UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA - VARESE

DiSTA

Dipartimento di Scienze Teoriche e Applicate

P H D T H E S I S

to obtain the title of

Doctor of Science

Specialty : Computer Science

Defended by

GÖKHAN SAĞIRLAR

ENHANCING DATA PRIVACY AND SECURITY IN INTERNET OF THINGS THROUGH DECENTRALIZED MODELS AND SERVICES

Advisor: Prof. Barbara Carminati

Advisor: Prof. Elena Ferrari

defended on October 5, 2018

Jury :

Reviewers :	Dr. Federica Paci	- University of Southampton
	Dr. Dan Lin	- Missouri University of Science and Tech.
President :	Dr. Elena Ferrari	- University of Insubria
Examinators :	Dr. Claudio Agostino Ardagna	- University of Milan
	Dr. Pierluigi Gallo	- University of Palermo

Bismillahirrahmanirrahim

Muhtaç Olduğun Kudret Damarlarındaki Asil Kanda Mevcuttur!

Ne Mutlu Türk'üm Diyene!

Candan öte aileme ithafen...

Acknowledgement

I would like to express the deepest gratitude to my advisors Dr. Elena Ferrari and Dr. Barbara Carminati for their academic guidance. I have learnt a great deal of privacy, security, and, how to do research from you. I will always be thanking you to providing me this opportunity to take the biggest challenge of my life so far when I was a senior bachelor student. Your patience towards my mistakes has always motivated and encouraged me to continue my research. I believe, your guidance through more than 1000 emails and over 100 meetings in last 3 years has transformed me from a motivated but confused rookie to a skilled professional with high potential and goals. Also, I will always be glad for letting me work on different topics as I wish. I hope to continue to be learning from you through all of my career.

I would like to thank Dr. Emanuele Ragnoli for supporting my research activities and helping me to improve my skills. I will always be thankful for your collaboration to my research and your trust to me. There are many things to do and many things to learn, and I will continue to do my best.

I would like to thank Dr. Pietro Colombo. I have learnt many stuff from you about technical matters, social things, and convenient solutions for countless questions that I have asked you. I will always envy your work ethics, discipline, and helpful attitude.

I thank my Ph.D. colleagues Bikash, Tu, Alberto, Stefania, Zulfikar, for good times we spent during my stay in Italy.

Abstract

Wearable devices tracking our fitness activities and health status, smart home technologies supporting home automation services, smart city technologies improving quality and performance of urban services. These are just some examples of Internet-connected "things" clearly proving that the Internet of Things (IoT) is already upon us, impacting our every-day lives. The promise of IoT is making the world smarter, more profitable, autonomous, more connected and more efficient. So far, IoT has already been applied to several environments: healthcare, manufacturing, retail, buildings, cities, automative, transportation, energy etc. Indeed, according to the IHS Markit, as of 2018, number of connected IoT devices has reached 27 billion.[1]

However, challenges posed by IoT have also increased with its popularity. As such, among others, challenges on data privacy, security, and limited decentralization of IoT systems introduce major threats for the future of IoT [140, 117]. Given that, in this thesis, we focus on data privacy and security issues in IoT under the decentralized model.

In the first part of the thesis, we focus on data privacy issues. In a typical IoT scenario individuals' privacy can easily be violated due to the high volume of managed personal data. Particularly, confidential information about individuals may be revealed to unauthorized parties, or, combination of different data may lead to infer sensitive information about individuals. In addition to those issues, in a decentralized IoT scenario, where IoT devices (i.e. smart objects) share data with each other, privacy protection is even more challenging as it is more difficult to control how data are combined and used by smart objects where future operations on data are unknown. Therefore, first challenge that we take in this thesis is enhancing data privacy in IoT with a user-centric model. First, we propose a privacy enforcement framework for centralized IoT systems. Then, we extend it for decentralized IoT systems. In this model, compliance check of user individual privacy preferences is performed directly by smart objects.

Decentralization, if coupled with proper security mechanisms, would have many advantages over centralized infrastructures for IoT, such as, among others: better privacy guarantees for data owners, more resilient and secure systems, improved interoperability between services, concerted and autonomous operations. Notably, blockchain is a promising decentralized platform due to its ability to achieve distributed consensus [128] and with it's intrinsic security features to ensure data integrity. Given that, we shift our focus to address issues related to security and decentralization of IoT systems with blockchain based systems. At this purpose, we first deal with security issues in IoT, as resource constrained IoT devices do not employ strong security

[1] cdn.ihs.com/www/pdf/IoT_ebook.pdf

mechanisms and they are easy targets for attackers. One of the most relevant attack is when attackers take advantage of the vulnerabilities of IoT devices, and compromise them to add them to *botnets*, that are collection of compromised internet computers controlled by attackers. Then, attackers use their botnets for their malicious purposes, such as performing Distributed Denial of Service (DDoS) attacks. Moreover, to increase attacks' success chance and resilience against defence mechanisms, modern botnets have often a decentralized P2P structure, which makes them harder to detect. In order to deal with this problem, we take a first step towards detecting P2P botnets in IoT, by proposing *AutoBotCatcher*. AutoBotCatcher exploits a Byzantine Fault Tolerant (BFT) blockchain, in order to perform collaborative and dynamic botnet detection by collecting and auditing IoT devices' network traffic flows as blockchain transactions. Secondly, we take the challenge to decentralize IoT, and design a hybrid blockchain architecture for IoT, by proposing *Hybrid-IoT*. In Hybrid-IoT, subgroups of IoT devices form PoW blockchains, referred to as PoW sub-blockchains. Connection among the PoW sub-blockchains employs a BFT inter-connector framework. We focus on the PoW sub-blockchains formation, guided by a set of guidelines based on a set of dimensions, metrics and bounds.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet, one of the greatest inventions of humankind, started out as a government funded defense project in 1962 and evolved to ARPANET in 1969, and significantly, this opened the way to innovate. In early 90's Internet saw Tim Berners-Lee's invention of World Wide Web (WWW) [18] to merge networked information retrieval and hyper-text documents. With many more inventions over last decades, the Internet has experienced a great success. Indeed, today Internet is evolved into a huge network that has many services, such as: e-commerce web-sites, online social networks, personal web blogs, news media sites, and so on.

The Internet, as a tool, opened a way to generate new technologies and inventions for engineers and researchers. In 1982, a group of graduate students came up with the idea of connecting their building's soda machine to the Internet in order to check whether the machine is empty or sodas are cold before going to the machine to buy a soda.[1] *Necessity was the mother of invention* as we learned from the famous english proverb[2], and the Internet was the enabling tool for the invention. According to many sources, this is considered as the very first example of a new kind of devices, which we call today *Internet of Things*, or short IoT. Yet, the term IoT was coined way later by British inventor Kevin Ashton[3]. Roughly, the term IoT refers to the network of physical objets, so called *things*, such as sensors, RFIDs or various kinds of physical devices that are able communicate with each other, server or cloud over Internet. IoT transforms physicals objects from being traditional to smart, by enabling them to see, hear, and perform tasks, by letting them to share information with each other [4]. Indeed, the promise of IoT is making various environments smarter, efficient, more connected and autonomous with intelligent decision making [4].

In the last decade, we have witnessed a great growth of IoT. Indeed, according to the IHS Markit, as of 2018, number of IoT devices is over 27 billion. Moreover, IoT devices have already been everyday objects in our daily lives with wearable devices, smart home applications, smart cities and so on. IoT has also been applied to several industries, such as food processing, agriculture, healthcare, environmental monitoring, transportation and logistics, mining production

[1] ibm.com/blogs/industries/little-known-story-first-iot-device
[2] In original it is first used by William Horman in his book *Volgar* in Latin as: *"Mater artium necessitas"*.
[3] rfidjournal.com/articles/view?4986

monitoring, security surveillance and so on [42]. As suggested by its growth in number of devices and many use cases, as also stated by the US National Intelligence Council [98], IoT is certainly one of the disruptive technologies today.

Despite its potential, many use cases, great growth, and many futuristic ideas for its future, IoT also comprises important challenges. Some of the key challenges are, among others: data privacy, security, and decentralization of IoT systems [140, 117, 4]. In this thesis, we deal with data privacy and security issues in IoT to support IoT to reach it's full potential. In doing that, we take *decentralization* as main approach and model, as we believe it will play fundamental role in future IoT systems. In the following, we describe the main motivations behind dealing with on data privacy and security issues of IoT, and why decentralization matters for IoT.

Data privacy issues in IoT

IoT is impacting our every-day lives with many applications that process personal and confidential data, such as wearable devices that track fitness activities and health status of individuals. With extensive number of IoT devices that are collecting and processing personal and confidential data, naturally, individuals privacy protection arises as a major challenge to overcome. In fact, data privacy issues in IoT has been widely investigated in the literature [86, 103, 140, 117]. Moreover, there are also data protection regulatory laws and frameworks, such as European Union General Data Protection Regulation (GDPR) [106]. In order to ensure European Union citizens' personal data protection GDPR introduces data protection principles and data subjects' rights [84]. As stated by GDPR, data subjects should be given more transparency on how their data is processed and they should be in charge of their personal data that refers to any information related to an identifiable person.[4]

First issue related to data privacy in IoT is revealing personal information to unauthorized parties/data consumers. Second privacy issue arises due to further processing of data. In that, sensitive information of data owners may be inferred through data analytics processes (e.g. data joins, aggregations). For example, by joining and combing data related to movements, heart beats and breath rate, it is possible to infer possible psychological disorders due to insomnia. Privacy issues get even more complicated when we consider a decentralized IoT scenario where devices are able to process and share data with each other. Indeed, in such a scenario, we do not have prior knowledge about how data are going to be shared and processed. Therefore, third issue on data privacy is protecting data owners' privacy even in unknown future use of data. For example, walked distance and number of steps sensed by a smart watch can be combined to infer individual's height information with some approximation. Starting from the height information and by combining it with weight information sensed by a smart scale one can infer body mass index of individuals. As such, future operations performed over data may introduce additional privacy violations, thus privacy of the users should be enforced in the future use of their data.

[4] eugdpr.org/the-regulation/gdpr-faqs

Security issues in IoT

Device vendors do not take security as primary goal in producing IoT devices [9], in order to produce IoT devices quickly to catch market trends and protect their business profits. However, this made IoT an amplifying platform for cyberattacks, where malicious parties can easily take control of IoT devices [140, 117, 9]. Threats caused by compromised IoT devices present serious security issues for online services' security, such as: attackers may control compromised devices for their malicious purposes and form malicious botnets, or steal confidential information stored by devices. We focus on malicious botnet threat due to their relevance and huge destructive effects on victims. A malicious botnet is a collection of compromised Internet computers being controlled remotely by attackers for malicious and illegal purposes, such as performing cyber-attacks (e.g.,Distributed Denial of Service (DDOS) attacks)[132]. Typically, attackers try to infect as many devices as possible in order to increase power and effect of their attacks. Indeed, in 2016 the Mirai malware [8] infected many IoT devices in order to perform DDOS attacks by generating extensive amount of Internet traffic (more than 1 tbps).

Why decentralization matters?

Today, most of the IoT systems are centralized cloud based computing infrastructures. However, centralized IoT infrastructures present a number of drawbacks to IoT. Such drawbacks are, among others: high cloud server maintenance costs, weak adoption and support for time-critical IoT applications, security issues (e.g., Single Point of Failure (SPoF)). Moreover, different parties (device vendors, service providers) have to trust each other or another third party in order to collaborate in centralized systems. This limits the interoperability between different IoT applications and services.

On the other hand, decentralization, if achieved, would have many advantages over centralized infrastructures. The most prominent outcomes of decentralizing IoT is achieving distributed consensus among IoT devices, and if properly governed it will improve security of IoT systems, and, providing better privacy guarantees to the users with effective data protection mechanisms. With that, IoT devices are able to perform concerted and autonomous operations, which increases data utility of IoT systems due to enhanced data processing and information generation. Moreover, amount of data transferred to the cloud for processing and cloud maintenance costs are reduced in decentralized systems. The last but not the least, decentralization would be instrumental to improve security and privacy of the managed data by assuring data security and accountability, and eliminating SPoF problem of centralized systems.

Decentralized IoT systems have to be able to process high throughput of transactions and scale to many peers in achieving consensus without a trusted central authority. Therefore, IoT decentralization requires frameworks that employ scalable and performant distributed consensus among peers. Lack of such frameworks has been a bottleneck against successful decentralization of many domains including IoT. But, rise of Bitcoin [92], the peer-to-peer digital currency, has the potential for paradigm shift in decentralization. Particularly, invention of *blockchain* as underlying technology of Bitcoin, has opened a way to overcome distributed consensus bottlenecks in a decentralized setting for large scale applications. Fundamentally,

blockchain is the concept of a distributed ledger maintained by a peer-to-peer network and it allows peers of a p2p network to reach consensus without needing a central authority and establishing trust. Moreover, blockchain employs cryptographic techniques, such as; asymmetric cryptography and hashing, enables to ensure integrity of the blockchain data structure (cfr. Chapter 2 for background information on blockchain). Given that, we consider blockchain as a promising decentralization platform. Therefore, we use blockchain technology as a tool in designing decentralized IoT systems.

1.1 Thesis Objective

By referencing to the above discussed motivations, we formulate our objective as follows:

> **OBJECTIVE: Establish and develop alternative architectures, frameworks, and models to enhance individuals privacy in IoT scenarios and improve security of IoT systems, especially under decentralized model, that are efficient and practical.**

A notable challenge in enhancing data privacy and security in IoT under decentralized model is developing lightweight and scalable decentralized solutions. In fact, decentralization comes with a cost, as it might introduce additional complexity and overhead, especially when considering cases that require distributed consensus. Therefore, in this thesis, in addition to challenges related to enhancing data privacy and security, we also deal with the intrinsic challenge of making lightweight and scalable decentralized systems for IoT. To this end, we use blockchain technology as decentralization tool.

1.2 Terminology

We provide definitions to the key concepts that are used in this thesis for reader's convenience. These definitions are mainly given based on the common understanding in the literature, but are also related to how they are used in this dissertation.

- **Data:** Information generated by IoT devices.
- **Smart objects:** IoT devices that are not only able to sense data, but also able to processes and aggregate data, and interact with other IoT devices.
- **Privacy:** Individuals' right to control how their data could be shared with others (e.g. third party data consumers).
- **Security:** Protection of data from disclosure, alteration, destruction and loss [116].
- **Data owner:** Owner of the IoT device that generates data.

- **Blockchain:** A distributed data structure shared across peers of a p2p network. Blockchain data structure is secured using cryptography to protect integrity of its records. We provide background on blockchain technology in in Chapter 2.

1.3 Main Contributions

In summary, this thesis provides the following main research contributions:

- A core framework to enforce user's privacy in centralized IoT systems with low overhead. For this framework, we design a novel *privacy preference* model, which allows to handle privacy with a user-centric approach. Particularly, with their privacy preferences users are able to state: which portion of their personal data can be accessed and how these data can be combined; and what can not be inferred from their data through any kind of analytics processes. Experimental results show the efficiency of the proposed enforcement mechanism.

- An extension of the above mentioned core framework for decentralized IoT smart objects with low overhead. The novel problem that this framework tackles is privacy enforcement without a centralized reference monitor. In order to solve this problem, we leverage on the privacy model developed for the core framework extend it to deal with decentralized ecosystems. In contrast, the enforcement mechanism leverages on ad hoc-designed security meta-data, called *Privacy-Enhanced Attribute Schema (PEAS)*, attached to each piece of data for decenralized privacy enforcement. Privacy preference compliance check is performed before data are going to be released to the third party data consumers. The proposed framework has been tested on different scenarios, and the obtained results show the feasibility of our approach.

- Architectural design of the first blockchain-based botnet detection architecture for IoT, called *AutoBotCatcher*. AutoBotCatcher performs dynamic and collaborative botnet detection and prevention for IoT with dynamic community detection methodology. In AutoBotCatcher, a blockchain is exploited to enable multiple parties to collaborate for botnet detection without needing to trust each other or a central server/database. Moreover, blockchain is exploited to model the botnet detection process as a set of shared application states of parties collaborating in botnet detection.

- Hybrid blockchain architecture to decentralize and secure IoT, called *Hybrid-IoT*. Hybrid-IoT exploits Proof of Work (PoW) blockchains to achieve distributed consensus among operations performed by IoT devices. By virtue of that, in Hybrid-IoT, IoT devices are able to collectively and autonomously execute their operations by forming their machine to machine (m2m) communications in form of blockchain transactions. This also guarantees accountability and security of the stored data. To measure performance of the PoW blockchains (i.e., transaction throughputs) we define a set of PoW blockchain-IoT integration metrics. We also provide a measurement study of the performance of PoW

blockchains in IoT, subject to the PoW blockchain-IoT integration metrics. We test performance and security of the proposed approach.

1.4 Thesis Organization

The dissertation is organized into eight chapters, briefly described in the following:

Chapter 2: Background - Blockchain

In this chapter, we first provide background information on the cryptography techniques used in blockchains, then we present blockchain, and, finally, we introduce the consensus problem and consensus protocols employed by blockchain.

Chapter 3: Literature Review

We review the literature on proposals dealing with data privacy in IoT, botnet detection, and blockchain based systems for IoT.

Chapter 4: Enhancing User Privacy in IoT

In this chapter, we present the core framework to enforce user's privacy in centralized IoT systems.

Chapter 5: Decentralizing Privacy Enforcement in IoT

In this chapter, we present the enhanced privacy enforcement framework, that is tailored to enforce user's privacy in decentralized IoT systems consisting smart objects.

Chapter 6: Blockchain-based P2P Botnet Detection for IoT

In this chapter, we present AutoBotCatcher, a blockchain-based P2P botnet detection mechanism for IoT.

Chapter 7: Hybrid Blockchain Architecture for IoT

In this chapter, we present Hybrid-IoT, a hybrid blockchain architecture for IoT.

Chapter 8: Conclusions

In this chapter, to sum-up this thesis, we discuss main arguments and contributions. This chapter outlines the future plan as well.

1.5 Related Publications

The research activities described in this thesis have brought to the following publications:

- Barbara Carminati, Elena Ferrari, Pietro Colombo, Gokhan Sagirlar, "Enhancing user control on personal data usage in Internet of things ecosystems", in *2016 IEEE International Conference on Services Computing (SCC)*, pp. 291 - 298, in San Fransisco, USA;
- Gokhan Sagirlar, Barbara Carminati, Elena Ferrari, "Decentralizing Privacy Enforcement for Internet of Things Smart Objects" in Elsevier Computer Networks Journal, Volume 143, 9 October 2018, Pages 112-125;
- Gokhan Sagirlar, Barbara Carminati, Elena Ferrari, "AutoBotCatcher: Blockchain-based P2P Botnet Detection for the Internet of Things", in *2018 IEEE International Conference on Collaboration and Internet Computing (CIC)*, in Philadelphia, Pennsylvania, USA;
- Gokhan Sagirlar, Barbara Carminati, Elena Ferrari, John D. Sheehan, Emanuele Ragnoli "Hybrid-IoT: Hybrid Blockchain Architecture for Internet of Things - PoW Sub-blockchains", in *2018 IEEE International Conference on Blockchain*, in Halifax, Canada;
- Gokhan Sagirlar, Barbara Carminati, Elena Ferrari, John D. Sheehan, Emanuele Ragnoli "Hybrid-IoT: Scalable and Hybrid Blockchain Architecture for Internet of Things", *Under preparation*;

Chapter 2

Background

The work conducted on this thesis is on enhancing data privacy and security of IoT systems under the decentralized model. Given that, in this section, we provide background information on architecture models and protocols for IoT (ie. Section 2.1). On the other hand, towards the thesis, we exploited different technologies and tools. The most significant one, among others, is the blockchain technology, as we use it to design a P2P botnet detection framework (Chapter 6), and a hybrid blockchain architecture for IoT (Chapter 7). Therefore, in this chapter (ie. Section 2.2) we also provide the background information on blockchain technology.

