: (1) verifies if the resource has its attribute defined by resource authority (line 22); (2) verifies if the requester or, if the `inherit_owner_attr` is true, its owner has the requester attribute defined by requester authority (line 23); (3) if the context and attempting action, defined in the `AttributeRule`, is satisfied and allowed, respectively, allow the access (line 25); (4) if one of the three steps fail to validate, the next rule is evaluated, starting from the step 1; (5) if all rules were evaluated and no one had succeed on the first three steps, the access is denied (line 30).

For security reasons, some of the functions presented in the listings can only be invoked by the owner of the address or the address itself. Also, although we did not specified it in the Listings A.1 and A.2, some function (that does not require alteration of the Blockchain state and, therefore, are characterized by only reading) can and should be computed locally on the Ethereum node, i.e. outside of the mining process. For example, the function `authorizedByAttributeRules`. In solidity, this can be granted adding the “view” keyword in the function declaration. In this way, even this functions being executed over the contract, they do not generate extra cost and its response time can be as rapidly as common databases.

APPENDIX B – E-CONTROLCHAIN CODE

Listing B.1: E-ControlChain code

```
pragma solidity 0.5.11;

contract BasicAndCommonControl {
  mapping (address => address) public owners;
  mapping (address => bool) public approvedOwnerships;
  mapping (address => address) public rulesAuthorities ;
  mapping (address => mapping (string => int)) public context;
  ContextRule[] public contextRules;

  enum Actions {Read, Write, Execute}
  enum Comparators {GreaterThan, GreaterThanOrEqualTo, Equal, NotEqual, LessThanOrEqualTo,
      ↪ LessThan}

  struct ContextRule {
    address source;
    string identifier ;
    Comparators comparator;
    int value;
  }

  event SetOwnerEvent(address indexed addr, address indexed newOwner);
  event ApprovedOwnershipEvent(address indexed addr, address indexed owner);
  event SetRulesAuthorityEvent(address indexed resource, address indexed authority);
  event SetContextEvent(address indexed source, string indexed identifier , int indexed value);
  event NewContextRuleEvent(uint id, address indexed source, string indexed identifier ,
      ↪ Comparators indexed comparator, int value);

  modifier isTheAddress(address addr) {
    require (msg.sender == addr);
    _;
  }

  modifier isTheOwner(address addr) {
    require (msg.sender == owners[addr]);
    _;
  }

  modifier isOwnershipApproved(address addr) {
    require (approvedOwnerships[addr]);
    _;
  }

  function setOwner(address addr, address newOwner) external {
    require ((owners[addr] == address(0) && msg.sender == newOwner) || (owners[addr] != address
        ↪ (0) && msg.sender == owners[addr]));
    approvedOwnerships[addr] = false ;
    rulesAuthorities [addr] = address(0);
    owners[addr] = newOwner;
    emit SetOwnerEvent(addr, newOwner);
  }
  function approveOwnership(address addr) isTheOwner(addr) external {
    approvedOwnerships[addr] = true;
    emit ApprovedOwnershipEvent(addr, msg.sender);
  }

  // To delete the delegation, just set its value to 0x000.., its default value.
  function setRulesAuthority(address resource, address addr) isTheOwner(resource)
      ↪ isOwnershipApproved(resource) external {
    rulesAuthorities[resource] = addr;
    emit SetRulesAuthorityEvent(resource, addr);
  }

  // To delete the context, just set its value to 0 (zero), its default value.
  function setContext(address source, string calldata identifier, int value) isTheAddress(source
      ↪ ) external {
    context[source][identifier] = value;
    emit SetContextEvent(source, identifier, value);
  }

  // The comparator is a uint defined in the enum ContextComparation.
  function newContextRule(address source, string calldata identifier, Comparators comparator,
      ↪ int value) external {
    ContextRule memory contextRule = ContextRule(source, identifier, comparator, value);