2.1 IoT Architectures and Protocols

In this section we provide an overview about architecture models for IoT systems in Section 2.1.1 and commonly used protocols in IoT in Section 2.1.2.

2.1.1 IoT Architectures

Despite some efforts to generate a reference IoT architecture (e.g. IoT-A [31]), there is no de facto architecture model for IoT. Yet, by reviewing relevant works on the literature, such as [138, 11, 73, 57, 4], we present common patterns in IoT architectures. As such, the basic IoT architecture is the 5 layered model as presented in [4] and [73], that includes following layers: *Object (Perception) Layer* where IoT devices exist, *Object Abstraction (Network) Layer* where IoT data are securely transferred (e.g. ZigBee, RFID) and managed by cloud computing or data management systems, *Service Management (Middleware) Layer* where data generated by heterogenous IoT devices are processed, decisions made, services are paired and delivered to the requesters, *Application Layer* where services provided to customers, and *Business Layer* where overall systems are managed with a business model.

An important aspect in discussing architecture models for IoT is the group of *elements* that are required to deliver the functionality of IoT systems. According to [4], the six main elements for IoT are: *Identification*, refers to naming and matching IoT services and devices; *Sensing*, refers to gathering data from the physical world within the network; *Communication*,

refers connecting IoT devices together securely to deliver smart services; *Computation*, refers processing of the IoT data and represents brain of the IoT application; *Services*, refers to IoT applications' services for customers or other services; and *Semantics*, refers to ability to extract knowledge smartly to provide required services.

There are two types of IoT architectures: *centralized* and *decentralized*. These architectures differentiate mainly in the way how they handle *computation*, *services* and *communication* elements of IoT systems. In the following, we provide background information on centralized and decentralized IoT architectures.

Centralized IoT Architectures

In centralized IoT architectures, IoT devices are linked to a central hub, such as server or cloud, which is used to provide backend services to smart devices [141]. In centralized IoT architectures, the main objective of IoT devices is sensing data from physical world around them. Then, they share sensed data with the central hub.The central hub performs operations related to computation, services and semantics elements of IoT systems, such as real-time analysis, event processing and management, data management, decision taking etc. To sum up, in centralized IoT architectures objectives of IoT devices are limited, and operations that require processing and data storage are left to the central hub. In the following we briefly discuss *cloud computing* based IoT systems as example to centralized IoT architectures.

Cloud-based IoT Systems. Essentially, the amount of data generated by IoT systems are huge and often referred as *big data*. Many IoT scenarios, such as smart grid, health monitoring etc., requires to perform real-time analytics, processing, and decision taking on such big data generated by IoT platforms. Cloud computing offers a new way to manage and process big data generated in IoT scenarios [4] and currently vast majority of IoT applications are based on cloud-based solutions. In cloud-based architectures, data sensed and sent by IoT devices are pooled at a single or geographically distributed cloud infrastructures. Where, they process data generated by IoT devices, generate services for IoT applications, take decision according to the stored data, and provide services to the users and customers when demanded. There are many cloud platforms that are in use today, such as Amazon Web Services[1] and IBM Watson IoT[2]. In addition, academic literature offers several works that propose cloud based IoT systems such as [2, 53, 74, 95, 90], where cloud computing have been used to store, process and manage IoT data for various applications scenarios.

Despite their wide usage, centralized systems systems have many drawbacks for IoT. In this thesis, the most relevant drawbacks addressed are data privacy and security issues of centralized IoT systems that are discussed in Chapter 1.

Decentralized IoT Architectures

Decentralized IoT architectures are essentially decentralized systems of cooperating smart objects. Such smart objects are able, not only to sense data, but also to interact with other objects

[1] aws.amazon.com/iot
[2] ibm.com/internet-of-things

and to aggregate data sensed through different sensors. This allows smart objects to locally create new knowledge, that could be used to make decisions, such as quickly trigger actions on environments, if needed. Smart objects are very heterogeneous in terms of data sensing and data processing capabilities. Some of them can only sense data, others can perform basic or complex operations on them. Such a scenario enacts the transition from the Internet of Things to the Internet of Everything, a new definition of IoT seen as a loosely coupled, decentralized system of cooperating smart objects, which leverages on alternative architectural patterns with regards to the centralized cloud-based one. Where, unlike centralized IoT architectures, computation and service elements of IoT platforms are also objectives of such smart objects. As such, devices perform some operations over sensed data and take decisions, depending their computation and storage capabilities. How the communication elements handled is also different than centralized architectures, since smart objects share their data with other smart objects, maybe autonomously, provided that they are able to convert and understand each other's communication protocols (i.e. with the help of a middleware layer). In the following, we briefly discuss *fog computing* as an example to decentralized IoT architectures.

Fog computing. Fog computing is a distributed computing paradigm that extends cloud to the edge of the network [21], where extensive amount of heterogeneous decentralized and ubiquitous IoT devices and gateways communicate and cooperate with each other in the network to perform computation and storage tasks [129]. Fog computing is not an alternative to the cloud computing, it is a supplementary paradigm that improves localization of services and reduces amount data transferred to the cloud, thus it acts as a bridge between smart devices and large-scale cloud computing and storage services [4]. Specifically, in fog computing devices at different hierarchical levels are equipped with "intelligence" to examine whether an application request requires the intervention of the cloud computing tier or not. It also has the potential to reduce delay in delivering services to end users due to localization of services with close proximity. In fact, as measured in [113], as number of latency-sensitive IoT services increase, fog computing outperforms cloud computing by decreasing overall latency of the services. Specifically, in [113] Sarkar *et al.* showed that, in an environment where 50% of applications are requesting real-time services, fog computing compared to cloud computing reduces the overall service latency by 50.09%. The fog play important role in many IoT scenarios, such as connected vehicles, smart grids, wireless sensor and actuator networks [21]. Additionally, academic literature offers interesting proposals that make use of fog computing in IoT applications such as [126, 62, 1].

Given smart objects' increasing storage, communication and computing capabilities in parallel to the Moore's Law, we can expect to see increase in number of decentralized IoT systems [125]. Despite their potential and benefits, as described in Chapter 1, decentralized systems face many new challenges on assuring data privacy and security are addressed in this thesis.

2.1.2 IoT Protocols

Thanks to adopted protocols by available IoT devices, IoT is a very heterogenous domain. Therefore, in order to increase interoperability of IoT services and applications, various working groups and consortiums from many groups, such as *Institute of Electrical and Electronics Engineers* (IEEE), *European Telecommunications Standards Institute* (ETSI), *International*

Telecommunication Union-Telecommunication (ITU-T) and *Internet Engineering Task Force (IETF)*, are working to generate standardized protocols that remove gaps between different protocols. In the following we present some prominent examples of such efforts through protocol standardization, namely *The Constrained Application Protocol (CoAP)*, *Message Queue Telemetry Transport (MQTT)*, *The Internet Protocol version 6 (IPv6)* and *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN)*.

The Constrained Application Protocol (CoAP)

The IETF Constrained RESTful Environments (CoRE) working group generated CoAP [115, 22] with aim to make *Hypertext Transfer Protocol*'s (HTTP) *Representational State Transfer* (REST) paradigm available to restrained IoT devices and networks. Given that, CoAP protocol stack is similar to, but less complex than, the HTTP protocol stack [22], and like HTTP it is a application layer protocol. Since it shares the REST architecture with the HTTP protocol, CoAP is capable of interacting with several device types easily. In order to reduce complexity of HTTP, CoAP uses *User Datagram Protocol* (UDP) as transport layer protocol rather than Transmission Control Protocol (TCP) used by HTTP.

On top of the UDP, CoAP deals with asynchronous nature of interactions and the request/response interactions [115]. Moreover, for applications that require security CoAP can be used on top of Datagram Transport Layer Security (DTLS) that protects confidentiality and integrity of message contents [22]. Where, device may adopt different security levels that are divided into four modes [115] as following: *NoSec* mode where DTLS is disabled so there is no protocol level security; *PreSharedKey* mode where DTLS is enabled and there is a list of shared keys (keys include a list of nodes can be used to communicate); *RawPublicKey* mode where DTLS is enabled and device has a asymmetric key without a certificate; and, *Certificate* mode where DTLS is enabled and device has a asymmetric key with a certificate signed by some common trust root. CoAP is also capable of carrying different types of payload and it integrates different data model types such as XML and JSON.

Message Queue Telemetry Transport (MQTT)

MQTT is a lightweight publish/subscribe messaging protocol that was invented in 1999 by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom [130] and in 2013 it became an OASIS standard. MQTT is suitable for low-bandwidth, unreliable and high-latency networks and is designed for constrained devices.[3] MQTT's publish/subscribe model has three main components: *publisher*, *subscriber* and *broker* [4]. By taking the subscriber role IoT devices subscribe to their interest of topics published by publishers. When a publisher publishes a message, MQTT transfers the message via broker to every IoT device that has subscribed the topic of the published message.

Unlike CoAP, MQTT works on top of TCP/IP protocol. For applications that require to take security measurements, MQTT connections can be complemented with Transport Layer

[3] mqtt.org/faq

Security (TLS). Even though MQTT runs on TCP/IP protocol suite rather than more light-weight UDP, it is designed to be low overhead, and thanks to its publish/subscribe model, subscribers do not respond to messages they have received from a publisher topic they have subscribed [68]. Therefore lower network bandwidths and less device resources are used. Moreover, as it runs over TCP/IP, it attempts to ensure some degree of assurance of delivery, even in unreliable networks.

The Internet Protocol version 6 (IPv6)

IPv6 was developed by IETF as new version of the Internet Protocol as the successor of the IPv4. IPv6 has been standardized by IETF in July 2017 with the publication of RFC8200. In order to solve problems related to the depletion of unallocated addresses in IPv4 address space, IPv6 expanded the address size 32 bytes of IPv4 to 128 bytes, and thus is able to support much greater number of addresses and more levels of address hierarchy [44]. In addition to that, in order to reduce IPv6 header packets' network bandwidth consumption and packet processing cost, some of the fields included in IPv4 header has been dropped. By using *Security Architecture for the Internet Protocol* defined in RFC4301 [72], IPv6 packets' integrity and confidentiality can be protected. In addition to that, IPv6 packets can be also protected with upper layer protocols such as TLS and Secure Shell (SSH) [44].

IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN)

In 2007, IETF's 6LowPAN working group defined 6LowPAN protocol to enable IPv6 packets to be carried on top of low power wireless networks, specifically IEEE 802.15.4, to apply the Internet Protocol (IP) to the constrained and small devices [89]. Thanks to 6LoWPAN, existing network architecture can be used and constrained devices can easily connect to other IP based networks without proxies and translation gateways [114]. Due to complexity and performance reasons, the most common transport protocol used with 6LoWPAN is the UDP [114]. In 6LoWPAN, the necessity of configuration servers DHCP and NAT is eliminated. Moreover, 6LoWPAN implementations can easily fit into 32K flash memory [89]. Given that, in 6LoWPAN overhead for the most common packets are much less than other protocols [89].

2.2 Blockchain Technology

The term *blockchain* has been first used to define the underlying technology of the first digital currency (aka cryptocurrency) called Bitcoin[4]. Bitcoin is proposed by a person or group using pseudonym Satoshi Nakamoto in 2008 [92]. Main invention of Bitcoin was its ability to remove any trust relation to perform money transfers.

In essence, blockchain is a cryptographically secure distributed data structure shared across the peers of the p2p network, where trust among peers to achieve consensus between peers. In this section we provide an overview about blockchain technology, specifically by focusing

[4]bitcoin.org

on the parts that will be needed later. First, in order to lay the foundation of important concepts, we provide background information on cryptography techniques used by blockchains in Section 2.2.1. Then in Section 2.2.2, we elaborate the discussion on the blockchain technology. In Section 2.2.3 we introduce the consensus problem and discuss consensus methodologies of blockchains. Finally, in Section 2.2.4, we present three blockchain platforms relevant to this thesis.

2.2.1 Cryptography

Cryptography is the study, with a long history, of mathematical techniques related to the information security (e.g. data integrity, authentication, data confidentiality) [71]. Cryptographic techniques lays the foundation for many modern security tools, such as encryption techniques, digital signatures etc. From distributed systems point of view, it is probably the key enabling technology for protecting security of the distributed systems [6]. In fact, as a distributed system, blockchain technology makes extensive use of cryptographic techniques. Thus, let us review and introduce some of the most fundamental cryptography concepts that are relevant to the blockchain technology, namely; *asymmetric cryptography*, *digital signatures*, and *hash functions*.

Asymmetric cryptography (Public key cryptography)

Asymmetric cryptography algorithms use a pair of keys, namely; *public key* and *private key*, where each component of the pair are used to verify the other and to perform two counterpart cryptographic operations (e.g. encryption - decryption) [116]. Private key is the secret component of the asymmetric key pair, which is essentially meant to be known only by the owner of it as a secret information. On the other hand, public key is the public component of the asymmetric key pair where owner can disclose it to the any part that she wishes. As an example use case of to asymmetric cryptography algorithms relevant to the blockchain, let us consider encryption-decryption algorithms. Let us assume, a user U_1 wishes to send message m to user U_2 securely, meaning that by assuring integrity and and confidentiality of the message. Let us also assume U_1 has the asymmetric cryptography key set, respectively public key and private key, Pb_1 and Pr_1, and U_2 has Pb_2 and Pr_2. In order to send m securely, U_1 encrypts m with the receiver's public key Pb_2 with an encryption algorithm that U_2 is aware of (e.g. RSA algorithm [107]), and then sends the message to the U_2. The caveat that ensures confidentiality of the message is: only way to decrypt this encrypted message is using Pr_2, which we assume only U_2 has.

Digital signatures

In above example scheme, if U_1 also uses his private key Pr_1 in encrypting m, authenticity and integrity of the message is also ensured, as U_2 can decrypt it by using public key Pb_1 and thus, she can be sure that this message has been generated by U_1 and has not been modified on the way (as it would require to have Pb_1 to encrypt modified message m). This is an example of the

methodology known as *digital signatures*, and is an application of asymmetric cryptography. The basic idea of the digital signatures is that, they can be created by only one, but can be read by everyone [6].

Hash functions

Essentially, hashing refers to mapping larger domains to smaller ranges [71]. In cryptography domain, hash functions are used to ensure data integrity, where they map a variable length string to fixed-length string [116]. Conceptually a good hashing algorithm never has collusions, meaning that there is no pair of inputs that would let algorithm to generate same hashed output. In this thesis, we assume that considered hashing algorithms are collusion-free. Today, algorithms belonging to *Secure Hash Algorithms (SHA)* family, such as SHA-2 are widely used by blockchain protocols such as Bitcoin. Hash functions are extensively used in blockchain systems to ensure integrity of the data structure by connecting blocks to each other in chain structure (explained in detail below). In addition to that, Proof of Work blockchains (explained in detail below), uses hash functions as proof of work functions.

2.2.2 Blockchain

Blockchain relies on the concept of a distributed ledger maintained by a peer-to-peer network [128]. Novelty of the blockchain technology lies in its ability to achieve coordination and verification of individual activities carried out by different parties without a centralized authority or trusted third party, that allows decentralization of application execution with concerted and autonomous operations.

In blockchain, *transactions* transfer information (i.e., data packets) between peers. They have a unique identifier (transaction-id), input data, and are bundled into data chunks, referred to as *blocks*. Block generator peers of the blockchain broadcast blocks by exploiting public-key cryptography. Blocks are recorded in the blockchain with an exact order.

Briefly, a block contains: a set of transactions; a timestamp; a reference to the preceding block that identifies the block's place in the blockchain; an authenticated data structure (e.g., a Merkle tree) to ensure block integrity.[5] The block height is block's distance to the genesis block, which has the height 0. An example blockchain containing four blocks is presented in Figure 2.1.

Blockchains can be classified into two groups as *public* and *permissioned* according to their way of regulating peers' participation in blockchain operations. Particularly, in public blockchains, any peer can read and write to the blockchain, meaning that anyone can participate in the consensus process. Whereas, in permissioned blockchains, only a set of previously identified peers can write to the blockchain and participate in the consensus, and, read rights may be public or limited to pre-identified peers.

[5]Block structure varies in different blockchain protocols, here we list the most common elements.

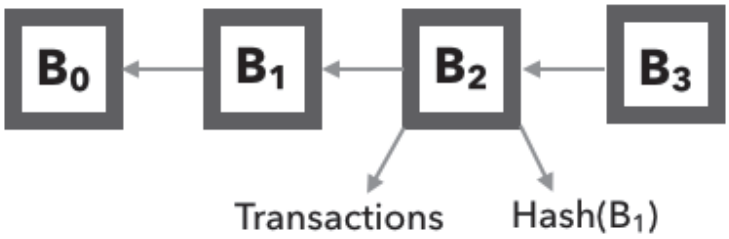

Figure 2.1: An example blockchain with 4 blocks

Stale blocks

Blockchains may have forks, aka different branches in the chain structure, due to malicious manipulations or propagation delays. The longest fork of the blockchain is accepted as the main branch (for rest of the thesis we refer blocks included to the main blockchain as genuine blocks), and remains as the agreed [128]. In general, blocks in shorter forks do not included in the blockchain and they referred as *stale blocks*, and transactions in stale blocks are considered as unprocessed by the network.

Smart contracts

Modern blockchains employ deterministic and self-executing contractual clauses called *smart contracts*. The smart contract concept was first introduced by Szabo in [124] as: "a computerized transaction protocol that executes the terms of a contract". Smart contracts are executable scripts stored forever in the blockchain, where nobody can modify or control them. They have unique address on the blockchain, and their clauses can be triggered by peers via sending transactions to execute them.

2.2.3 Consensus

In essence, any kind of distributed system includes a set of processes (i.e. abstract units that are able to perform computations) and in order to execute properly it seeks to achieve some kind of cooperation among these processes [28]. Consensus is a form of agreement among set of processes. Processes use consensus to agree on a common result value after their operation [28]. In a distributed setting, achieving consensus between a set of processes is not an easy objective, considering the fact that anyone of processes may fail or crash, communication among processes may be delayed or even blocked (i.e. due to network latency and/or network partition), or some of processes may act maliciously and not follow the protocol. In the literature consensus problem has been widely studied [76, 77, 32, 83, 28], and many different models and systems has been proposed with various fault model abstractions of processes. Please refer to [28] for broad discussion on processes in distributed systems, consensus problem, and different consensus models.