    uint index = contextRules.push(contextRule) - 1;
    emit NewContextRuleEvent(index, source, identifier, comparator, value);
  }

  function isContextSatisfied(uint[] memory contextRulesIndexes) internal view returns(bool) {
    for (uint i = 0; i < contextRulesIndexes.length; i++) {
      ContextRule memory contextRule = contextRules[contextRulesIndexes[i]];
      if (
      (contextRule.comparator == Comparators.GreaterThan
      && context[contextRule.source][contextRule.identifier] <= contextRule.value) ||
      (contextRule.comparator == Comparators.GreaterThanOrEqualTo
      && context[contextRule.source][contextRule.identifier] < contextRule.value) ||
      (contextRule.comparator == Comparators.Equal
      && context[contextRule.source][contextRule.identifier] != contextRule.value) ||
      (contextRule.comparator == Comparators.NotEqual
      && context[contextRule.source][contextRule.identifier] == contextRule.value) ||
      (contextRule.comparator == Comparators.LessThanOrEqualTo
      && context[contextRule.source][contextRule.identifier] > contextRule.value) ||
      (contextRule.comparator == Comparators.LessThan
      && context[contextRule.source][contextRule.identifier] >= contextRule.value)
      ) {
        return false;
      }
    }
    return true;
  }
}

contract AttributeControl is BasicAndCommonControl {
  mapping(address => mapping(address => mapping(string => bool))) public attributes;
  mapping(address => AttributeRule[]) public attributeRules;

  struct AttributeRule {
    address resourceAuthority;
    string resourceAttr;
    address requesterAuthority ;
    string requesterAttr ;
    bool inheritOwnerAttr ;
    uint [] context ;
    Actions [] actions ;
}

event SetAttributeEvent (address addr, string attr , bool assigned );
event CreateAttributeRuleEvent (address indexed resourceAuthority , string resourceAttr , address
    ↪ indexed requesterAuthority , string requesterAttr , bool inheritOwnerAttr , uint []
    ↪ context , Actions [] actions );
event DeleteAttributeRuleEvent (address indexed resourceAuthority , string resourceAttr , address
    ↪ indexed requesterAuthority , string requesterAttr , bool inheritAwnerAttr , uint []
    ↪ context , Actions [] actions );
event DeleteAllAttributeRulesEvent (address addr);

// To delete the attribute , just set its value to false , its default value.
function setAttribute (address addr, string calldata attr , bool assigned ) external {
    attributes [msg.sender][addr][ attr ] = assigned;
    emit SetAttributeEvent (addr, attr , assigned );
}

function createAttributeRule (address resourceAuthority , string calldata resourceAttr , address
    ↪ requesterAuthority , string calldata requesterAttr , bool inheritOwnerAttr , uint []
    ↪ calldata ruleContext , Actions [] calldata actions ) external {
    AttributeRule memory attributeRule = AttributeRule ( resourceAuthority , resourceAttr ,
        ↪ requesterAuthority , requesterAttr , inheritOwnerAttr , ruleContext , actions );
    attributeRules [msg.sender ]. push( attributeRule );
    emit CreateAttributeRuleEvent ( resourceAuthority , resourceAttr , requesterAuthority ,
        ↪ requesterAttr , inheritOwnerAttr , ruleContext , actions );
}

function deleteAttributeRule (address resourceAuthority , string memory resourceAttr, address
    ↪ requesterAuthority , string memory requesterAttr, bool inheritOwnerAttr , uint [] memory
    ↪ ruleContext, Actions [] memory actions) public {
    AttributeRule [] storage resourceAttributeRules = attributeRules [msg.sender ];

    uint resourceAttributeRulesLength = resourceAttributeRules . length ;
    for (uint i = 0; i < resourceAttributeRulesLength ; i++) {
        if ( resourceAttributeRules [ i ]. resourceAuthority == resourceAuthority && keccak256(abi.
            ↪ encodePacked( resourceAttributeRules [ i ]. resourceAttr )) == keccak256(abi.
            ↪ encodePacked( resourceAttr )) && resourceAttributeRules [ i ]. requesterAuthority ==
            ↪ requesterAuthority && keccak256(abi.encodePacked( resourceAttributeRules [ i ].