Consensus Protocols

As blockchains are essentially distributed systems, consensus problem is one of the most fundamental problems that must be handled by blockchain protocols. In fact, different participants in the blockchain have to achieve consensus on the latest state of the ledger in order to achieve coordination on the processes that they perform on the ledger. There are different methodologies to achieve consensus in the blockchain, we explain the related consensus protocols in the following.

Proof of Work (PoW) protocols. As introduced by Back in HashCash [14], PoW consensus mechanisms rely on the condition of doing some computation that requires to use hardware resources and energy to prove legitimacy of the performed operation. Parallel to that, in blockchains using PoW-based consensus protocols (for the rest of the thesis we will refer such blockchains as PoW blockchains), such as Bitcoin [92], block generators have to solve a cryptographic puzzle to generate a valid block. Roughly, main mechanism of the PoW is as follows: upon forming a new block, block generators add a nonce to the block and take hash of the block; if the hash value satisfies the predefined threshold, then they can seal the block and publish it to the blockchain network. Block generation operation is called *mining* and block generators are referred as *miners.* In PoW consensus, it is hard to generate a block, however it is easy to confirm its validity once it is generated. The main virtue of PoW is preventing instant block generation to reduce conflicts such as double spending and sybil attacks.

PoW consensus protocols requires to have a system where majority (i.e. one-half plus one of all peers) of the mining power follows the protocol, aka are honest peers. If majority is not honest, malicious miners may be able to generate malicious blocks (i.e. blocks that consist bogus transactions) more frequently than the honest minority and have the longest branch of the blockchain, which violates correctness of the consensus. Therefore, throughout this thesis, we assume that in PoW blockchains, at least more than one-half of the total mining power of the blockchain is honest.

PoW is a probabilistic consensus method, meaning that possibility of a block or transaction being in the correct branch of the blockchain increases with the more blocks added to the blockchain as confirmations. Indeed, it is harder for an attacker to generate more blocks to form an alternative branch of the blockchain when the genuine blockchain is longer as each block requires to solve PoW puzzle.

Byzantine Fault Tolerant (BFT) protocols. Let us first briefly describe the concepts called Byzantine processes and Byzantine Fault Tolerance. Byzantine processes are malicious processes that may fail arbitrarily in any possible way from it's algorithm and task [28]. For example, Byzantine processes may not follow the protocol that they have assigned to, they may stop responding, they may reject connection request, they may selectively drop messages or they may lie and propagate false information and so on. A distributed system designed to be BFT must be able to operate correctly and achieve consensus in existence of such arbitrary behaving processes. In order to achieve BFT consensus, it is a very-well known fact that, in an asynchronous network (where messages between processes may delay for unbounded times), the best we can do is to assume that at most less than one-thirds of the all processes are

Byzantine. If this assumption does not hold, simply the processes in the distributed system can not achieve the consensus. Therefore, throughout this thesis, whenever we refer to BFT consensus protocols, we are intrinsically assuming that Byzantine processes in the system are less than one-thirds of the total process amount.

Today, various blockchain protocols exploit BFT consensus methodologies (for the rest of the thesis, we will refer such blockchains as BFT blockchains), such as Hyperledger [7] and Tendermint [26]. Such protocols depend on state replication between block generators. In BFT blockchains, a block generator selected (via some leader election algorithm) to process transactions and generate blocks for the next round (i.e. for certain time period or number of blocks). Then, all blocks are broadcasted to all other block generators, which are entitled to check validity and correctness of the block generated by the leader of that round. Block generators add a block to their local copy of the blockchain, aka replication, if at least two-thirds plus one of the block generators have approved correctness of that block. Different BFT blockchains apply different algorithms in broadcasting, verification and signing of the blocks.

We want to underline that, there are other protocols that are gaining popularity in the blockchain space, such as *Proof of Stake (PoS)* protocols. In PoS consensus model, block generator selection depends on part of peers' wealth that they have voted as their stake in the block generator selection process. We will not dive deep into PoS blockchains, as they are not used in this thesis.

Comparison of PoW and BFT protocols

In this thesis, we exploit blockchains for IoT, where *scalability* is a fundamental concern for successful integration. Therefore, we will compare PoW and BFT blockchains according to their scalability.

For blockchain-based systems, scalability should be considered under two dimensions, namely, scalability in terms of transaction throughput and scalability in terms of number of peers that are participating in the consensus process. First scalability dimension is an important parameter showing the performance of the blockchain system, whereas latter is an important parameter in assessing decentralization degree of the blockchain system.

Unluckily those scalability dimensions contradict each other when it comes to PoW and BFT blockchains. Particularly, BFT blockchains can maintain a relative high throughput. For example, BFT Tendermint consensus protocol [26] is able to process thousands of transactions per second. However, BFT blockchains can scale to few dozens of block generators. Whereas PoW blockchains are able to maintain a relative low throughput while scaling to thousands of nodes in achieving consensus. Therefore, we are combining PoW and BFT blockchains as hybrid blockchain architecture to design a scalable blockchain system.

2.2.4 Blockchain platforms

There are many blockchain platforms with different capabilities and properties that are in use today. We briefly discuss those relevant to this thesis.

Blockchain technology has been first introduced with *Bitcoin*, that is essentially an abstract protocol [92] to perform peer to peer trustless electronic cash payments. Bitcoin uses a Proof of Work protocol (described in Section 2.3) to achieve consensus. Essentially, Bitcoin is a public blockchain, however permissioned Bitcoin blockchains can be generated for testing and development purposes. Bitcoin protocol employs a scripting language, that is non-Turing complete, which consists of byte opcodes that are specifying operation types. Where, transactions can only be generated using the operations in the opcode set. There are many libraries in various programming languages (e.g., C, C++, Java, Javascript) that allow to generate applications for Bitcoin blockchain. However, operations that can be performed with libraries are limited to the opcode set of the Bitcoin.

Ethereum is a second generation blockchain protocol that fundamentally presents a generalized and decentralized state transition machine [136]. Similar to Bitcoin, initial version of the Ethereum blockchain also uses Proof of Work protocol to achieve consensus, however Proof of Stake protocol is under development for future releases.[6] Ethereum blockchain is a public blockchain, but the protocol also allows to generate permissioned Ethereum blockchains with different consensus paradigms, such as Proof of Authority. Ethereum achieves generalization by employing the smart contract concept with a Turing complete programming language called *Solidity*[7]. With this, blockchain technology gains much broader applicability to many different domains such as IoT, automative, media, marketing, etc.

Another popular blockchain platform is *Hyperledger Fabric*, which is one of the Hyperledger[8] projects hosted by the Linux Foundation. Fabric supports smart contracts like Ethereum. Additionally Fabric allows smart contracts to be written in any general purpose language (e.g. Java, Golang and Node) to run distributed applications without dependency on a native cryptocurrency. Unlike Bitcoin and Ethereum, Fabric is a framework for deploying permissioned blockchains. Fabric uses a special kind of Byzantine Fault Tolerant protocol to achieve consensus, where blockchain transactions are first executed, then ordered and verified.

[6] github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ
[7] solidity.readthedocs.io
[8] hyperledger.org

Chapter 3

Literature Review

In this chapter, we review related work that are dealing with data privacy and security issues of IoT systems, and also work that apply blockchain technology to IoT. To this end, first, in Section 3.1 we explore work dealing with privacy issues in the IoT domain. Afterwards, in Section 3.2 we survey work performing botnet detection. Finally, in Section 3.3, we review previous work that apply blockchain to IoT and also work that target to improve scalability of blockchain protocols.

3.1 Privacy for IoT

In recent years, privacy in the IoT domain have been deeply investigated, with the results that various approaches have been proposed for dealing with different aspects of privacy. In this section, we provide an overview of those proposals that are more related to this thesis. In particular, we focus on those approaches that enforce, in some ways, users' privacy. However, we also have to note that literature offers several interesting proposals that, like our framework, deal with the problem of decentralized policy enforcement. All these efforts have been done in domains different from IoT but they deserve to be cited. In the following, we summarized work in these two directions.

3.1.1 Enforcement of users' privacy in the IoT domain

We start noticing that relevant efforts have been done with the aim of granting/denying accesses to data sensed through IoT devices. In general, these grants are regulated according to a predefined set of rules (e.g., access control policies). So far different access control models have been exploited in the IoT domain: role based access control (RBAC) (e.g., [82], [16]); capability based access control (CapBAC) (e.g., [121]); attribute based access control (ABAC) (e.g., [118], [142]), and access control models based on semantic rules (e.g., [33]).

Recently, blockchain has been gaining interest as overlay framework to perform access control [46, 48, 100, 99, 102]. For instance, [100] and [99] introduces a blockchain based access control framework called *FairAccess* to meet security and privacy needs of IoT. In FairAccess,

Reviewed approaches	Decentralized architecture	Privacy preference support	Tailored for IoT	Control on future use of data	Check compliance with consumers policies
[82], [16], [121], [118], [142], [33]	✓/✗	✗	✓	✗	✓
[46, 48, 100, 99, 102]	✓/✗	✗	✓	✗	✓
[127]	✗	✓	✓	✗	✓
[123]	✗	✗	✓	✗	✗
[23]	✗	✓	✓	✗	✓
[120]	✗	✗	✓	✗	✗
[65]	✗	✗	✓	✗	✗
[49]	✗	✓	✓	✗	✗
[85]	✗	✗	✓	✗	✗
[119]	✓	✓	✓	✓	✗
[145], [61]	✓	✗	✗	✗	✗
[25], [20]	✓	✗	✗	✗	✓
Approach in Chapter 4	✗	✓	✓	✓	✓
Approach in Chapter 5	✓	✓	✓	✓	✓

Table 3.1: Features of the considered state-of-art approaches. If the approach contains the feature: "✓".

blockchain is used to keep track and validity of *access transactions* among autonomous organizations in a distributed manner. To this end, FairAccess introduces new blockchain transaction types to grant, get, delegate, and revoke access rights to resources. Similarly, in [48], blockchain based architecture is used to manage data privacy and security of smart vehicles. Whereas, [46] proposes a blockchain architecture to protect security and privacy of data generated by IoT devices on smart home example. Also in [102] blockchain is used as a central enabler for access authorizations in IoT, which supports variety of access control models.

Although these proposals are instrumental to control how users' personal data are used, and thus, in some sense, to protect users' privacy, they do not make users able to provide their own preferences on how their data have to be used and distributed. As such, users' privacy depends on how access control rules are specified. Thus, they do not give users ability to have a full control on how data have to be processed (e.g., accessed, aggregated, released).

User's privacy preferences have been considered in [127], which proposes a framework avoiding inference of personal data due to data fusion. Users specify their privacy preferences in terms of a level of confidentiality associated with each data. The proposed framework consists of a central unit, called Personal Data Manager (PDM), that manages personal data collected by different devices, playing thus the role of a gateway between users and third party applications. A further module, called Adaptive Interface Discovery Service (AID-S), computes the risk of inference associated with a data disclosure, via probabilistic models and learning algorithms (e.g., RST, KNN, Bayes Filter, HMM etc.). Based on this risk value, AID-S recommends optimal privacy settings to users to reduce the privacy risks. Similar to our proposal, also this approach considers user's perspective, but only in stating the confidentiality level of personal data. Moreover, this approach does not enforce privacy of the user against data inferences in a decentralized setting and thus it is not able to pose more limitations on possible data fusions.

Compliance of user's privacy preferences with third party's privacy policies have been considered in [23]. Here, it has been proposed an application for mobile phones that supports customers in making privacy decisions. Privacy preferences are automatically generated according to the result of a questionnaire filled by users. Proposed mechanism acts as a privacy compliance mediator between the corporate and the user, with use of privacy policies of the corporate which are attached to the RFID tag of the product, and privacy preferences of the user stored in her mobile phone. The application informs the user whether his/her privacy preferences complies with the corporate's privacy policies. This is a limited approach, as it is only valid for an application scenario, and only valid for known queries.

Similar to our approach, other proposals have targeted smart environments (e.g., smart home and smart city systems) with the aim of protecting users' privacy. In [120], authors address the security and privacy problems of IoT smart home at the network level, that is, by monitoring network activities of IoT devices to detect suspicious behaviors. In [65], a privacy and security model for smart homes is presented, where risk analysis is performed on smart automation system, and, based on this they identified central concepts of their privacy and security model. An external entity, called Security Management Provider (SMP) has been proposed. SMP can add access control rules to protect specific IoT devices or can apply dynamic policies to change access control rules depending on the context (e.g., the family

members being present or absent from the house). Similarly in [49], a context-aware adaptive approach for devices that access location-based services is proposed. Privacy is enforced by an agent, which derives location information through context analysis on network-based services and reacts to context variations. Moreover, software defined networking (SDN) technology is used to block/quarantine IoT devices in the smart home network, based on their network activities. This proposals aims at protecting privacy of the user by limiting access on data through an external entity, i.e., SMP, with the use of context information. However, those approaches have limitations, as they do not allow users to define privacy preferences on how their data might be used in an decentralized scenario.

In [85], a two layered architecture is proposed for protecting users' privacy in smart city applications. A trusted layer is designed to store real identities of individuals that can be processed only by the platform's components, without disclosing the identities to the outside world. In contrast, an untrusted second layer only makes generic, unidentifiable and identity-independent information available to external applications. Even if this proposal protects personal data, this is enforced only inside the trusted layer, without considering future operations that may be done on the released data to infer new sensitive information. Moreover, users are not able to set and enforce their own privacy preferences.

3.1.2 Decentralized policy enforcement

A notable example of decentralized privacy management is represented by the sticky policy approach [101]. According to this approach user privacy preferences are strictly associated (sticky) with users' data. [101] describes the core mechanisms required for managing sticky policies, along with Public Key Infrastructure (PKI) and encryption methodologies to attach sticky policies with data as well as to enforce them. [24] presents a distributed enforcement approach for sticky policies that permits data to be disseminated across heterogeneous hardware and software environments without pre-existing trust relationships. Also, [119] presents a sticky policy approach to manage the access to IoT resources by allowing users to set and manage access control policies on their own data. In this approach, sticky policies allow to define: owner of the data; purposes for which the data can be used; a timestamp that points out the validity; and constraints which represent the rules for filtering the data with obligations and restrictions. Even though sticky policy approach has the goal of decentralized enforcement of user privacy preferences, it is only limited to traditional privacy preference model with retention, purpose etc. Moreover, sticky policy approaches use encryption mechanisms to enhance privacy, which add extra level of complexity and demand higher resources from the devices.

However, we have to note that cryptographic solutions have been often used to enforce distributed access control in several domains, like online social networks (e.g., [30, 66]), or cloud infrastructure (e.g., [133, 109]). In the following we focus on efforts done in wireless sensor networks, as these are more relevant for the IoT domain. For instance, [145] presents a distributed privacy preserving access control scheme designed for network of sensor nodes owned by different users, connected via a network, and managed via an offline certificate authority. In the proposed scheme, access control is regulated exploiting tokens that users have to pre-buy from the network owner before entering the sensor network. Users can query sensor data

with unspent tokens and sensor nodes are subject to validate the token and grant appropriate amount of requested data. The scheme makes use of blind signatures in token generation, which ensures that tokens are publicly verifiable yet they are unlikable to user identities. Similarly to the previous cited work, [61] presents a distributed protocol for achieving privacy-preserving access control for sensor networks by exploiting a ring signature scheme. To query sensor nodes, a user needs to build a ring signature along with the query command and send them together to sensor nodes. However such approaches do not consider privacy preferences of the data owner.

Literature offers also works dealing with composition of heterogeneous access control policies. These have been done with the goal of composing a set of access control policies into a single one [25, 20]. For instance, [25] proposes a semantic framework for policy composition without committing to a specific access control model. Access control policies are modelled as four valued predicates. Similarly, [20] proposes an algebra for constructing security policy from different policies.

Table 3.1 presents a summary of the main features of the approaches reviewed in this section and their comparison with our privacy enforcement approaches presented in Chapters 4 and 5. As presented in the table, all above-mentioned proposals do not completely address challenges arisen in IoT ecosystems. It is fundamental to give users more control on data management within IoT platforms and control future uses of data to achieve proper privacy protection.

3.2 Botnet detection systems

In this section, we review the related work for P2P botnet detection, as we propose AutoBotCatcher, a P2P blockchain based botnet detection framework, in Chapter 6. In recent years, vast amounts of work has been devoted to P2P botnet detection. In general, botnet detection methodologies can be categorized into two groups: host-based and network-based approaches. Host-based approaches require the monitoring of all hosts, which is impractical for the IoT domain. Therefore, we focus on network-based approaches, which in turn can be classified into two groups, discussed in the following. Table 3.2 presents a summary of the main features of the approaches reviewed in this section and their comparison with AutoBotCatcher presented in Chapters 6.

Network traffic signature based approaches

Literature offers many work that classify hosts based on their network traffic behaviour. In general, these approaches exploit supervised/unsupervised machine learning techniques to identify whether hosts are benign or malicious [56, 143, 67, 110, 104, 144, 93, 88, 59]. In machine learning based approaches, an algorithm is trained with the samples of network traffic, in order to detect malicious network traffic. For example, in [144], a statistical network traffic fingerprinting approach that is using unsupervised machine learning techniques to identify P2P activity and to group hosts participate in malicious P2P traffic is proposed. Entelecheia, proposed in [59], aims to detect in bots in their waiting stage by exploiting their social behaviour. Specifically, Entelecheia uses network traffic signatures to create a graph of likely malicious flows and

perform graph mining steps to to cluster and label botnet nodes. In [110], Saad *et al.* uses five different machine learning algorithm to extract features from the observed benign and botnet network traffic, and use that features for botnet detection. PeerSark, proposed in [93], is a port and protocol oblivious conversation-based approach to detect P2P botnet traffic. However, in a dynamic network, botmasters can randomize botnet traffic by changing communication frequency, packet sizes, etc. As such, network traffic signatures learned by machine learning approaches may not be robust enough to identify bots [139], which would eventually make such approaches ineffective.

Moreover, some of the proposed approaches, such as [56, 88], rely on deep packet inspection techniques (DPI) to analyze network packet contents. Particularly, BotMiner proposed in [56], groups hosts that share similar communication patterns in performing malicious activities. More recently, BotDetector proposed in [88], leverages on DPI techniques to monitor information recorded on HTTP headers is proposed. However, these checks can be bypassed through encryption of C&C channels. Moreover, DPI based approaches are computationally expensive. Given that, we conclude that network traffic signature based approaches are not suitable for dynamic and evolving IoT environments.

Group and community behavior based approaches

Some works use group and community behaviour analysis for botnet detection [39, 139, 91, 149]. As an example, similarly to us, [39] exploits mutual contacts extracted from network traffic flow of hosts, in order to identify bots in a P2P network. [39] executes *dye-pumping algorithm*, where iteratively pumps dye to the nodes in the mutual contacts graph through a coefficient called *dye-attraction* to send more dye to the nodes that are more likely to be bots, and, algorithm picks the nodes with more dye than a threshold. Whereas [139] collects network flow records at the edge routers of a campus network, and performs group level behaviour analysis on network traffic flow with Support Vector Machine (SVM) to label P2P bot clusters. Botgrep [91] is a network graph structure based botnet detection method that uses data mining techniques for bot detection. The graph is structured according to the information on node pair communications as communication graph. Botgrep partitions the communication graph into smaller pieces as localized communication graphs. However, these approaches are able to detect only previously known bot types. Therefore, they are not suitable for IoT, where new botnets emerge frequently [75]. Differently, PeerHunter [149] exploits Louvain method to perform network flow level community behaviour analysis on mutual contacts graph, without relying on previously known bot types. Yet, PeerHunter performs static botnet detection on the collected network traffic flow data, which is inadequate for a dynamic and evolving IoT environment that requires dynamic botnet detection.