            ↪ resourceAttr )) == keccak256(abi . encodePacked( requesterAttr )) &&
            ↪ resourceAttributeRules [ i ]. inheritOwnerAttr == inheritOwnerAttr ) {
            if (ruleContext . length == resourceAttributeRules [ i ]. context . length && actions. length ==
                ↪ resourceAttributeRules [ i ]. actions . length ) {
                bool deleteFlag = true;
                uint j;
                for (j = 0; j < ruleContext . length ; j++) {
                    if (ruleContext [ j ] != resourceAttributeRules [ i ]. context [ j ]) {
                        deleteFlag = false ;
                    }
                }
                for (j = 0; j < actions . length ; j++) {
                    if ( actions [ j ] != resourceAttributeRules [ i ]. actions [ j ]) {
                        deleteFlag = false ;
                    }
```

```
          }
          if ( deleteFlag ) {
            resourceAttributeRules [ i ] = resourceAttributeRules [ resourceAttributeRules . length −1];
            resourceAttributeRules .pop();
            emit DeleteAttributeRuleEvent ( resourceAuthority , resourceAttr , requesterAuthority ,
                ↪ requesterAttr , inheritOwnerAttr , ruleContext , actions );
            break;
          }
        }
      }
    }
  }

  function deleteAllAttributeRules () external {
    delete ( attributeRules [msg.sender]);
    emit DeleteAllAttributeRulesEvent (msg.sender);
  }

  function authorizedByAttributeRules (address resource , address requester , Actions action )
      ↪ external view returns (bool) {
    AttributeRule [] storage resourceAttributeRules = attributeRules [ rulesAuthorities [resource ]];

    uint resourceAttributeRulesLength = resourceAttributeRules . length ;
    for (uint i = 0; i < resourceAttributeRulesLength ; i++) {
      AttributeRule memory attributeRule = resourceAttributeRules [ i ];
      if ( attributes [ attributeRule . resourceAuthority ][ resource ][ attributeRule . resourceAttr ]) {
        if ( attributes [ attributeRule . requesterAuthority ][ requester ][ attributeRule . requesterAttr ]
            ↪ || ( attributeRule . inheritOwnerAttr && approvedOwnerships[requester] &&
            ↪ attributes [ attributeRule . requesterAuthority ][ owners[ requester ]][ attributeRule .
            ↪ requesterAttr ])) {
          uint [] memory ruleContext = attributeRule . context ;
          for (uint j = 0; j < attributeRule . actions . length ; j++) {
            if (action == attributeRule . actions [ j ] && isContextSatisfied (ruleContext )) {
              return true ;
            }
          }
        }
      }
    }
    return false ;
  }
}

contract CapabilityControl is BasicAndCommonControl {
  mapping (address => mapping (address => mapping (address => uint ))) public
      ↪ capabilitiesShortcuts ;
  mapping (address => mapping (address => CapabilityRule [])) public capabilitiesRules ;

  struct CapabilityRule {
    address resource ;
    uint [] context ;
    Actions [] actions ;
  }

  event AuthorizeCapabilityAccessEvent (address indexed rulesAuthorities , address indexed
      ↪ resource , address indexed requester , uint [] context , Actions [] actions , uint index);
  event DeauthorizeCapabilityAccessEvent (address indexed rulesAuthorities , address indexed
      ↪ resource , address indexed requester );
  event CapabilityIndexChangeEvent(uint indexed changedCapabilityShortcutIndexFrom, uint indexed changedCapabilityShortcutIndexTo);

  function authorizeCapabilityAccess(address resource, address requester, uint[] calldata context, Actions[] calldata actions)
  external
  {
    CapabilityRule memory capabilityRule = CapabilityRule(resource, context, actions);
    uint index = capabilitiesRules[msg.sender][requester].push(capabilityRule) - 1;
    capabilitiesShortcuts[msg.sender][requester][resource] = index;
    emit AuthorizeCapabilityAccessEvent(msg.sender, resource, requester, context, actions, index);
  }

  function deauthorizeCapabilityAccess(address resource, address requester)
  external
  {
    uint index = capabilitiesShortcuts[msg.sender][requester][resource];

    CapabilityRule[] storage requesterCapabilities = capabilitiesRules[msg.sender][requester];
    if (requesterCapabilities[index].resource == resource) {
      requesterCapabilities[index] = requesterCapabilities[requesterCapabilities.length-1];
      capabilitiesShortcuts[msg.sender][requester][requesterCapabilities[index].resource] = index;
      delete(capabilitiesShortcuts[msg.sender][requester][resource]);
      requesterCapabilities.pop();

      emit DeauthorizeCapabilityAccessEvent(msg.sender, resource, requester);
      emit CapabilityIndexChangeEvent(requesterCapabilities.length, index);
    }
  }

  function authorizedByCapabilityRules(address resource, address requester, Actions action)
  external
  view
  returns (bool)
  {
    uint index = capabilitiesShortcuts[rulesAuthorities[resource]][requester][resource];
    if (capabilitiesRules[rulesAuthorities[resource]][requester][index].resource == resource) {
      uint[] memory ruleContext = capabilitiesRules[rulesAuthorities[resource]][requester][index].context;
      Actions[] memory actions = capabilitiesRules[rulesAuthorities[resource]][requester][index].actions;
      for (uint i = 0; i < actions.length; i++) {
        if (actions[i] == action) {
          return isContextSatisfied(ruleContext);
        }
      }
    }
    return false;
  }
}

contract AclControl is BasicAndCommonControl {
  mapping(address => mapping(address => mapping(address => uint))) public aclsShortcuts;
  mapping(address => mapping(address => AclRule[])) public aclsRules;

  struct AclRule {
    address  requester ;
    uint []  context ;
    Actions []  actions ;
  }

  event AuthorizeAclAccessEvent(address indexed rulesAuthorities , address indexed resource , address indexed requester , uint [] context , Actions [] actions , uint index ) ;
  event DeauthorizeAclAccessEvent(address indexed rulesAuthorities , address indexed resource , address indexed requester ) ;
  event AclIndexChangeEvent(uint indexed changedAclShortcutIndexFrom, uint indexed changedAclShortcutIndexTo) ;

  function authorizeAclAccess( address resource , address requester , uint [] calldata context , Actions [] calldata actions )
  external
  {
    AclRule memory aclRule = AclRule( requester , context , actions ) ;
    uint index = aclsRules [msg.sender ][ resource ]. push(aclRule ) - 1;
    aclsShortcuts [msg.sender ][ resource ][ requester ] = index ;
    emit AuthorizeAclAccessEvent(msg.sender, resource , requester , context , actions , index ) ;
  }

  function deauthorizeAclAccess( address resource , address requester )
  external
  {
    uint index = aclsShortcuts [msg.sender ][ resource ][ requester ];

    AclRule[] storage resourceAcls = aclsRules [msg.sender ][ resource ];
    if ( resourceAcls [ index ]. requester == requester ) {
      resourceAcls [index ] = resourceAcls [ resourceAcls . length -1];
      aclsShortcuts [msg.sender ][ resource ][ resourceAcls [index ]. requester ] = index ;
      delete ( aclsShortcuts [msg.sender ][ requester ][ resource ]) ;
      resourceAcls .pop() ;

      emit DeauthorizeAclAccessEvent(msg.sender, resource , requester ) ;
      emit AclIndexChangeEvent(resourceAcls. length , index ) ;
    }
  }

  function authorizedByAclRules( address resource , address requester , Actions action )
  external
  view
  returns (bool)
  {
    uint index = aclsShortcuts [ rulesAuthorities [resource ]][ resource ][ requester ];
    if ( aclsRules [ rulesAuthorities [resource ]][ resource ][ index ]. requester == requester ) {
      uint [] memory ruleContext = aclsRules [ rulesAuthorities [resource ]][ resource ][ index ]. context ;
      Actions [] memory actions = aclsRules [ rulesAuthorities [resource ]][ resource ][ index ]. actions ;
      for ( uint i = 0; i < actions . length ; i++) {
        if ( actions [ i ] == action ) {
          return isContextSatisfied ( ruleContext ) ;
        }
      }
    }
    return false ;
  }
}
```

```
contract  ControlChain  is   AttributeControl , CapabilityControl ,AclControl  {
   constructor ()  public  {
   }
}
```