Blockchain-based botnet detection approach

Blockchain technology might be a solution to the problems faced by relevant works proposed in the literature. In fact, to enable multiple parties to collaborate for botnet detection, by designing AutoBotCatcher (Chapter 6) we chose to use blockchain rather than a centralized

Reviewed approaches	Decentralized architecture	Dynamic bot detection	Working on encrypted payload	Tailored for IoT	Multi-party collaboration	Blockchain based
[56]	✗	✗	✗	✗	✗	✗
[143]	✗	✗	✓	✗	✗	✗
[67]	✗	✗	✓	✗	✗	✗
[110]	✗	✗	✗	✗	✗	✗
[104]	✗	✓	✓	✗	✗	✗
[144]	✗	✗	✓	✗	✗	✗
[93]	✗	✓	✓	✗	✗	✗
[88]	✓	✓	✓	✓	✗	✗
[59]	✗	✓	✓	✗	✗	✗
[39]	✗	✓	✓	✗	✗	✗
[139]	✗	✓	✓	✗	✗	✗
[91]	✗	✗	✓	✗	✗	✗
[149]	✗	✗	✓	✗	✗	✗
AutoBotCatcher	✓	✓	✓	✓	✓	✓

Table 3.2: Features of the considered state-of-art approaches. If the approach contains the feature: "✓".

system given the benefits blockchain might bring. Thanks to its distributed consensus protocol [1], blockchain platform does not require a central trusted party to validate the correct execution of the collaborative process (aka botnet detection), and ensure transparency on collected snapshots of communities of IoT devices overcoming the possible lack of trust among parties involved in the botnet detection (see Section 2.2.2). Moreover, as a state transition machine, blockchain lets us model the whole botnet detection process as a set of shared application states (aka states of parties collaborating in the botnet detection). This allows AutoBotCatcher to perform dynamic and collaborative botnet detection on large number of IoT devices.

3.3 Decentralizing IoT with Blockchain

Blockchain is a promising technology to achieve decentralization, as it allows to achieve distributed consensus among different parties without needing to trust each other or a central server/database. In the literature, there are few instances of application of blockchain to IoT. Let us provide an overview of these proposals, as we propose Hybrid-IoT, a hybrid blockchain architecture for IoT, in Chapter 7. However, we also have to note that literature offers some interesting proposals that deal with the problem of scalability of PoW blockchains. All these efforts have been done in domains different from IoT, but they deserve to be cited.In the following, we summarized works in these two directions.

Blockchain for IoT

In the literature, there are few instances of application of blockchain to IoT. One application of blockchain to IoT is [15], where a blockchain platform for industrial IoT (BPIIoT) has been proposed. BPIIoT exploits smart contracts to develop a decentralized manufacturing applications of cloud based manufacturing (CBM). In that, IoT devices run blockchain services and contain blockchain wallets for sending transactions to the smart contracts. Smart contracts have also been exploited in [64] in order to manage smart meter data. Likewise, in [78] a blockchain based system has been proposed to manage firmware updates of IoT devices. [60] exploits blockchain to store access control data, as a data storage system in a multi-tier IoT architecture. In [5], blockchain and smart contracts are used to secure authorization requests to IoT resources. The above mentioned works make use of blockchain to either execute smart contracts or perform application specific tasks, but not to decentralize IoT systems and achieve autonomous application execution.

[47] proposes a blockchain architecture for IoT containing two layers, namely: smart home layer (centrally managed private ledgers) and a overlay layer (public blockchain). Resource constrained devices form private ledgers in smart home layer, that are centrally managed by constituent nodes. Group of constituent nodes select a cluster head operating in the overlay network. It relies on distributed trust algorithms to eliminate computational overhead from IoT devices due to PoW solving task. However, the proposed architecture does not help with

[1] With the assumption that more than two-thirds of the block generators are honest.

the decentralization of IoT. In fact, IoT devices are centrally managed and connected to one constituent node that does not take part in the distributed consensus.

Differently from previous works, the Tangle[2] protocol implements a global distributed ledger for IoT by using Directed Acyclic Graph generated by transactions as a blockchainless approach. Tangle is designed as a cryptocurrency for IoT to make micro-payments possible, it does not provide an architecture or data structure to decentralize IoT and it is not Turing complete to allow scripting and smart contracts.

Blockchain scalability

In the literature, limitations related to scalability of PoW blockchains have been widely studied [43, 40] and different methodologies have been proposed to overcome such limitations [79, 122, 50, 13]. One notable approach is using block Decentralized Acyclic Graphs as reconstruction of classical longest chain protocols [79, 122]. Another significant proposal is Bitcoin-NG protocol [50] by Eyal *et al.* Bitcoin-NG protocol proposes use of two types of blocks in the Bitcoin blockchain, namely; key blocks and microblocks, to achieve higher transaction throughput. Differently, [13] Back *et al.* proposes the use of multiple interoperable blockchains, referred to as pegged sidechains, to allow assets in different ledgers to be transferred between each other.

In [13], Back *et al.* proposed to use multiple interoperable blockchains, referred to as pegged sidechains, to allow assets in different ledgers to be transferred between each other. This approach uses multiple interoperable blockchains, referred to as sidechains.Even though this approach aims to scalability performance improvement, the main purpose of the pegged sidechain approach is to allow users to transfer their assets in different blockchains.

Likewise, in [45], a blockchain architecture that is containing private ledgers managed centrally, a public overlay blockchain, and cloud infrastructure is proposed. Resource constrained devices form private ledgers, that are centrally managed by constituent nodes. Group of constituent nodes select a cluster head that is operating in the overlay network. Transaction validation is done via distributed trust. The proposed architecture does not help with the decentralization of IoT. In fact, IoT devices are centrally managed and connected to one constituent node that does not take part in the distributed consensus.

[2] `iota.org/IOTA_Whitepaper.pdf`

Chapter 4

Enhancing User Privacy in IoT

Internet of Things (IoT) applications aim at improving our every-day lives in a variety of forms. However, the acceptance of IoT services is hindered by customers perceived risks, and due to the high volume of collected personal data, as well as by trust in organizations providing the services. In fact, as discussed in the Chapter 1, privacy is considered as a major concern [86] in IoT.

In general, with some differences, privacy laws of many countries such as European Union's GDPR [106] impose that any stakeholder collecting users' personal data has to make available the adopted privacy policies, commonly provided in the form of a legal statement that declares how data are collected, used and disclosed to third parties. In order to gain a service, then, the interested users have to check privacy policies in place at the service provider site and accept them (or eventually partially accept them). However, it is not easy for the average user of IoT applications to make a conscious decision since it is very difficult to have a clear understanding of what the service provider privacy policies mean in terms of personal data disclosure.

As presented in Chapter 3.1, recent research efforts tried to address privacy issues (e.g., [123], [23], [96], [65], [49]). User privacy has also been the subject of intensive research in domains complementary to IoT. For instance, recent research efforts have focused on the web domain (e.g., [17]), BigData analytics systems (e.g., [37]), NoSQL datastores (e.g., [36]), data stream management systems (e.g., [29]), and RFID technology (e.g., [54]). However, these proposals do not completely address the new challenge posed by IoT ecosystems. In fact, due to the complexity of the data flows among IoT devices and back-end systems, users easily lose the control on how their data are distributed and processed. This is even made worse by the lack of an effective control on how data generated by different IoT devices are combined to infer new information on individuals. In contrast, we believe that it is fundamental to give users more control on data management within IoT platforms.

In order to cope with this challenge, we present a core framework which aims at enhancing user control on personal data usage within IoT platforms. The proposed privacy framework consists of a novel model supporting the specification of privacy preferences regulating data analysis within IoT environments, and related enforcement mechanisms. The problem addressed by the proposed framework is preventing inference of sensitive and confidential in-

formation about the user that resulted from the aggregation of data generated by various IoT devices of user. Particularly, we believe that it is important to make users able to express their own privacy preferences on how their data have to be managed, generally expressed in terms of well-established concepts in the context of privacy management, like: purpose of the usage of personal data, time of retention of personal data in the stakeholder system, disclosure to third parties, etc. The framework has been designed to operate in a reference IoT platform with a general architecture that easily fits numerous IoT systems. The key contribution of this work is twofold. On the one hand it supports novel privacy preferences tailored for the IoT domain, which constrain: 1) which portion of users personal data can be accessed and how these data can be combined, and 2) what cannot be derived from users data by analytics processes. On the other hand, in order to enhance user control, it supports the automatic definition of new privacy preferences for regulating the processing of new data generated by analytics processes. We would like to note that, we are aware that an average owner of an IoT device may not be able to set his/her privacy preferences due to lack of technical capabilities and knowledge. Indeed, as future work we plan to improve usability of the framework by designing tools to help them set their privacy preferences.

The remainder of this chapter is organized as follows: Section 4.1 discusses key requirements for the privacy framework definition; Section 4.2 briefly presents the reference IoT platform which can host the framework; Section 4.3 introduces the privacy preference model; Section 4.4 discusses the automatic generation of new privacy preferences; Section 4.5 presents the proposed enforcement mechanisms; and Section 4.6 assesses the enforcement overhead.

4.1 Requirements

Let us now consider two key requirements related to the definition of the envisaged privacy framework.

The first requirement, consists in making users able to state *which pieces of their data, possibly generated by different devices, can be jointly accessed, composed or aggregated* for given purposes. As an example, referring to the domain of fitness IoT applications, a user may want to avoid that personal data, like weight and height, could be jointly accessed with fitness data, like movements, as this could lead to infer information on the physical state.

However, these types of restrictions are not enough. In fact, as an average user could not figure out the potential risks of combining data of different devices as he/she could not be aware of what new information can be inferred, thus failing in setting these constraints. As an example, a user might not be aware that, by combing data related to movements, heart beats and breath rate, possible psychological disorders due to insomnia can be inferred. Therefore, we believe that individuals should *explicitly state what cannot be inferred from their data about them, by any analytics process.* For instance, referring to the previous example, a user may want to forbid that his/her movements are used to infer psychological diseases.

A second challenging requirement is related to the management of new derived data. Indeed, although the above-mentioned features help one to regulate the derivation of new data, they fail in controlling how these new data will be managed. Indeed, these new data could flow

through other analytics processes with purposes different from the one authorized for the process originating them. Moreover, these processes could in turn infer additional new knowledge about individuals, on which the user loses control. As such, to make the user able to have some control also on new data, we see the need of the *automatic definition of privacy preferences for new derived data*. These new preferences have to be defined taking into account the privacy preferences of owners of each single piece of data used for computing the new derived data.

Example 1 *Let us consider a scenario with a smart scale, a step counter that also calculates the walked distance, and a fitness watch with a heart rate monitor and movements sensors, and let us assume that privacy preferences are specified for the data generated by all these devices. The walked distance and the number of steps reveal the average step length, which in turn, with some approximation allows inferring the individual height. Starting from height and weight one can infer the body mass index of the individual, which is new inferred information not directly sensed by any IoT device. Since no privacy preference has been specified for these new derived data, a process that combines the body mass index with heart beats and movements data could infer information related to the individual physical state. In order to avoid undesired inferences, the processing of the body mass index, needs to be regulated by privacy preferences derived from those specified for the data generated by the smart scale and the step counter.*

4.2 The reference IoT platform

The main purpose of the framework is making users able to set and enforce their privacy preferences in a centralized IoT scenario. In the following, we present the subjects of this scenario and reference IoT platform that we consider.

Subjects. We consider two types of subjects, namely, data owners and data consumers. Data owner is the user that owns the IoT device generating data. We assume that data owners define individual privacy preferences over their data in order to control how their data are distributed and processed. In contrast, data consumers consume raw or processed data from IoT systems. We assume that data consumers adopt privacy practices that specify how the consumer will use the users' data.

The reference IoT platform. Once the IoT privacy language/model is defined, we have the further challenge to envisage how the defined mechanism for enforcing privacy preferences against stakeholder privacy policies can fit in the IoT ecosystem. As this purpose, we target a platform whose general architecture represent a variety of IoT systems. The platform, shown in Figure 4.1, consists of a few components supporting communication, analysis and enforcement. This platform represents a centralized scenario, where data processing and privacy enforcement performed by a central entity. In the reference IoT platform that message broker, Complex event process (CEP) system and the privacy preference enforcement monitor are deployed in the cloud. This platform represents a centralized scenario, where data processing and privacy enforcement performed by a central entity.

Communication features are mainly in charge of a message broker, namely a component

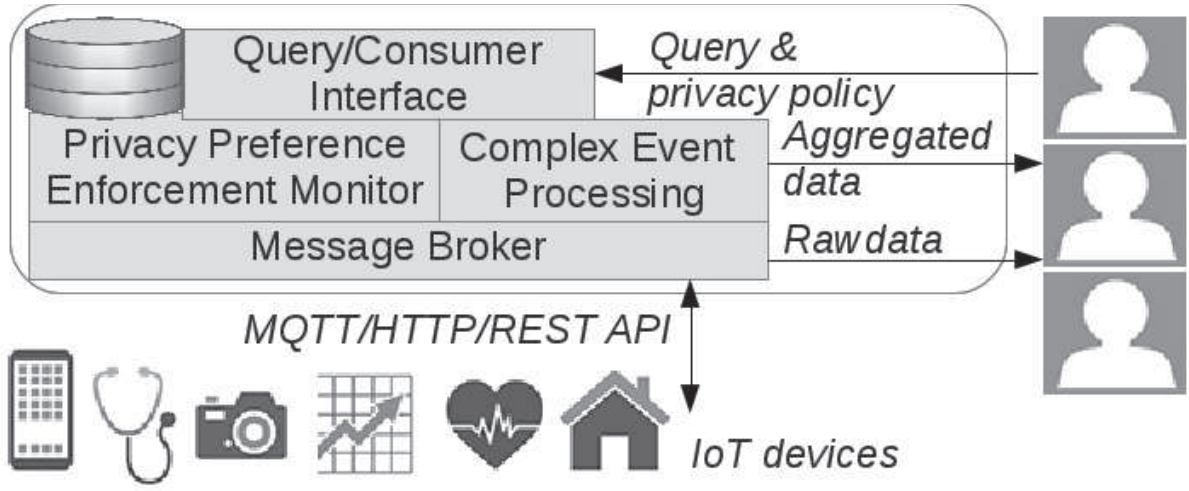

Figure 4.1: Reference IoT platform

acting as a message passing server, through which, by means of different protocols (e.g., MQTT[1] etc.), IoT devices publish their data, and consumers subscribe to access available data. In the reference IoT platform we assume that the broker publishes only data sensed from IoT devices, each of which refers to a single user.

The analysis features are supported by analytics tools. The reference platform targets a CEP system, as CEPs provide analysis toolkit tailored for the IoT domain (e.g., see [34]). Sensed data are forwarded by the message broker to the CEP as append-only streams of tuples, where registered queries analyze, combine and aggregate them generating new output data.[2] We assume that all data tuples in a stream have the same scheme, and we also assume that each stream is associated with a single device.

Finally, the enforcement monitor (see Figure 4.1) implements the enforcement mechanisms of the proposed privacy framework. This module has to analyze every consumer request and decide if his/her privacy policy satisfies the privacy preferences specified for the accessed data on the basis of the proposed model. These requests can be a simple demand for accessing raw data generated by a device, as well as complex queries to be addressed to the CEP.

4.3 The privacy preference model

In this section, we introduce the core elements required for the specification and enforcement of privacy preferences.

[1] `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/`

[2] CEP queries are an example of possible implementation of the above mentioned analytics processes.

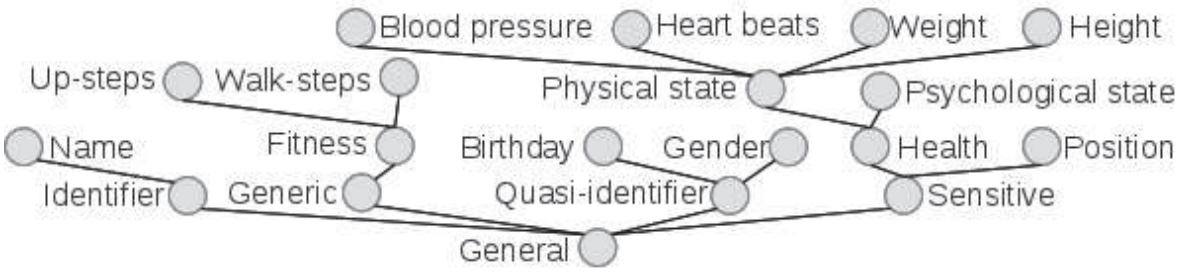

Figure 4.2: Example of data category tree

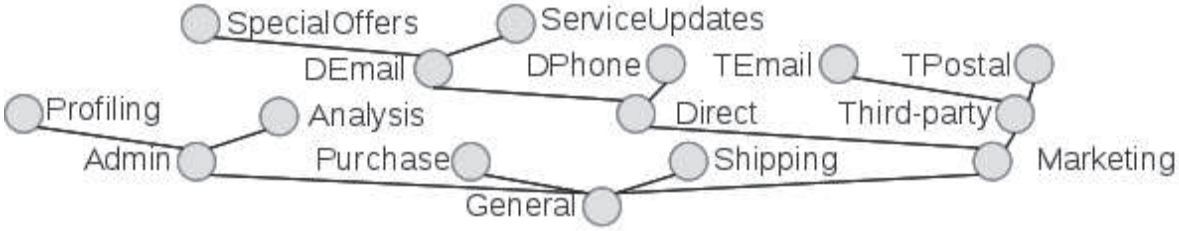

Figure 4.3: Example of purpose tree

4.3.1 Data categories and purposes

A first key feature of our privacy preferences is the ability to regulate data analysis on the basis of data categories, specifying the type of information conveyed by data streams. Categories are hierarchically organized into a tree structure, where data belonging to a category *dc* implicitly belong to all categories that have been directly or indirectly specialized by *dc*. In what follows, we denote with C the set of adopted data categories, whereas Ct_C denotes the data category tree of C. A data category is therefore modelled as a pair $\langle$ *id, pc* $\rangle$, where *id* is the category identifier, whereas *pc* is the identifier of the category in Ct_C which is specialized by *dc*, if any, or it is set to $\perp$, if the category is the root of Ct_C.

Categories can range from general ones, possibly denoting heterogeneous data (e.g., personal data) to fine grained ones, modelling data represented by single attributes of a data stream (e.g., heart beats). The proposed model does not constrain the categories to be used and system administrators are free to classify data on the basis of application scenarios requirements. For instance, in [35] we have proposed the data categories *identifier*, *quasi identifier*, *sensitive* and *generic*, which have been derived from the analysis of privacy legislation and the literature on privacy aware data publishing. The category *sensitive* refers to data, such as political preferences, religious creed, employment status, and health conditions, which reveal sensitive information of individuals private life. The category *identifier* refers to data that reveal the identity of data subjects, whereas the category *quasi identifier* refers to a set of data which linked to external data allow identifying the individual to whom these data are referring to. Finally, the category *generic* groups all data that do not belong to the previously considered

categories. We believe that the above mentioned categories are general enough to be used in a variety of IoT scenarios, as they can be easily specialized or generalized to meet scenario dependent requirements. For instance, Figure 4.2 shows how such categories can be specialized for the domain of sport and fitness IoT applications.