APPENDIX C – AUDITING TOOLS REPORTS

C.1 ANCHAIN.AI

Search for the vulnerabilities:

- integer underflow;
- integer overflow;
- parity multisig bug;
- call stack depth attack;
- transaction ordering dependency;
- timestamp dependency;
- re-entracy.

Unfortunately, the AnChain.AI isn't working anymore to test E-ControlChain. After we tested it, we updated it to Solidity v0.5. So we believe that perhaps AnChain.AI cannot deal correctly with the new version although it is listed as compatible. Being so, we do not have the full report and only has parts of the report from E-ControlChain in Solidity v0.4.23. Next, we present the error that is being produced instead of the report. We verified and the contract is compiling and even setting the compiler version manually, it did not work (two of three steps from the troubleshooting list). We did not contact the service desk as this is a feature that requires payment.

Listing C.1: AnChain.AI report

```
ETH Smart Contract Audit Report

Overview

No information is available in this section.

Please try these steps below:
  - This problem may be caused by the unmatched solc compiler version. Please manually select a
      ↪ compiler version and retry.
  - This problem may be caused by a bad syntax within the contract. Please make sure the
      ↪ contract source code can be compiled.
  - Contact service desk and provide the info below:
    - Contract source code or address
    - Error messages you see
    - A screenshot of the report section

Recommendations

No information is available in this section
```

C.2 SECURIFY

Search for the vulnerabilities:

- no Ether liquidity;
- writes after call;
- no restricted write;
- no restricted transfer;
- no handled exception;
- transaction ordering dependency;
- no validated arguments.

Listing C.2: Securify report

```
Total issues: 17

Insecure Coding Patterns: 14
  Unrestricted write to storage: contracts fields that can be modified by any user must be
    ↪ inspected
    BasicAndCommonControl: 3
    BasicAndCommonControl: 43
    BasicAndCommonControl: 44
    BasicAndCommonControl: 45
    BasicAndCommonControl: 69
    BasicAndCommonControl: 305
     AttributeControl : 97
     AttributeControl : 118
     AttributeControl : 124
     AttributeControl : 148
     AttributeControl : 149
     AttributeControl : 305
    ControlChain: 302
    ControlChain: 305

Unexpected Ether Flows: 3
  Locked Ether: contracts that may receive ether must also allow users to extract the deposited
    ↪ ether from the contract
    BasicAndCommonControl: 3
     AttributeControl : 97
    ControlChain: 300
```

C.3 CONTRACTGUARD

Search for the vulnerabilities:

- gasless send;
- exception disorder;
- reentrancy;

- timestamp dependency;
- block number dependency;
- dangerous delegatecall;
- integer overflow;
- integer underflow;
- freezing ether.

Listing C.3: ContractGuard report

```
BasicAndCommonControl
  Lines: 3−95
  Errors: 0

 AttributeControl
  Lines: 97−182
  Errors: 0

ControlChain
  Lines: 302−305
  Errors: 0
```

C.4 SMARTCHECK

Search for the vulnerabilities:

- balance equality;
- unchecked external call;
- DoS by external contract;
- send instead of transfer;
- re-entrancy;
- malicious libraries;
- using tx.origin;
- transfer forwards all gas;
- integer division;
- locked money;
- unchecked math;
- timestamp dependence;
- unsafe type inference;

- byte array;
- costly loop;
- token API violation;
- compiler version not fixed;
- private modifier;
- redundant fallback function;
- style guide violation;
- implicit visibility level.