The second element of our model are the purposes which are used to specify the reasons for which data generated by user devices can be accessed and analyzed. The collection of all the purposes defined for an application scenario forms the scenario purpose set, denoted in what follows as P. Like data categories also purposes are hierarchically organized into a tree, referred to as the purpose tree Pt_P. A purpose p is modelled as a pair $\langle pid,\ pa \rangle$, where pid is the identifier of p, whereas pa is set to the identifier of the direct ancestor of p into the purpose tree, or $\perp$, if p does not descend from any other purpose. Figure 4.3 shows an example of the purpose tree.

System administrators define the purpose set and data category set related to a given application scenario, as well as the related trees, and classify the data flowing throughout the platform assigning a data category to each stream attribute.

4.3.2 Privacy preferences

Privacy preferences are specified for each user device, and regulate the access and processing of the data stream generated by such a device. They are fine-grained in the sense that they regulate the access at the level of each single attribute. In particular, they first specify, through purposes, the reasons for which data streams can be analyzed, and those for which the access is explicitly prohibited. This second case is particularly useful for specifying exceptions when purposes are organized into a tree. We model this by a privacy preference component, denoted as *intended purposes*, formally defined as follows.

Definition 1 (Intended purpose) *An intended purpose ip is a pair $\langle Aip,\ Exc \rangle$, where Aip (allowed intended purposes) is a set of purposes belonging to P and Exc (exceptions) is a set of purpose elements that descend from elements in Aip. ip authorizes the access for all purposes that descend from*[3] *the elements in Aip except for those that descend from any element of Exc.*

Example 2 *Let us suppose to specify an intended purpose ip wrt the purpose tree in Figure 4.3, which authorizes the access for the purposes that descend from marketing and analysis, except for those that descend from direct. According to Def. 1, ip is defined as $\langle \{marketing,\ analysis\},\ \{direct\} \rangle$.*

Besides intended purposes, a privacy preference specified for a target attribute a of a data stream allows one to specify the data categories that can be jointly accessed with those associated with a for a given purpose. This is formalized through the definition of a *joint access constraint.*

[3]Hereafter we assume this relationship as reflexive, meaning that every element descend from itself.

Definition 2 (Joint access constraint) *Let a be an attribute of a data stream and let c be the data category associated a. A joint access constraint jac for a is a tuple ⟨Adc, Exc, ip⟩, where Adc (allowed data categories) is a set of data categories belonging to C, Exc is a set of data categories that specialize elements in Adc, and ip is an intended purpose. jac specifies that the categories that descend from those specified in Adc can be jointly accessed with c for the purposes authorized by ip, except for those that descend from any element in Exc.*

It is worth noting that the *ip* component of *jac* specializes the homonymous component of the privacy preferences in that, in order to grant the joint access, the authorized purposes implied by the specification of *jac.ip* must not be disjoint by those implied by *pp.ip*.[4]

Example 3 *Let us consider the data categories in Figure 4.2, and let us consider an attribute a classified as Physical state. Suppose to define the following joint access constraint for a: jac=⟨{sensitive, identifier, fitness}, {health}, ⟨{third-party}, ∅⟩⟩. jac authorizes the joint access of a with data classified as sensitive, identifier or fitness, except for data classified as health for any purpose descending from third-party.*

A privacy preference protects target data combining together intended purposes and joint access constraints. Additionally, it allows specifying a set of categories that should not be derived from the accessed data, independently from the type of access that is performed (e.g., joint or simple). For instance, from the analysis of movements, heart-beats and blood pressure one could infer that the individual to whom these data refer to could suffer from any circulatory disease. As such, users may want to avoid that apparently generic data, such as their movements, could be used to infer their physical state.

Definition 3 (Privacy preference) *A privacy preference pp is a tuple ⟨a, ip, jac, cdc⟩, where a is an attribute of a data stream, ip specifies the intended purposes for which a can be collected and used, jac specifies a joint access constraint, whereas cdc is a set of data categories belonging to Dc, which denotes the set of categories which cannot be derived from a.*[5]

Example 4 *Let us now suppose to specify a privacy preference pp that protects the attribute heart-beats to which the category Physical state has been assigned. pp is specified in such a way to integrate the intended purpose ip and joint access constraint jac introduced in Ex. 2 and 3, and to forbid the derivation of health related data, that is, pp=⟨heart-beats, ⟨{marketing, analysis}, {direct}⟩,⟨{sensitive, identifier}, {health}, ⟨{third-party}, ∅⟩⟩,{health}⟩. Thus, pp grants the access to heart-beats for the purposes Marketing, Analysis, Third-party, TPostal and TEmail, the joint access with other data provided that jac is satisfied, and, finally, forbids using heart-beats in any inference process that derives health data.*

The considered privacy preference model mainly focuses on the new privacy components tailored for the IoT domain (i.e., consumer identities, joint access constraints, and category

[4]The consistency of the specification of *ip* with *jac.ip* is verified at privacy preference specification time.
[5]The categories are those in *cdc* and their descendants in the category tree.

derivation constraints). However, it is relevant to note that the model can be easily extended to support also preferences on additional components used in common privacy practices, like: data retention, aka how long data can be stored in the information system of consumers, third-party data sharing options. More precisely, these additional preferences could be verified before data release by matching these preferences against corresponding statements in the consumer privacy policy. Moreover, it is important to highlight that the additional elements that we have introduced to cope with the new issues that IoT scenario poses, (i.e., consumer identities, joint access constraints, and category derivation constraints) are not mandatory. This provides flexibility to vendors to decide whether to adhere only to common privacy practices or to exploit the extended privacy preferences so as to provide users with a more complete control on their personal data.

4.4 Privacy Preferences for new derived data

In this section we propose an approach to automatically generate new privacy preferences for new generated data. In order to present the approach we first overview the query model and its operators so as to highlight which kind of new data the privacy preferences have to deal with.

4.4.1 Query model

A variety of query languages have been proposed for CEPs, but no de-facto standard language has emerged so far. However, most of the existing languages share the same set of core operators. As such, we consider a CEP with queries modelled as a loop-free directed graph, where nodes are SQL-like operators that are incrementally performed on streams, and edges indicates the flow that tuples follow through the graph. According to this query model, each single operator (node) modifies the stream entering the operator generating a new *internal stream*, which will be further modified by other operators in the graph till being composed into the final *output stream*. Hereafter we consider the following operators:

- IN, which conveys a data stream, containing tuples modelling data sensed by owner device, to other query operators;
- OUT, which returns the final data stream resulting from executing all query operators in the graph on the input streams;
- σ, which selects only those tuples of the stream entering the operator that satisfy given selection criteria;
- π, which generates a stream resulting from the projection and possible combination of attributes of the entering stream;
- $\bowtie$, which performs the join of two data streams on the basis of an equality condition of a pair of attributes;

- Σ, which executes given aggregate functions over windows of data items of the stream entering the operator.

Operators σ, *IN*, and *OUT* do not generate any new data, as they do not change the schema of the stream entering the operator. In contrast, operators π and Σ might create new data. Indeed, standard SQL π allows specifying new attributes as combination of attributes of the entering stream. For instance, a derived attribute *steps* can be specified as *walking-steps + up-steps*, where *walking-steps*, *up-steps* are attributes belonging to the stream received as input by π. Similar for Σ, which allows specifying new attributes resulting from the aggregation of multiple instances of a target attribute. For instance, Σ could derive the average heart beats as *avg(heart-beats)*.

A separate discussion is deserved for the $\bowtie$ operator, which executes the equi-join of two streams. $\bowtie$ returns a stream whose schema is defined as the union of the attributes of the two streams received as input, except for the attributes referred to within the equality condition, of which only one copy is included. For instance, let us suppose to execute the join of the stream S_1 with attributes *name*, *height*, and *weight*, with the stream S_2, with attributes *user*, *heart-beats*, *walk-steps* specifying *user=name* as join condition. The derived stream is composed of the attributes *height*, *weight*, *steps*, *heart-beats* and a copy of *name / user*. Although $\bowtie$ does not create any new data, it merges into a unique attribute (i.e., the one over which the join is computed) two attributes, which might be regulated by different privacy preferences.

Example 5 *As an example, let us suppose to execute the equi-join of the data streams generated by the fitness watches owned by Bob and Mary on attribute position which is included in both the streams. Bob and Mary specify different privacy preferences for their data as well as for position. More precisely, let us suppose that Bob specifies a privacy preference that grants the access to position for the intended purpose* ip_B[6]$=\langle\{Analysis\},\emptyset\rangle$*, whereas Mary for the intended purpose* $ip_M=\langle\{Admin\},\emptyset\rangle$*. In addition, let us suppose that Bob specifies the jac constraint* $jac_B=\langle\{Generic, Sensitive\},\{Identifier, Quasi\text{-}identifier\},\langle\{Marketing\},\emptyset\rangle\rangle$*, whereas Mary* $jac_M=\langle\{General\},\{Identifier\},\langle\{Direct\},\emptyset\rangle\rangle$*. Finally, let us suppose that neither Bob nor Mary specify any cdc constraint, thus,* $cdc_B = cdc_M = \emptyset$*. The attribute position of the output stream resulting from the execution of* $\bowtie$ *should specify a privacy preference resulting from the composition of preferences specified by Mary and Bob.*

4.4.2 Composed privacy preferences

As above-pointed out, with the exception of σ, *IN*, and *OUT*, all other operators return a stream that might contain new attributes defined on the basis of a selection of attributes, denoted as *At*, belonging to the stream(s) entering the operator. In defining privacy preferences for these new attributes we decided to consider the privacy preferences defined for each attribute in *At*, so that every preference specified by the user to which this new data might refer to are considered. As such, the new privacy preference for a derived attribute *da* is defined by *composition of the*

[6]Hereafter we denotes the components of Bob and Mary privacy preferences using the subscript B and M, respectively.

privacy preferences of any attribute a in At. For instance, for the above mentioned example related to the operator π, $At = \{walking\text{-}steps, up\text{-}steps\}$. The privacy preference of *steps* is thus derived by the composition of the preferences specified for *walking-steps* and *up-steps*.

In defining this composition, we have adopted a conservative approach, that is, we define the new privacy preference in such way that each derived component (i.e., *ip*, *jac*, *cdc*) is satisfied iff all corresponding components in the preferences associated with any attribute a in At are also satisfied. The following formalizes this composition.

Definition 4 (Composed privacy preferences) *Let P and C be the purpose and data category set of the considered application scenario, and let Pt_P and Ct_C be the related purpose and category tree, respectively. Let da be an attribute belonging to the output stream generated by an operator op, and let At be the set of attributes from which da is derived. The privacy preference pp of da is derived as follows:*

- *Intended purpose component ip is defined as:*
  - *Aip (allowed intended purposes) is the intersection of the purposes implied by the ip's allowed intended purposes of all attributes in At, that is: $ip.Aip = \bigcap_{a \in At} \bigcup_{p \in a.pp.ip.Aip} p^{\downarrow}$, where $p^{\downarrow}$ denotes a set composed of p and all purposes descending from p in Pt_P.*
  - *Exc (exceptions for the intended purposes) is the union of the ip's exceptions of the privacy preferences specified for At attributes, that is: $ip.Exc = \bigcup_{a \in At} a.pp.ip.Exc$*
- *Joint access constraint jac is defined as:*
  - *- Adc (allowed data categories) is the intersection of the jac's allowed data categories of all attributes in At, that is: $jac.Adc = \bigcap_{a \in At} \bigcup_{c \in a.pp.jac.Adc} c^{\downarrow}$, where $c^{\downarrow}$ denotes the set composed of c and all categories that descend from c within Ct_C.*
  - *Exc (exception) is the union of the jac's exceptions specified for all attributes in At, that is: $jac.ip.Exc = \bigcup_{a \in At} a.pp.jac.ip.Exc$;*
  - *ip (jac intended purposes) is defined following the same criteria of intended purposes, that is: $jac.ip.Aip = \bigcap_{a \in At} \bigcup_{p \in a.pp.jac.ip.Aip} p^{\downarrow}$, and $jac.ip.Exc = \bigcup_{a \in At} a.pp.jac.ip.Exc$.*
- *The category derivation constraint cdc is defined as the union of prohibited data categories specified in the privacy preferences of all attributes in At, that is: $cdc = \bigcup_{a \in At} a.pp.cdc$*

Example 6 *Let us newly consider the scenario introduced in Ex. 5, and let us consider the derivation of the privacy preferences to be assigned to the derived attribute position. On the basis of Def. 4, the ip component of the composed privacy preference is $\langle \{Analysis\}, \emptyset \rangle$, as Analysis descends from Admin within Pt_P (see Figure 4.3). Similarly, according to Pt_P and Ct_C in Fig 4.2 and 4.3, the derived jac component is $\langle \{Generic, Sensitive\}, \{Identifier, Quasi\text{-}identifier\}, \langle \{Direct\}, \emptyset \rangle \rangle$, whereas component cdc is set to $\emptyset$ as neither Bob nor Mary constrain what can be inferred from the processing of position.*

4.5 Enforcement

A key element of the framework depicted in Figure 4.1 is the module in charge of privacy preference enforcement. Given a query q with a set of streams $\{S_1,\ldots,S_n\}$ as input, the goal of the enforcement is granting the q execution only if the privacy policy specified by the consumer submitting q satisfies the privacy preferences specified by owners of all devices generating $S_1,\ldots,S_n$.[7] Query compliance relies on the verification that each single operator in q complies with the privacy preferences specified for the accessed attributes of the data stream(s) received as input. In performing this check we need to consider that, except for *IN*, all operators receive as input data stream(s) generated by other operators in the graph, which can be characterized by new defined attributes as well as by attributes originating from the input streams. The privacy preferences associated with any new derived data are specified by the Privacy-enhanced schema of the stream, an enforcement artifact that keeps track of the privacy metadata of each internal stream. In the remainder of this section we first describe the rationale of query operators compliance, and then we propose an enforcement mechanism performing the a query compliance verification.

4.5.1 Query operators compliance

Let q be a query which is submitted for execution for access purpose *ap*. Given an attribute *at*, we denote with *at.pp* the privacy preference specified for *at*.

Definition 5 (Query operator compliance) *Let op be an operator in the query graph of q, let $oS_1 \ldots oS_n$ be the internal streams entering op, and let At_{oS_i} be the attributes of oS_i accessed by op. op complies with the privacy preferences of $oS_1 \ldots oS_n$ iff for each attribute $at \in \bigcup_{i=1}^{n} At_{oS_i}$ for which a privacy preference has been specified: (i) ap complies with the intended purposes ip specified within at.pp; and (ii) op complies with the joint access constraint jac and the category derivation constraint cdc of at.pp.*

Let us now focus on the compliance of *op* with the privacy preference specified for an accessed attribute *at*. Let us start to look at purpose compliance. Let P be the purpose set of the considered application scenario, Pt_P be the related purpose tree, and let *ip* be the intended purpose component of *at.pp* specified on P. Let us denote with $p^{\downarrow}$ a set composed of p and all p descendants in Pt_P. The set of purposes implied by *ip*, denoted with $\vec{ip}$, is given by $\bigcup_{p \in ip.Aip} p^{\downarrow} \setminus \bigcup_{p' \in ip.Exc} p'^{\downarrow}$.

Definition 6 (Purpose compliance) *Let op be an operator of q, let at be an attribute of a stream accessed by op, and let pp be the privacy preference specified for at. op complies with the intended purposes ip of at.pp iff $ap \in \vec{ip}$*

Example 7 *Let us assume that a query q which implements the analysis described in Ex. 1 is submitted for access purpose TPostal (see Figure 4.3). Let us here consider the compliance of*

[7] We recall that we assume that each stream contains only data generated by a single device, owned by a unique user.

an operator π of q, which projects the attributes heart-beats, movements and userid originated by the fitness watch (see Ex. 1), with the privacy preferences pp specified for heart-beats. In particular, let us start to assess the compliance of q's access purpose TPostal with the intended purpose ip of heart-beats's privacy preference, supposing that ip matches the specification in Ex. 2. Since TPostal descends from Marketing but not from Direct, according to Def. 6, it complies with ip.

Let us now consider the satisfiability of the joint access constraint *jac* of *at.pp*. Let C be the the data category set of the considered application scenario, and let Ct_C be the related data category tree. Let us denote with $c^{\downarrow}$ ($c^{\uparrow}$) the set of all categories that directly or indirectly specialize (generalize) a data category c within Ct_C. The set of categories implied by *jac* is composed of all categories that descend from the catagories in *jac.Adc* which do not descend from any category of *jac.Exc*. More precisely, $\vec{jac} = \bigcup_{c \in jac.Adc} c^{\downarrow} \setminus \bigcup_{c' \in jac.Exc} c'^{\downarrow}$.

Let As be the set of attributes that are accessed by *op* and let *pp* be the privacy preference specified for an attribute $at \in As$. We say that *op* satisfies the joint access constraint *jac* of *pp*, iff all the data categories associated with the attributes in $As \setminus at$ belong to the set of categories implied by *jac*, and *ap* complies with the intended purpose component *ip* of *jac*. Let us denote with *at.dc* the set of data categories specified for *at*. The set of categories implied by *at.dc*, denoted with $\vec{at.dc}$, is composed of all categories c in *at.dc* and all categories that generalize c within Ct_C, i.e., $\vec{at.dc} = \bigcup_{c \in dc} c^{\uparrow}$

Definition 7 (Jac satisfiability) *Let op be an operator of q, let at be an attribute of a stream accessed by op, and let pp be the privacy preference specified for at. op satisfies the joint access constraint jac of at.pp iff* $\forall a \in (As \setminus \{at\}) \rightarrow \forall c \in a.\vec{dc} \rightarrow c \in at.pp.\vec{jac}) \wedge ap \in jac.\vec{ip}$

Example 8 *Let us consider again the scenario in Ex. 7, and let us now evaluate whether π satisfies the jac constraint of heart-beats's privacy preference, assuming that this constraint matches the one presented in Ex. 3. jac grants the joint access of heart-beats with data classified as identifier, fitness or sensitive, except for health data, for access purposes that descend from Third-party (see Figure 4.3). Let us now assume that movements has been classified as fitness, whereas userid as identifier (see Figure 4.2). The categories of the accessed attributes are implied by jac, and since the access purpose TPostal descends from Third-party, we derive that jac is satisfied.*

Let us finally consider the satisfiability of the *cdc* constraint of *at.pp*, which specifies the data categories that cannot be derived from the processing of *at*. The enforcement of these constraints require to know the data categories that could be derived when given operations are executed on *at*. As an example, as mentioned in Section 4.1, combing data related to movements, heart beats and breath rate, it is possible to infer that the referred individual could suffer from psychological disorders due to insomnia. More precisely, on the basis of the average observed data, one could estimate whether the referred individuals: 1) sleep well and thus are supposed to be in a good health status, 2) have a poor sleep quality, and thus may suffer from chronic tiredness and low concentration, 3) or they do not sleep enough and thus

may suffer from mental disorders or serious pathologies. We assume that information related to possible data inferences is provided by domain experts or data mining tools that calculates association rules among data categories. The inferences are specified by means of *derivation paths*, which model a set of data categories, which composed or aggregated by means of given operators allow deriving data potentially related to the user to whom the data protected by the privacy preferences refer to. Derivation paths represent possible inferences that can be performed within an application scenario.