Listing C.4: SmartCheck report

```
Costly loop
  Lines: 74−92
   Severity : 1

  Lines: 132−155
   Severity : 2

  Lines: 137−141
   Severity : 1

  Lines: 142−146
   Severity : 1

  Lines: 167−179
   Severity : 2

Prefer external to public visibility level
  Lines: 128−156
   Severity : 1


Detail
  Costly loop: Ethereum is a very resource−constrained environment. Prices per computational
      ↪ step are orders of magnitude higher that with centralized providers . Moreover,
      ↪ Ethereum miners impose a limit on the total number of gas consumed in a block. If
      ↪ array . length is large enough, the function exceeds the block gas limit , and
      ↪ transactions calling it will never be confirmed. This becomes a security issue , if an
      ↪ external actor influences array . length . E.g ., if array enumerates all registered
      ↪ addresses , an adversary can register many addresses, causing the problem described
      ↪ above.

   Prefer external to public visibility level : a function with public visibility modifier that is
      ↪ not called internally . Changing visibility level to external increases code
      ↪ readability . Moreover, in many cases functions with external visibility modifier spend
      ↪ less gas comparing to functions with public visibility modifier .
```

C.5 SLITHER

Search for the vulnerabilities:

- right-to-left-override control character is used;
- state variables shadowing;
- functions allowing anyone to destruct the contract;
- uninitialized state variables;
- uninitialized storage variables;
- functions that send ether to arbitrary destinations;
- controlled delegatecall destination;
- reentrancy vulnerabilities (theft of ethers);
- incorrect ERC20 interfaces;
- incorrect ERC721 interfaces;
- dangerous strict equalities;
- contracts that lock ether;
- state variables shadowing from abstract contracts;
- constant functions changing the state;
- reentrancy vulnerabilities (no theft of ethers);
- dangerous usage of tx.origin;
- unchecked low-level calls;
- unchecked send;
- uninitialized local variables;
- unused return values;
- built-in symbol shadowing;
- local variables shadowing;
- constructor called not implemented;
- multiple calls in a loop;
- benign reentrancy vulnerabilities;
- dangerous usage of block.timestamp;
- assembly usage;

- deprecated Solidity Standards;
- un-indexed ERC20 event parameters;
- low level calls;
- conformance to Solidity naming conventions;
- if different pragma directives are used;
- incorrect Solidity version (< 0.4.24 or complex pragma);
- unused state variables;
- conformance to numeric notation best practices;
- state variables that could be declared constant;
- public function that could be declared as external.

Listing C.5: Slither report

```
INFO:Detectors:
 AttributeControl . deleteAttributeRule ( address , string , address , string , bool , uint256 [],
    ↪ BasicAndCommonControl.Actions[]) (ControlChain.sol#128−156) ignores return value by
    ↪ external calls "resourceAttributeRules.pop()" (ControlChain.sol#149)
Reference: https :// github . com/crytic / slither /wiki/Detector−Documentation#unused−return
INFO:Detectors:
Pragma version "0.5.9" necessitates versions too recent to be trusted . Consider deploying with
    ↪ 0.5.3 (ControlChain. sol#1)
Reference: https :// github . com/crytic / slither /wiki/Detector−Documentation#incorrect−versions−of−
    ↪ solidity
INFO:Detectors:
 deleteAttributeRule ( address , string , address , string , bool , uint256 [], BasicAndCommonControl.Actions
    ↪ []) should be declared external :
 − AttributeControl . deleteAttributeRule ( address , string , address , string , bool , uint256 [],
    ↪ BasicAndCommonControl.Actions[]) (ControlChain.sol#128−156)
Reference: https :// github . com/crytic / slither /wiki/Detector−Documentation#public−function−that−
    ↪ could−be−declared−as−external
INFO:Slither:ControlChain. sol analyzed (3 contracts with 38 detectors ), 5 result (s) found
```

C.6 OYENTE

The Oyente's page on GitHub (https://github.com/melonproject/oyente#paper) shows a link (https://www.comp.nus.edu.sg/~loiluu/papers/oyente.pdf) as the one containing the bugs that can be detected by the tool. However the link does not load the file. Therefore, we do not know what types of vulnerabilities it could be detect.

Next, we present the error that was produced when the tool was executed with E-ControlChain.

Listing C.6: Oyente report

```
WARNING:root:You are using evm version 1.8.2. The supported version is 1.7.3
WARNING:root:You are using solc version 0.4.21, The latest supported version is 0.4.19
CRITICAL:root:Solidity compilation failed . Please use −ce flag to see the detail .
```

C.7 MANTICORE

We did not find what vulnerabilities it searches for. We only found its features. Some of them are:

- input generation: Manticore automatically generates inputs that trigger unique code paths
- error discovery: Manticore discovers bugs and produces inputs required to trigger them
- execution tracing: Manticore records an instruction-level trace of execution for each generated input

Next, we present the error that is being produced instead of the report. As can be seen it interprets "calldata" as the name of the variable and not the storage location. This clearly means that it does not support Solidity v0.5 (`https://solidity.readthedocs.io/en/v0.5.0/050-breaking-changes.html#syntax`).

Listing C.7: Manticore report

```
2019-11-16 22:03:57,918: [19] m.main:INFO: Registered plugins: <class 'manticore.ethereum.plugins.KeepOnlyIfStorageChanges'>, DetectExternalCallAndLeak, DetectDelegatecall, DetectSuicidal, DetectUninitializedMemory, DetectInvalid, DetectUninitializedStorage, DetectUnusedRetVal, DetectReentrancySimple, DetectManipulableBalance, DetectReentrancyAdvanced, DetectEnvInstruction, DetectIntegerOverflow
2019-11-16 22:03:57,919: [19] m.main:INFO: Beginning analysis
2019-11-16 22:03:57,923: [19] m.e.manticore:INFO: Starting symbolic create contract
2019-11-16 22:03:57,936: [19] m.e.manticore:ERROR: Errors: Invalid solc compilation ControlChain.sol:61:57: Error: Expected ',' but got identifier
    function setContext(address source, string calldata identifier, int value) isTheAddress(source) external {
                                                        ^--------^

. Solidity failed to generate bytecode for your contract. Check if all the abstract functions are implemented.
2019-11-16 22:03:58,070: [55] m.c.manticore:INFO: Generated testcase No. 0 - NO STATE RESULT (?)(0 txs)
2019-11-16 22:03:58,132: [19] m.c.manticore:INFO: Results in /mcore_8p1zra6a
```

C.8 CONTRACTFUZZER

Search for the vulnerabilities:

- gasless send;
- exception disorder;
- reentrancy;
- timestamp dependency;
- block number dependency;
- dangerous delegatecall;

- integer overflow;
- integer underflow;
- freezing ether.

We did not executed it because we did not achieve the docker image of the ContractFuzzer.

C.9 MYTHRIL

Search for the vulnerabilities:

- delegatecall to untrusted callee;
- weak randomness;
- timestamp Dependence;
- use of deprecated functions;
- unprotected Ether withdrawal;
- assert Violation;
- signature malleability;
- integer overflow and underflow;
- denial of service with failed call;
- unprotected selfdestruct;
- reentrancy;
- unchecked call return value;
- assert violation

The Mythril did not finished its execution as it exceeded our limit of 6 GB of RAM.

C.10 OCTOPUS

We did not find what vulnerabilities it searches for. We only found its features. Some of them are:

- control flow analysis: Octopus can generate a control flow graph;
- call flow analysis: Octopus can generate a call flow graph (function level);
- symbolic execution: Octopus use symbolic execution to find new paths into a program.

We tried to execute both tasks, the control flow graph (CFG) and the SSA. The CFG generated a PDF with a list of more than 300 rectangle blocks side by side each one with the writing “block_H”, where “H” is an apparently unique hexadecimal number. Also, there was no link between the rectangles. The SSA did not finished its execution as it exceeded our limit of 6 GB of RAM. The scenario repeated itself even setting the options “–simplify –onlystatic”, that would make a more superficial analysis.