Definition 8 (Derivation path) *A derivation path dp specified wrt C is a tuple ⟨ Ac, fn, op, dc ⟩, where Ac is a set of data categories of C, fn refers to a function / operation that can be executed by an operator op of type Σ / π,* [8] *and dc specifies a derived data category belonging to C.*

Example 9 *Let us suppose to define a derivation path dp specifying that the knowledge of average movements, heart beats and breath rate allows deriving health state: dp=⟨{movements, heart-beats, breath-rate}, avg, Σ, health⟩.*

Let $SoAC$ be the union of the data categories associated with the attributes accessed by q and let Dp be the set of derivation paths specified for the target application scenario. The set of data categories which on the basis of cdc cannot be derived, hereafter denoted as $\vec{cdc}$, is composed of all C categories that specialize those referred to within cdc, i.e., $\vec{cdc} = \bigcup_{c \in cdc} c^{\downarrow}$.

Definition 9 (Cdc satisfiability) *Let op be an operator of q, let at be an attribute of a stream accessed by op, and let pp be the privacy preference specified for at. op satisfies the category derivation constraint cdc of at.pp, iff it does not exist any derivation path dp within Dp such that: 1) op corresponds to the operator referred to by dp.op, 2) the function/operation specified by dp.fn corresponds to the one performed by op (e.g., +, avg, etc.) denoted as op.fn, 3) the set of accessed data categories specified with dp.Ac is included in SoAC, and 4) the implied category referred to by dc is among the categories implied by cdc, i.e., iff* $\nexists dp \in Dp | dp.op = op \wedge dp.fn = op.fn \wedge dp.Ac \subseteq SoAC \wedge dp.dc \in pp.\vec{cdc}$.

Example 10 *Let us consider again Ex.7 and 8, and let us now check whether π satisfies the cdc constraint {Health} assuming that Dp uniquely consists of the derivation path in Ex. 9. The derivation path in Ex. 9 does not match the processing activities of π as the referred operators are different, thus the constraint is satisfied.*

4.5.2 Query compliance analysis

Let us now focus on the enforcement mechanism. In order to keep track of the privacy meta-data associated with any internal stream of a query graph, which are required to evaluate the compliance of each operator, we introduce a structure denoted *privacy enhanced schema*.

[8] These are the only operators that generate new data.

Definition 10 (Privacy enhanced schema) *Let C and Ct_C be the data category set and related data category tree of the considered scenario. The privacy enhanced schema pes of an internal stream is a pair $\langle sid, As\rangle$, where sid is the stream identifier, whereas As is a set of tuples $\langle id, dc, pp\rangle$, each specifying privacy metadata related to an attribute of the considered stream. More precisely, id denotes the referred attribute, dc specifies the categories of id selected from C, whereas pp specifies the privacy preference of id.*

The privacy enhanced schemas of the internal streams can be straightforwardly derived traversing the query graph and applying the composition criteria presented in Def. 4 for each operator. However, due to space limitations, we do not present here the derivation mechanism.

Query compliance is verified when all the query operators comply with the privacy preferences specified for the accessed data. For presentation purposes let us introduce a basic notation to represent query operators, thus modeling an operator *op* as a tuple $\langle tp, Pa, inS_1, inS_2, outS, pOp_1, pOp_2\rangle$, where: *tp* models the type of operation performed by *op*, *Pa* models *op*'s input parameters, inS_1, inS_2 and *outS* model the privacy enhanced schemas of the streams received as input and generated by *op*, respectively, and pOp_1, pOp_2 model the operators that generate the streams processed by *op*. Finally, let us denote with *op.cmp* a component *cmp* of *op*.

Compliance analysis is orchestrated by function *compliesWith*, whose pseudocode is shown in Algorithm 1, which traverse the query graph in post order, analyzing operator by operator. The analysis of a query q starts invoking *compliesWith* on operators OUT of q for q's access purpose *ap*.

Algorithm 1: *compliesWith*(op, ap)

```
1  if op.tp==IN then
2  |  Return true;
3  rOp1= compliesWith(op.pOp1, ap);
4  if op.pOp2 ≠⊥ then
5  |  rOp2= compliesWith(op.pOp2, ap);
6  |  cmp=checkOp(op.pOp1.outS, op.pOp2.outS, op, ap);
7  |  Return cmp∧rOp1∧rOp2;
8  else if Op = ⋈ then
9  |  cmp=checkOp(op.pOp1.outS, ⊥, op, ap);
10 |  Return cmp∧rOp1;
```

The compliance of an operator *op* is evaluated by function *checkOp*, shown in Algorithm 2, which is invoked on the privacy enhanced schema of the streams received as input by *op*

whenever a node is visited (see line 6 and 9 of Algorithm 1).

Algorithm 2: *checkOp*(peS_1, peS_2, *op*, *ap*)

```
1 refAs=∅;
2 if op.tp=OUT then
3 |   refAs=peS1.As;
4 else
5 |   refAs=⋃_{rAt∈op.Pa} ⋃_{at∈rAt.As} (σ_{(id=at)}op.inS1.As)∪(σ_{(id=at)}op.inS2.As);
6 |   Return checkIp(refAs,'ip',ap) ∧ checkJacCdc(refAs,op,ap);
```

checkOp, on the basis of the operation type performed by *op*, the input parameters *Pa* of *op* and the privacy enhanced schema of the streams entering *op*, derives the set *refAs* of attributes accessed by *op* and the respective privacy metadata. Then, it invokes the functions *checkIp* and *checkJacCdc* which verify whether *op* complies with the privacy preferences of the attributes in *refAs*. The compliance of an operator *op* is verified by separately checking the components of the privacy preferences specified for the accessed attributes (collected with *refAs* in Algorithm 2), which are derived on the basis of the operator type. More precisely, *checkOp* evaluates the components *ip*, *jac* and *cdc* of the privacy preferences of all the attributes referred to within the input parameters *Pa* of operators of type σ, Σ, $\bowtie$ and π, and the whole set of attributes characterizing the privacy enhanced schema of the input stream of an operator *OUT*. Indeed, any attribute referred to within *Pa* is accessed for deriving the corresponding operator output stream, and any attribute in the input stream of *OUT* is projected to the output stream. The compliance checks for the privacy preferences of the derived attributes are handled by functions *checkIp* and *checkJacCdc*.More precisely, *checkIp* verifies the compliance of the access purpose *ap* of *q* with the intended purposes specified for the attributes in *refAs*, thus verifying if *ap* is among the purposes implied by each *ip* component, implementing the check in Def. 6. The pseudocode of *checkIp* is presented in Algorithm 3. Similarly, function *checkJacCdc*, defined on the basis of Def. 7 and 9, verifies whether the *jac* and *cdc* constraints associated with the

attributes in *refAs* are satisfied. The pseudocode of *checkJacCdc* is presented in Algorithm 4.

Algorithm 3: *checkIp*$(As, iptp, ap)$

```
1  ipCmp=true;
2  for each at∈As do
3  |  switch ctp do
4  |  |  case ip do
5  |  |  |  ipCmp=ipCmp∧(ap∈at.pp.$\vec{ip}$);
6  |  |  |  break ;
7  |  |  case jac do
8  |  |  |  ipCmp=ipCmp∧(ap∈at.pp.jac.$\vec{ip}$);
9  |  |  |  break;
10 |  Return ipCmp;
```

Algorithm 4: *checkJacCdc*$(refAs, op, ap)$

```
1  jacCmp=true;
2  cdcCmp=true;
3  SoAC=∅;
4  for at∈refAs do
5  |  oAt=refAs\at;
6  |  SoAC=SoAC∪at.dc↑;
7  |  for a∈oAt do
8  |  |  jacCmp=jacCmp∧(∀c∈a.dc↑→c∈at.pp.$j\vec{ac}$);
9  jacCmp=jacCmp∧checkIp(refAs,'jac',ap);
10 SoDP=$\sigma_{(op=op.fn \wedge Ac \subseteq SoAC)}$Dp;
11 for at∈refAs do
12 |  for dp∈SoDP do
13 |  |  cdcCmp=cdcCmp∧(dp.dc⊈at.pp.$c\vec{dc}$);
14 Return jacCmp∧cdcCmp;
```

4.6 Performance Analysis

In this section we experimentally evaluate the performance of the proposed privacy preferences enforcement mechanism, within a scenario where a monitor implementing them is connected with Streambase 7.6,[9] a known commercial CEP system [41]. In particular, given a Streambase query Q, serialized as an XML document, we measure the time required by our monitor to verify whether the privacy practice of the consumer submitting the query satisfies all the privacy preferences associated with the streams entering Q.

Our experiments have been run on an Intel Core-i5 PC with 6 GB RAM, and the monitor prototype has been implemented with Java SE Development Kit 8u73. The experiments have

[9] http://www.streambase.com/

been carried out with synthetic data. More precisely, data streams have been generated as composed of 3 attributes, each of which associated with a different privacy preference. Privacy preferences have been defined in such a way that their components refer random elements of the Data Category and Purpose sets, which, in turn, have been populated with synthetic data. In this scenario, we ran two main experiments, which, under different settings, show how the performances scale on varying the query and domain complexity.

4.6.1 Experiment 1 - Time overhead varying query complexity

Acknowledging that query complexity can affect the performance of the enforcement mechanism, our first experiment measures the time overhead for a benchmark of 10 queries with different number and type of operators. We note that complexity of queries varies based on the number and type of involved operators. The considered queries are characterized in the left part of Table 4.1.

For this experiment, we prefixed the number of elements in the Data Category Set (C), the Purpose Set (P) and the Derivation Path Set (Dp) as : 100 categories, 30 purposes and 10 derivation paths. More precisely, aligned with the order of magnitude in [27], we populated P with 30 purposes, whereas in order to have enough data categories to cover a variety of domain, we populated C with 100 elements, finally, we specified 10 derivation paths as we consider this as an acceptable amount of possible data inferences. We believe that this case could represent common use scenarios. Purpose and Data Category trees have been randomly generated on the basis of the respective sets, limiting the maximum number of children that each node can have as 3.

Figure 4.4 shows the time required by the monitor to analyze each query. The overall time is given by the duration of three enforcement phases: 1) the Query Parsing, which handles the parsing of the XML document encoding the query to be verified; 2) the PES Derivation, which derives the privacy-enhanced schema of all internal streams; and lastly, 3) the Compliance Analysis, which evaluates query compliance.

The execution times of all phases increase linearly with the growth of queries complexity. It is worth noting that the Query Parsing phase, which only implements preliminary configuration activities, takes a longer time than the 2 phases implementing the enforcement mechanism. Overall the execution times is always very short even in the worst case (less than 0.1 sec). This shows that our approach does not give too much overhead to the system even with complex queries.

4.6.2 Experiment 2 - Time overhead varying domain complexity

In this experiment we evaluate the impact of Data Category and Purpose Set on the time overhead, by varying the cardinality of these sets in a benchmark of 10 scenarios.

The right part of Table 4.1 characterize the benchmark wrt the Purpose Set (P) and Data Category Set (C) cardinality. The size of the Derivation Path Set has been fixed to 10.[10]

[10]We fixed Derivation Path Set cardinality, as we have observed that variance of this parameter does not significantly impact the execution time

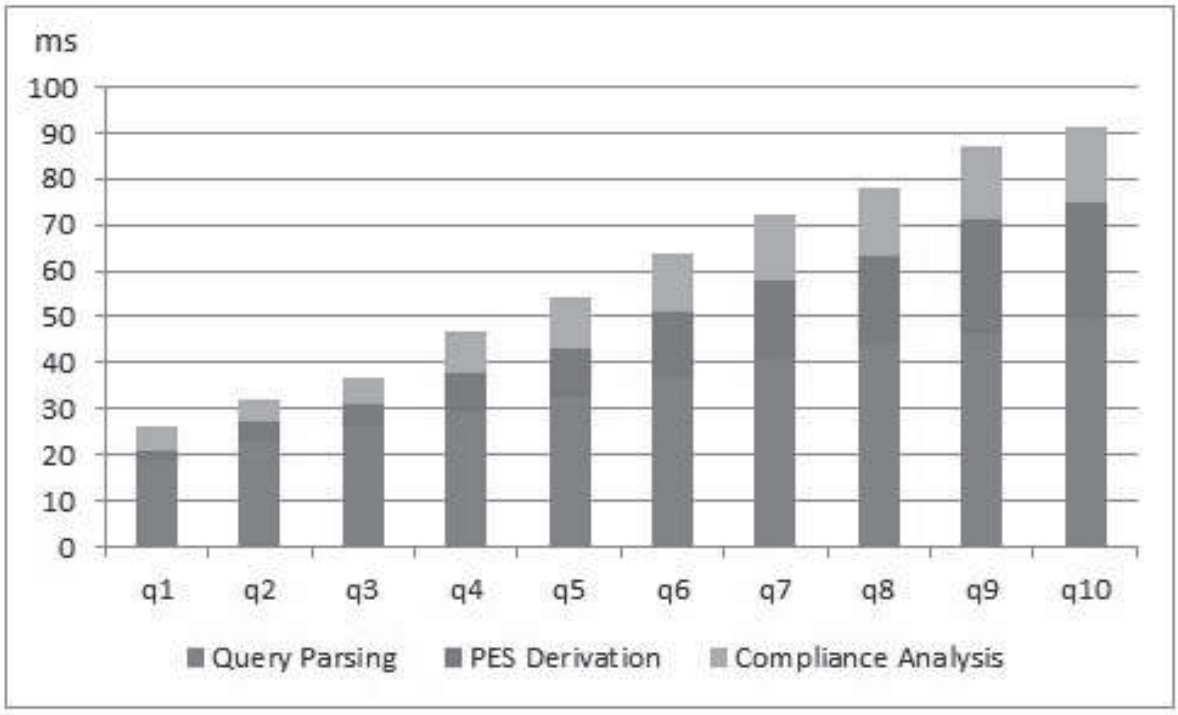

Figure 4.4: Experiment 1 Results

Figure 4.5 shows the time required by the proposed monitor to evaluate the compliance of query q5 (see Table 4.1) in each scenario. The execution time of PES Derivation and Compliance Analysis rises with the increase of Data Category and Purpose Sets cardinality. This behavior is imputable to the complexity of search operations on Data Category and Purpose Trees. In contrast, query parsing time keeps constant since the same query is used for all scenarios.

As in previous experiment, the overhead is low even in the worst case (less than 0.1 sec). Overall, even considering that enforcement checks are performed before the query is registered into the CEP, we conclude that the proposed mechanism enforces privacy with a negligible overhead.

Table 4.1: Experiments configuration

Experiment 1		Experiment 2		
Query	Operations	Scenario	$\|P\|$	$\|C\|$
q_1	1 π, 1 σ, 1 $\bowtie$, 1 Σ	s_1	100	100
q_2	2 π, 2 σ, 2 $\bowtie$, 2 Σ	s_2	200	200
q_3	3 π, 3 σ, 3 $\bowtie$, 3 Σ	s_3	300	300
q_4	4 π, 4 σ, 5 $\bowtie$, 4 Σ	s_4	400	400
q_5	5 π, 5 σ, 5 $\bowtie$, 5 Σ	s_5	500	500
q_6	6 π, 6 σ, 6 $\bowtie$, 6 Σ	s_6	600	600
q_7	7 π, 7 σ, 8 $\bowtie$, 7 Σ	s_7	700	700
q_8	8 π, 8 σ, 8 $\bowtie$, 8 Σ	s_8	800	800
q_9	9 π, 9 σ, 9 $\bowtie$, 9 Σ	s_9	900	900
q_{10}	10 π, 10 σ, 10 $\bowtie$, 10 Σ	s_{10}	1000	1000

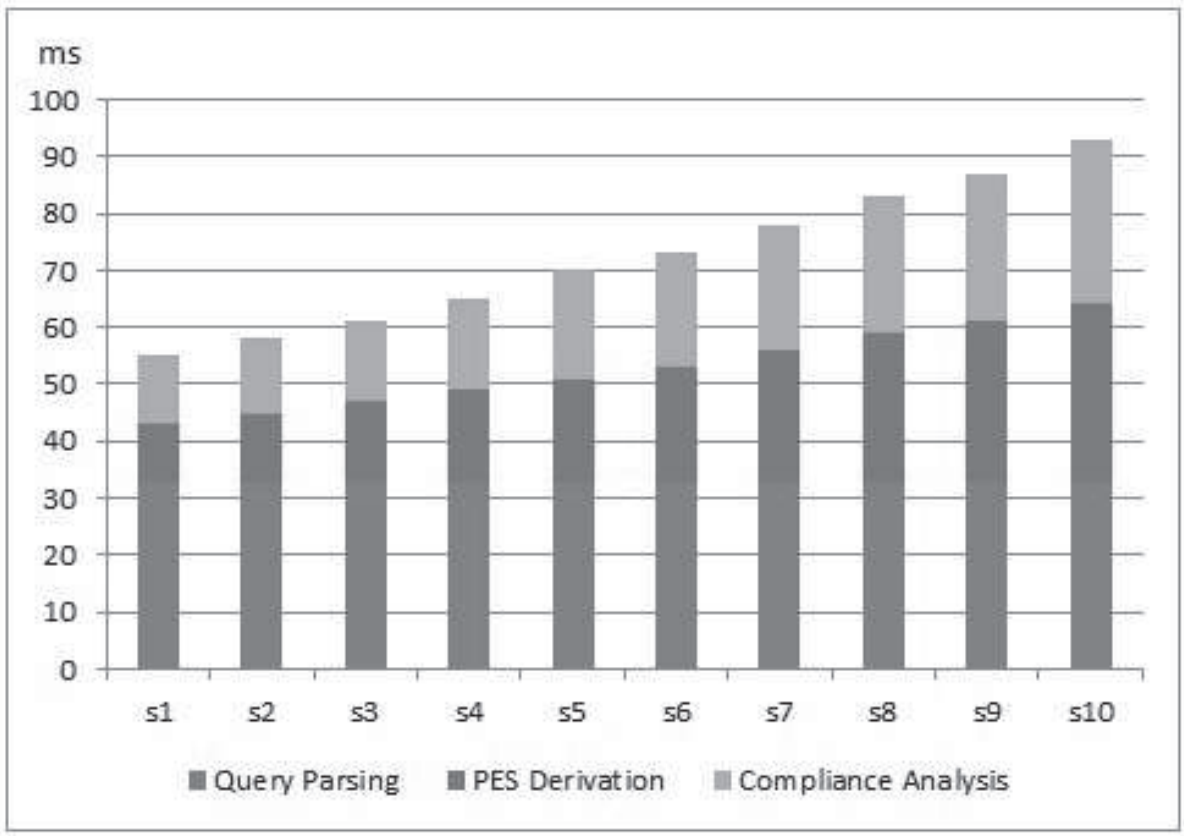

Figure 4.5: Experiment 2 Results

Chapter 5

Decentralizing Privacy Enforcement for IoT

Rise of smart objects enact transition from centralized cloud-based IoT systems to decentralized IoT systems of cooperating smart objects. Decentralization, if not properly governed, might imply loss of control over the data, with consequences on individual privacy, as discussed in Chapter 1. In this chapter, we focus on the challenging issue of designing a decentralized privacy enforcement mechanism, where compliance check of user individual privacy preferences is performed directly by smart objects, rather than by a central entity. Restrictions on devices' capabilities let us discard existing proposals for decentralized access control (e.g., [121, 118, 82, 61]), as these heavily rely on cryptographic primitives.

Previously, in Chapter 4, we addressed the problem of specifying and enforcing privacy preferences in the IoT scenario, but for a centralized architecture, that is, a scenario where devices have only the capability to sense data and send them to a data center for being analyzed. Whereas decentralized privacy enforcement scenario requires to address new important research challenges. Particularly, in decentralized setting smart objects have to perform privacy enforcement operations as they are part of data processing scheme. However, considering various types of data and data processing operations and heterogeneous processing capabilities of smart objects it is a challenging endeavour. To address these challenges, in this chapter, we extend the privacy preference model proposed in Chapter 4, by designing a set of privacy meta-data that are used by smart objects for locally checking privacy preferences and for locally enforcing user privacy preferences at smart object level. Smart objects are thus able to derive privacy meta-data for newly created data items, keep track of the operations performed over data items, denoted as *history*, in order to ease the privacy preference enforcement, and, finally, check compliance of the privacy policy of the data consumer with the privacy preferences of data items. To the best of our knowledge this is the first work proposing a decentralized enforcement of privacy preferences able to work locally at smart object level.

Acknowledging that embedding the enforcement mechanism into smart objects might imply some overhead, we have extensively tested the proposed framework. In doing the experiments, we have considered several scenarios, by varying the complexity of the privacy preferences,

smart object networks, and evaluated queries. The experiments allow us to asses the feasibility of the proposed approach in a variety of application domains.

The remainder of this chapter is organized as follows. Section 5.1 describes the system model and design assumptions of the proposed privacy preserving framework. Section 5.2 presents the enhanced privacy model and proposed enforcement mechanism. Experimental results are illustrated in Section 5.3.

5.1 System model and assumptions

The privacy enforcement mechanism relies on some privacy meta-data encoding user privacy preferences, data categories, and data history generated by smart objects.Similar to the core framework presented in Chapter 4, we assume that data generated by smart objects are modelled as data streams, and smart objects are able to perform SQL-like operations on streaming data. In the following, we present the system model and our design assumptions.

5.1.1 System model

We consider a decentralized IoT system consisting of data owners, consumers and smart objects.

We assume data owners specify privacy preferences over their data to control how their data are distributed and processed and data consumers adopt privacy practices to specify how they will use the users' data, as defined in Chapter 4.

Smart objects are extremely heterogeneous regarding their capabilities and roles in the IoT ecosystem and this also impacts the privacy checks they can perform. Since the proposed privacy preserving mechanism aims at working at the smart object level, we have first to detail which kind of IoT devices we are targeting. In particular, we need to classify smart objects based on their capabilities so as to assign to each of them a possible *role*, and corresponding function in the proposed decentralized privacy preference enforcement mechanism. In the IoT domain, device capabilities are extremely heterogeneous. On the basis of several standardization recommendations from different organizations [131, 105], we propose a smart objects taxonomy based on the object *smartness*, aka abilities in sensing and processing individual data, as follows:

– *First Level of Smartness:* at this level, devices have a very limited capacity, being able only to act as basic sensors, for sensing data from the environment.

– *Second Level of Smartness:* at the second level, devices have a limited capacity and computing power, making them only able to perform basic operations by their own, like projection and selection, which do not require window-based processes that are working on tuples organized as sliding windows, such as join or aggregation. These devices might be virtual or physically manufactured, e.g., a field-programmable gate array (FPGA) or hard-coded devices.

– *Third Level of Smartness:* at this level, we group those devices that have enough capacity and computing power for performing complex operations by their own, including window-based operations.

According to this taxonomy, a *smart object (SO)* may play three different roles in the proposed decentralized privacy enforcement:[1]

– *Sensing Smart Objects:* the main objective of this type of smart objects is sensing data from the environment. Typically, a smart object with a first level of smartness will play this role.

– *Processor Smart Objects:* this type of smart objects aim at creating new information from sensed data by performing operations over them. Complexity of these operations depends on the smartness level of the object. As such, a smart object with second or third level of smartness will be able to play this role.

– *Consumer Smart Objects:* a smart object playing this role is in charge of verifying user privacy preferences before sending data to consumers. This process is called privacy preference compliance (see Section 5.2.3 for more details). Such role can be taken by SOs with third level of smartness (e.g., gateway devices, such as Dell Edge Gateways or HPE Edgeline Gateways).

Example 11 *Throughout the chapter, we will use as an example a smart home system. We assume that the smart home is equipped with a smart heating and electricity usage control systems. The smart heating system is built on top of a network of smart objects able to sense temperature values from several rooms, adjust temperature, if needed, and share data with the heating company. The smart electricity usage system relies on a network of smart objects able to sense electricity usage data, locally compute usage trends, and send them to the electricity distribution company (see Figure 5.1). In this scenario, sensing smart objects are those devices used for measuring electricity usage and sensing room temperature, whereas processor smart objects are those used to locally aggregate data sensed by sensing SOs, with the purpose of generating trends of electricity usage as well as average rooms temperature. Finally, smart objects sending collected information to electricity and heating system companies can be categorized as consumer smart objects, as their main role is to send data to consumers.*

5.1.2 Security assumptions

Security vulnerabilities, if not properly addressed and prevented, may allow adversaries to damage correct execution of IoT systems and the proposed privacy preserving mechanism. In designing the privacy preserving mechanism, we have assumed that smart objects conduct a set of defence strategies to protect the IoT system from security threats and attacks. Let us first define the adversary model. We assume that an adversary is capable to block and intercept communication, and is also able to overhear, inject and delete messages exchanged

[1] We postpone the reader to Section 5.2 for details about processes and algorithms implemented for each role.

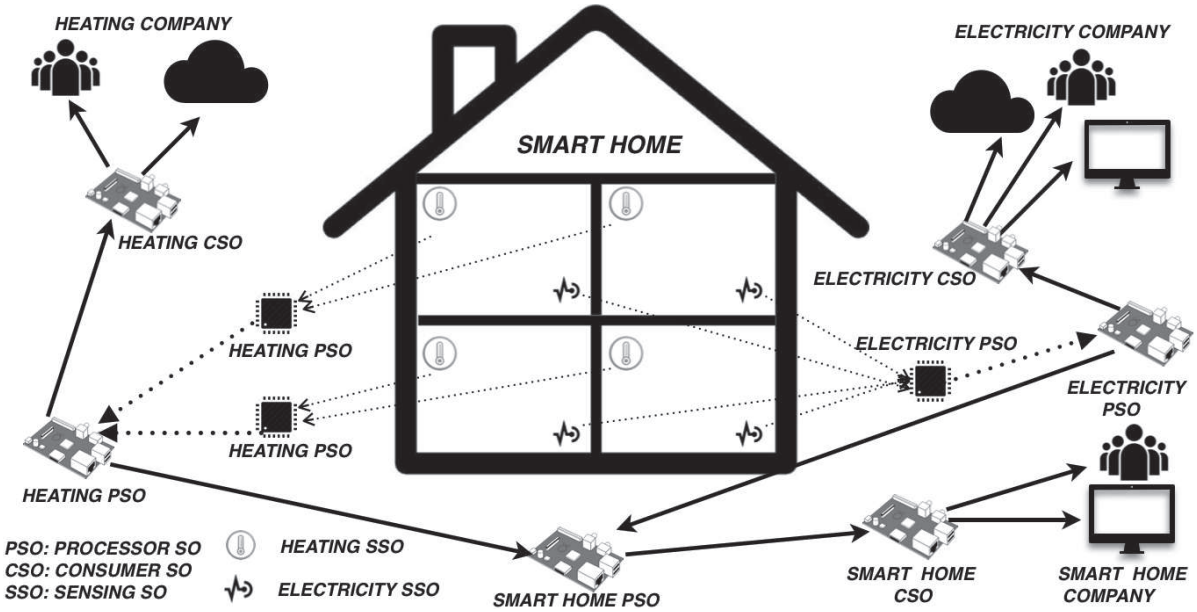

Figure 5.1: Smart home scenario

between smart objects, as well as analyze communication patterns. Moreover, we assume that sensing and processor SOs are susceptible to be compromised by adversaries and thus can act maliciously, whereas consumer SOs are honest, i.e., they behave as expected according to the proposed algorithms. This assumption is based on the conjecture that IoT gateway devices, such as Dell Edge Gateways[2], or HPE Edgeline Gateways[3] hosting consumer SOs are securely produced by their manufacturers and have high resources. Indeed, as stated by [10], a vulnerability in the hardware of such devices will inevitably threat the whole system. Therefore, we assume that device manufactures will do their best in designing secure gateway devices. Given the adversary model, the privacy preserving mechanism relies on the following defense techniques that are assumed to be conducted by smart objects:

Network monitoring and intrusion detection systems. We assume that consumer SOs are complemented with network monitoring (e.g., [111]) and intrusion-detection mechanisms (e.g., [81]). Because such mechanisms help to detect and reduce effects of *Denial of Service (DoS)* and *node compromise* attacks. Indeed, as our approach relies on machine to machine communication among smart objects in which the operations of one smart object, e.g., meta-data generation and update, mostly depend on the data sent by another smart object, DoS attacks may affect the correct execution of the protocols. Furthermore, compromised and malicious processor SOs can alter the derivation of new privacy meta-data with the result of not correctly enforcing privacy requirements of the data owners. By virtue of equipping consumer SOs with intrusion detection mechanisms, we assume that consumer SOs analyze the network

[2] $dell.com/us/business/p/edge-gateway$
[3] $hpe.com/emea_europe/en/servers/edgeline-iot-systems$

and detect compromised and infected devices acting maliciously or violating policies.

Lightweight encryption and authentication mechanisms. We also assume that all smart objects are equipped with encryption mechanisms (such as [146, 135]) that make use of lightweight cryptographic primitives (e.g., hash, nonce, etc), and thus able to encrypt the data and the meta-data before sending them. In such a way, that data cannot be disclosed and privacy meta-data cannot be compromised during the communication.

We also assume that smart objects are equipped with hop by hop authentication protocols, such as [63]. This is a a way to contrast *impersonation* attacks, where an adversary might aim to seize the identity of a legitimate smart object, such as access credentials of the device, to act on behalf of the legitimate device (e.g., injecting fake data) that may compromising privacy preference enforcement.

Clone detection protocols. We assume that smart objects are equipped with a globally aware distributed clone detection protocol (such as [38]). Cloned smart objects may generate fake data and affect the correct execution of the privacy protection protocols, as well as they can be used to disable and compromise honest smart objects and thus damage privacy enforcement.

5.2 Decentralized privacy preference enforcement

In order to assure that user privacy preferences are taken into consideration during data usage and processing, there is the need of an enforcement mechanism whose primary goal is to verify whether the privacy practices adopted by the entity wishing to consume the data, aka the consumer, satisfies data owners' privacy preferences. Privacy preferences are specified according to the model proposed in Chapter 4, which has been designed to cope with two relevant challenges that IoT ecosystems pose on the management of individual privacy. However, as in decentralized settings data might be consumed by multiple consumers, we extend the privacy preference model in Chapter 4, so as to allow data owners to specify data consumers to which a preference applies, with a new field named *consumer*.

An interesting feature of the model in Chapter 4 is its ability to associate privacy preferences with new data generated by smart objects. These new privacy preferences are defined by combining the privacy preferences specified by each owner of smart objects sensing the data involved in the derivation process. Privacy preferences are combined by taking the most conservative approach so that the resulting privacy preference satisfies the constraints specified in all the privacy preferences associated with smart objects involved in the data fusion process. More precisely, the composed privacy preference is defined by taking the intersection of the *consumer, jac.Adc* and *ip.Aip* elements, and the union of the *jac.Exc, ip.Exc* and *cdc* fields of the privacy preferences involved in data fusion. In the following, let us give an example of privacy preference composition relevant to the decentralized privacy enforcement as a part of the running example.

Example 12 *Let us consider the smart home scenario given in Example 11 and let us suppose that a user, say Charlotte, has installed this smart home system in her apartment and that she specified a privacy preference pp for the temperature data sensed by the system, as fol-*

lows: ⟨temperature, {smart-home-company}, ⟨{admin}, ∅⟩, ⟨{generic, health},∅, ⟨{admin},∅⟩⟩, {sensitive}⟩. Let us further suppose that Charlotte also specifies a privacy preference for the electricity usage data as follows: ⟨electricityUsage, {smart-home-company, electricity-company}, ⟨{admin},∅⟩, ⟨{generic},∅, ⟨{admin}, ∅⟩⟩, ∅⟩.

Let us assume that the smart home network executes an equi-join on streams containing the electricity usage and temperature data generated by Charlotte's smart home system on id_{room} attributes, which we assume to have in both the two streams. Moreover, we assume that id_{room} has associated the same privacy preferences of temperature and electricity usage in both streams. When performing the join, there is the need to fuse together these two privacy preferences. The composed privacy preference for id_{room} in the stream containing temperature data is:

⟨id_{room}, {smart-home-company}, ⟨{admin}, ∅⟩, ⟨{generic}, ∅, ⟨{admin}, ∅⟩⟩, {sensitive}⟩.

To make compliance check automatic, we assume that consumer privacy practices are encoded into privacy policies, that is, a set of statements on how the consumer will use the users' data. Typically, a privacy policy specifies: the consumer identity (id), data attributes that consumer wishes to consume, purpose of the data usage, release of the data to third-parties and time of retention of the data in the consumer information system. In what follows, we focus on the purpose and consumer id components of a privacy policy/preference and the related checks, in that they are the most important information used by the enforcement monitor. The enforcement mechanism has to verify whether the declared purpose of data usage and id of the consumer in the consumer's privacy policy is compliant with the privacy preferences specified by the users owning the smart object devices generating the data.

We handle compliance check through an aposteriori approach, that is, we perform compliance check only when data are going to be shared with consumers. The main motivation is that leaving compliance check to consumer SOs increases the possibility for data to be processed and thus to be eventually consumed according to user privacy preferences. For instance, let us consider the equi-join operation performed in Example 12. Let us assume that Charlotte's privacy preferences for electricity usage data only allow joint access with position data (e.g., coordinates of the individual), and let us also assume that joined streams have further attributes, in addition to electricity usage and temperature data. In such case, the equi-join operation in Example 12 is not compliant with Charlotte's privacy preferences. Performing a compliance check at this point would prevent to generate the joined stream, and further data flows would be cut. In contrast, by an aposteriori compliance check approach attributes in the joined streams can be further processed in the smart object network and consumers can consume the data that would not violate the privacy preferences of the users.

In order to push privacy preference enforcement at object level, we have to first complement the processed data with *privacy meta-data* that specify, for each piece of sensed or newly created data, all information needed to enforce privacy preferences (see Figure 5.2). As an example, the sensing smart objects have to be able to associate with each piece of sensed data the corresponding privacy preferences specified by object owners. Moreover, processor SOs have to be able to locally generate additional privacy preferences for the newly created or modified data. This process requires to make processor SOs aware of how the new data have been created (e.g., which are the operations, the input data involved in the operations, as well as,

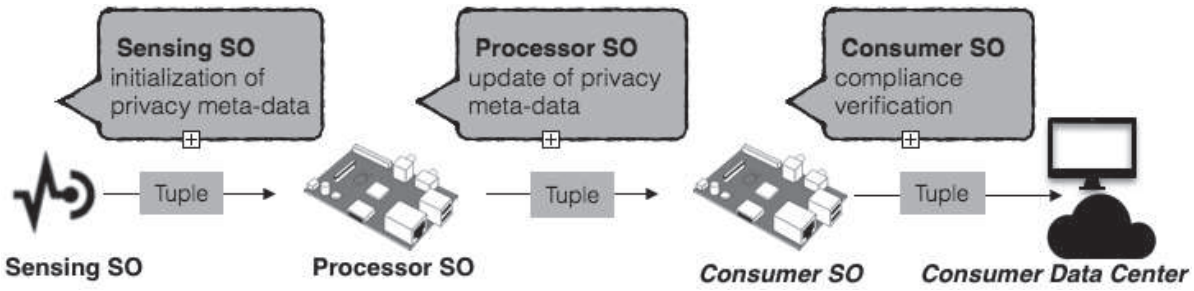

Figure 5.2: Privacy enforcement by SO roles

the corresponding privacy preferences). Finally, smart objects playing the role of consumer SOs have to be able to locally perform compliance check.

We would like to underline that complementing data with privacy meta-data allows the enforcement monitor to be executed directly on each piece of data itself. According to the proposed framework, sensed data continuously flow through smart objects, complemented with privacy meta-data. Smart objects perform some operations on data, such as aggregation, and privacy meta-data update. When data arrive to consumer smart objects, these check the compliance of the data owner privacy preferences, embedded in privacy meta-data, and the privacy policy of the consumer. Only if privacy preferences of the data owners are satisfied, data are sent to consumers' data centers (e.g. server, cloud etc.). At smart object side, the only data that has to be stored for privacy enforcement are the *derivation paths*, that is, set of data inference rules needed by processor smart objects. By virtue of that, smart objects do not need to store additional data for privacy enforcement.[4]

5.2.1 Privacy meta-data

To model additional privacy information, we need to add, for each attribute contained into the original schema, three additional attributes, namely *category*, *pp* (privacy preferences), and *history*.

Let us start discussing the *category* attribute. The category associated with an attribute is not a static information, as data associated with an attribute might be fused with other information, or simply modified, with the result of a change of its content and thus of its associated data categories. To cope with these dynamic aspects, we need to trace, for each attribute, its updated set of data categories, based on its current content. Secondly, in order to make a consumer SO able to perform privacy preference enforcement, it has to know the privacy preferences associated with each attribute. Such information is encoded in the additional *pp* attribute. Moreover, information about every operation performed on data has to be documented alongside the data itself. Indeed, by design, compliance check is only performed by consumer SOs, just before data are sent to the consumer (see Figure 5.2). Processor SOs are only in charge of executing data processing and privacy meta-data generation, without caring

[4]We underline that the total size of the derivation paths stored by processor SOs is negligible.

about compliance checks. As such, it might be possible that in a smart object network, a processor SO performs an operation, e.g., a join, that breaks a *jac* or *cdc* condition. In order to make a consumer SO able to verify if operations performed by all processor SOs satisfy *jac* and *cdc* constraints, it has to know all operations performed on each piece of data (aka each attribute). At this purpose, we introduce, for each attribute in the stream schema, a new attribute, denoted as *history*, which contains a list of history entries, one for each operation performed on the corresponding attribute.

This privacy meta-data are directly encoded into the original data stream schema, obtaining thus a new schema, called *Privacy-Enhanced Attribute Schema* (PEAS), formally defined as follows:

Definition 11 ***(Privacy-Enhanced Attribute Schema).*** *Let $S=(A_1, \ldots, A_n)$ be the schema of a data stream, where $A_1, \ldots, A_n$ are the data stream attributes. The Privacy-Enhanced Attribute Schema of S is defined as $\overline{S}=(\overline{A}_1, \ldots, \overline{A}_n)$, such that $\overline{A}_j = (A_j, pp, category, history)$, $\forall j \in 1, \ldots, n$, where:*

- *pp is the privacy preference associated with attribute A_j, as defined in Definition 3;*
- *$category$ is the set of data categories associated with the current value of attribute A_j;*
- *$history$ is a list $\{HE_1, \ldots, HE_m\}$, containing an element for each operator Op, A_j has undergone so far. $HE_i=(AC_i, RC_i)$, for $\forall i \in 1, \ldots, m$, where: – AC_i is the set of categories associated with attributes on which Op is executed for deriving $\overline{A}_j$;*
  – RC_j is the set of data categories associated with $\overline{A}_j$ after the execution of Op;

As depicted in Figure 5.2, privacy meta-data are initialized by sensing SOs, which collect the raw data and generate the corresponding PEAS schemas. In particular, at this stage in the PEAS schema the *history* field is initialized as empty, whereas the *pp* and *category* fields contain privacy preferences and associated data category information defined by smart object owners. Processor SOs also modify attributes of PEAS schema. In particular, as it will be discussed in Section 5.2.2, for each processed attribute, they have to update the *category* field, based on current value of the corresponding attribute as well as the *history* field.

Example 13 *Let us refer again to Example 12, and, in particular, to attribute att containing the temperature data of Charlotte's smart home. Recalling the privacy preferences in Example 12, and supposing that the current temperature is 24.4 Celsius degree, the initial PEAS generated by the sensing SO, is the following:*

- *att = {24.4},*
- *att.pp = ⟨temperature, {smart-home-company}, ⟨{admin}, ∅⟩, ⟨{generic, health}, ∅, ⟨{admin}, ∅⟩⟩, {sensitive}⟩,*
- *att.category = {generic},*
- *att.history = {∅}.*

5.2.2 PEAS generation

For each sensed data, the sensing SO has to initialize the proper PEAS, with the privacy preferences specified by sensing SO's owner, the initial data category of the sensed data, and an empty *history* field.[5] Additionally, after each operation, processor SOs should derive the privacy preferences of the eventually newly created attribute content, as well as update the *category* and *history* fields of each attribute involved in the operation. Hence, we assume that processor SOs are complemented with additional logic, implementing PEAS update.

Since PEAS generation in sensing SOs is quite straightforward, in that it mainly consists in initializing *pp* and *category* attributes with predefined values, in the following we focus on PEAS update algorithms. PEAS update is mainly driven by the operations performed by processor SOs. As such, we define algorithms for all query operators except for the selection operator, as, by definition, this operator does not generate any new data, as such, no modification of the PEAS is needed. The algorithm receives as input the performed SQL operator *Op* (e.g., avg, sum, $\times$, etc.) and input streams, used by the corresponding operation. In particular, we model an SQL operator based on operations it performs on the attributes. For example, the join of streams $S1(a, b, c)$ and $S2(a, d, e)$ on attribute a is modelled as two parameters: {join, (S1.a, S2.a)}, that is, the involved attributes and the performed operation. Since, an SQL operator *Op* might perform more than one operation (see, for instance, the Π operator in the following Example 14), we formally define *Op* as a set $\{\text{Pa}_1,\ldots,\text{Pa}_l\}$, where every Pa_i models a single operation. As such, $\text{Pa}_i = \{Attributes, fn\}$, $\forall\ i \in \{1,\ldots,l\}$, where *Attributes* is a set of attributes given as input to function fn when operation Pa_i is executed.

Example 14 *Suppose that the stream resulting from the equi-join operation illustrated in Example 12 also contains humidity data, thus resulting in S(temperature, electricityUsage, humidity). Let us assume to perform on this joined stream a projection* Π_1 *computing two values: the first obtained as multiplication of temperature and electricityUsage to estimate the number of people in the smart home,*[6] *and the second as execution of* $f()$ *that takes as input temperature and humidity and returns the air-quality information.* Π_1 *is thus modeled as:* $\Pi_1 = \{Pa_1, Pa_2\}$*, where* $Pa_1 = \{\{$*temperature, electricityUsage*$\}$*, multiplication*$\}$ *and* $Pa_2 = \{\{$*temperature, humidity*$\}, f()\}$*.*

PEAS generation algorithm exploits function *derivePP()*, that derives new composed privacy preferences, and function *createHistoryEntry()*, which creates a new *history* entry, based on the parameters of the performed operation. Finally, function *deriveDC()* determines the category of the new generated data given in input. To implement such a function, we assume the presence of a set of inference rules, called *derivation paths*. Each derivation path associates with a query operator and the set of categories of the data on which the operator is executed, the category of the data resulting from the operator execution. We therefore assume that derivation paths are present in every processor SO, so derived data category can be locally computed.

[5] Since we assume that sensing smart objects are in the first level of smartness, we suppose these operations are embedded by design into them with some kind of dedicated microcontroller or hard-coded application logic.

[6] For simplicity, we assume that this is possible by multiplying electricity and temperature usage.

Example 15 *Let us give an example of derivation path, by referring to the smart home scenario illustrated in Figure 5.1 and by assuming that also humidity information can be sensed. The derivation path* $dp_1 = \langle \{temperature, electricityUsage\}, \times, \Pi, sensitive \rangle$*, specifies that the knowledge of temperature and electricity usage allows, through the* Π *and* $\times$ *operators, the derivation of the number of people inside the smart home. In contrast, the derivation path* $dp_2 = \langle \{temperature, humidity\}, f(), \Pi, air\text{-}quality \rangle$ *specifies that the knowledge of temperature and humidity usage allows derivation of air quality information, through operator* Π *and function* $f()$*.*

For PEAS derivation, Algorithm 5 takes as input tuples s_1 and s_2 (if operation is not $\bowtie$ then s_2 will be empty), and parameters modeling operation Π, Σ or $\bowtie$. The algorithm returns an updated tuple, containing new attribute(s) generated by execution of the corresponding operation given in input and containing the updated *pp*, *category* and *history* fields associated with each attribute involved in the operation. In performing PEAS update for $\bowtie$ operation, attributes of tuples s_1, s_2 not used by $\bowtie$ remain unchanged, and by the nature of Σ (such as avg(), sum(), etc.), only one parameter will be present in $Op.Parameters$. The algorithm

exploits previously discussed functions: *derivePP()* and *createHistoryEntry()*.

Algorithm 5: *PEAS Derivation*(Op, s_1, s_2)

Let s_{new} be a new stream, initialized with an empty schema;
Let $refAttSet$ be the set of attributes, initialized to be empty;
if $Op = \Sigma$ *or* π **then**
  for *each* $Pa \in Op.Parameters$ **do**
    Let att_{new} be an attribute, initialized to be empty;
    Let $newData$ be the result of operation Pa;
    $refAttSet$ = Pa.Attributes;
    att_{new}.name = $Pa.name$;
    att_{new}.data = newData;
    att_{new}.category = deriveDC(Op, $refAttSet$);
    att_{new}.PP = derivePP(Op, $refAttSet$);
    att_{new}.History = $\bigcup_{att \in refAttSet} att.History \bigcup$ createHistoryEntry($\bigcup_{a \in refAttSet} a.Category$, att.category);
    s_{new}.Attributes = s_{new}.Attributes $\cup$ att_{new};
else if $Op = \bowtie$ **then**
  Let Op.Parameters=$\{a_1, a_2\}$ be the two attributes used in the Join statement;
  Let $s_1.Attributes$ and $s_2.Attributes$ be the set of attributes contained in s_1 and s_2 schema, respectively;
  Let $attributeSet$ be a set of attributes, initialized to be empty;
  $refAttSet = a_1 \cup a_2$;
  $attributeSet = s_1.Attributes \cup s_2.Attributes \setminus refAttSet$;
  $a_{1,2}$.PP = derivePP(Op, $refAttSet$)
  $a_{1,2}$.History = a_2.History $\bigcup$ createHistoryEntry($refAttSet$.category, a_1.category | a_2.category);
  s_{new}.Attributes = $attributeSet \bigcup refAttSet$;
Return $\langle s_{new} \rangle$;

For s_{new} and att_{new}, dot notation is used in accessing their elements specified by Definition 11.

Example 16 *Let us give an example of PEAS derivation by Algorithm 5 for the join of temperature and electricity usage data described in Example 12. Let us assume that the PEAS of electricity usage data has been created with the same logic given in Example 13. First, Algorithm 5 stores the parameters used in the join operation into refAttSet. All attributes of the two streams except parameters used in the join operation are stored as attributeSet to be included in the resulting stream without any modification. Then, the algorithm performs composition of privacy preferences for attributes in refAttSet and updates attributes in refAttSet with newly derived privacy preferences. Then, the new history entry newHE is created by function createHistoryEntry(), and added to the history fields of both the attributes used as parameters in the operation. Finally, the union of the attributes used as parameters of the operation and attributeSet are added to the resulting stream. The updated PEAS for the temperature attribute*

is therefore:

- att = *{24.4},*
- $att.pp$ = $\langle$*temperature, {smart-home-company},* $\langle\{admin\}, \emptyset\rangle$, $\langle\{generic\}, \emptyset, \langle\{admin\}, \emptyset\rangle\rangle$, *{sensitive}*$\rangle$,
- $att.category$ = *{generic},*
- $att.history$ = *{{generic, generic}, {generic}}.*

5.2.3 Compliance verification

Compliance verification is the operation of verifying if constraints specified by data privacy preferences are satisfied by the privacy policy of the consumer. These checks are performed by consumer SOs on every piece of data passing through them. More formally, a consumer SO receives as input a stream S and returns as output a stream $\overline{S}$, containing only those tuples whose attributes' values satisfy privacy preferences of all the owners involved in the generation of attributes' contents. Compliance check implies three main steps: (i) checking compliance of the consumer's identity (i.e., $consumerID$) with the allowed consumers (i.e., $consumer$) specified in the data privacy preference; (ii) checking compliance of the purpose (i.e., ap) of the consumer with the intended purposes (i.e., ip) specified in the data privacy preference; and (iii) checking compliance of the operations performed on the data (e.g., on each single attribute of the input stream) with constraints imposed by the joint access constraint jac and the category derivation constraint cdc.

Step (i) is straightforward in that it should only be checked that $consumerID$ specifies the identity of a consumer allowed to consume attribute A, that is, $consumerID \in A.pp.consumer$. Regarding step (ii), this check is satisfied if ap specifies an allowed purpose, that is, a purpose contained in ip or in the set of allowed purposes implied by ip, if purposes are organized into a tree. More formally, denoting with $\vec{ip}$ the set of purposes implied by ip, ap complies with ip iff $ap \in A.pp.\vec{ip}$.

For step (iii), let us first discuss compliance of joint access constraint jac for a given attribute A with privacy preference pp. Jac states which portions of user's personal data can be combined/aggregated with other data, and with which consumer purpose.In order to explain how jac is verified, let us refer to Example 14. Let us focus on the derived attribute obtained by performing operation Pa_1 of operator Π in Example 14, which computes number of people inside the smart home as derived data. After such operation, in order to ensure compliance of the jac of the newly derived attribute, history of the resulting attribute must be checked, since joint access of temperature and electricity usage data may be prohibited by composed privacy preferences. The set of categories implied by jac denoted as $\vec{jac}$, is composed of all categories that descend from the categories in $jac.Adc$ which do not descend from any category of $jac.Exc$. We say that the jac of attribute A is satisfied iff: (i) all the accessed data categories in every history entry of A belong to the set of categories implied by $\vec{jac}$, and (ii) ap complies with the intended purpose component of jac. Formally:

Definition 12 ***Jac compliance.*** *Let pp be a privacy preference specified for attribute A. Compliance of the pp.jac component is satisfied iff:* $ap \in jac.\vec{ip} \wedge \forall HE \in A.history, \forall c \in HE.\vec{AC} \rightarrow c \in A.pp.\vec{jac}$.

Let us now discuss compliance of category derivation constraint *cdc* for a given attribute A with privacy preference *pp*, which specifies the data categories that cannot be derived from the processing of attribute A. The set of data categories which on the basis of *cdc* cannot be derived denoted as $\vec{cdc}$, is composed of all categories that specialize those referred to within *cdc*. Similarly to what happened for the *jac* component, history of the resulting attribute must be checked, since derivation of new attributes may be performed several times, and every derivation process has to comply with privacy preferences of the attribute. Therefore, we say that *cdc* of attribute A is satisfied iff all the data categories in every history entry of A does not belong to the set of categories implied by $\vec{cdc}$. Formally:

Definition 13 ***Cdc compliance.*** *Let pp be a privacy preferences specified for attribute A. Compliance of pp.cdc is satisfied iff:* $\forall HE \in A.history, \nexists c \in HE.\vec{RC} \wedge c \in A.pp.\vec{cdc}$.

Consumer SOs perform compliance check for any attribute in every tuple before sharing it with consumer. Let us present and describe our algorithm for checking compliance. As previously discussed, *jac* and *cdc* compliance have to be ensured for every entry in the *history* field of PEAS. Algorithm 6 takes as input a tuple s, the id of the consumer $consumerID$, and the access purpose of the consumer *ap*. The algorithm returns an updated tuple, containing attribute(s) that complies with the privacy preferences of the involved users and access purpose

of the consumer.

Algorithm 6: *complianceCheck*$(s, consumerID, ap)$

```
1  Let s_new be a stream, initialized as empty;
2  for each A ∈ S do
3  |  Let cdcFlag and jacFlag be boolean variables, initialized as true;
4  |  Let ipFlag and jacIpFlag be boolean variables, initialized as false;
5  |  if (consumerID ∈ A.pp.consumer) then
6  |  |  if (ap ∈ A.pp.ip⃗) then
7  |  |  |  ipFlag = true;
8  |  |  if (ap ∈ A.pp.jac.ip⃗) then
9  |  |  |  jacIpFlag = true;
10 |  |  for each HE ∈ A.History do
11 |  |  |  Let accessedCategories = HE.AC;
12 |  |  |  Let resultCategories = HE.RC;
13 |  |  |  Let jacDataFlag be a boolean variable, initialized as true;
14 |  |  |  for each cat ∈ accessedCategories do
15 |  |  |  |  Let flag be a boolean variable, initialized as false;
16 |  |  |  |  if cat ∈ A.pp.jac⃗ then
17 |  |  |  |  |  flag = true;
18 |  |  |  |  jacDataFlag = jacDataFlag ∧ flag;
19 |  |  |  for each cat2 ∈ resultCategories do
20 |  |  |  |  Let flag2 be a boolean variable, initialized as true;
21 |  |  |  |  if cat2 ⊆ A.pp.cdc⃗ then
22 |  |  |  |  |  flag2 = false;
23 |  |  |  |  cdcFlag=cdcFlag ∧ flag2;
24 |  |  jacFlag = jacDataFlag∧jacIpFlag;
25 |  |  if jacFlag=true ⋀ cdcFlag=true ⋀ ipFlag=true then
26 |  |  |  S_new = S_new ⋃ A;
27 Return ⟨ S_new ⟩;
```

For s_{new} and att_{new}, dot notation is used in accessing their elements specified by Definition 11.

Example 17 *Let us suppose that a processor SO performs the equi-join and projection* Π_1 *given in Example 16. Moreover, suppose to have the derivation paths presented in Example 15. Algorithm 6 first checks compliance of the consumer (line 5). Let us assume that the consumer entity is smart-home-company. In this case, the check returns true. Then, Algorithm 6 checks compliance of peopleCount.pp.ip and peopleCount.pp.jac.ip with the given consumer access purpose ap (lines 6 and 8). Let us assume that the consumer has specified admin as access purpose. In this case, the check returns true as both the purposes are admin, hence ipFlag and jacIpFlag are set to true. Then, for both jac and cdc, Algorithm 6 checks compliance of every history entry HE. Since peopleCount.history has two elements, the check is performed twice (line 10). In both of them, the algorithm checks that every category cat in HE.AC belongs to*

peopleCount.pp.$\vec{jac}$ (line 16). In our example, as peopleCount.pp.jac allows joint access with data belonging to category generic, $jacDataFlag$ will remain true for both of the history entries of $peopleCount.History$. In the next step, the logical conjunction of $jacDataFlag$ and $jacIpFlag$ is done, to get $jacFlag$. Since both of them are true, $jacFlag$ will be true. Similarly, the algorithm checks that every category $cat2$ in $HE.RC$ does not belong to peopleCount.pp.$\vec{cdc}$ (line 21). For the first history entry, $cdcFlag$ will remain true, as category in $HE.RC$ is not sensitive. For the second history entry, $cdcFlag$ will change to false, since the category in $HE.RC$ is sensitive. Finally, since $cdcFlag$ is false (line 25), attribute peopleCount is not added to the final schema, hence it will not be shared with the consumer.

5.3 Experiments

In order to evaluate the performance and overhead of the proposed mechanism, we have implemented several scenarios introduced in what follows. It is relevant to note that, as highlighted in Table 3.1, the state-of-art approaches reviewed in Section 3.1.2 offer only a subset of the features supported by the proposed framework. This brings us to prefer not having a performance comparison among them.

5.3.1 Experimental scenarios

We developed four experimental scenarios to estimate the overhead at smart object and at network level.

Processor SO I

In this scenario, we consider processor SOs of second smartness level, able only to perform data selection and projection. In particular, we assume that processor SOs I are physically manufactured low capability devices, with hard-coded logic.To simulate this low capability devices, we have implemented smart objects using Freescale FRDM-K64F[7] with *mbed* operating system.[8] This device has 256 KB RAM and ARM Cortex-M4 core. Coding has been done using C++. Due to limited capabilities of the FRDM platform, we were not able to use real world data. As such, in this scenario, we used synthetic data, randomly created on the FRDM platform. We generate a tuple per second, containing a unique attribute of float data type, with randomly assigned value.

Processor SO II

In this scenario, we assume that the processor SO is a device with third smartness level, able to perform, in addition to selection and projection operations, also window-based operations, such as join and aggregation. This processor SO II has been implemented on a Raspberry

[7]developer.mbed.org/platforms/FRDM-K64F/
[8]www.mbed.com/en/platform/mbed-os/

Pi 3 Model B with *Raspbian* operating system, with 1GB RAM and a 1.2 GHz 64-bit quad-core ARMv8 CPU. The implementation has been done with Java SE Development Kit 8u73. Moreover, to implement into processor SOs II an SQL engine able to manage window-based operations, we exploit Esper.[9] We make use of the IoT-Lab testbed[10] data, which provides a very large scale infrastructure suitable for testing small wireless sensor devices and heterogeneous communicating objects. Each node of IoT-Lab testbed produces tuples containing a unique attribute containing data sensed by that node. In particular, IoT-Lab provides three hardware platforms: WSN430, M3 Nodes, and A8 Nodes. Our experiments have been run with data transferred from M3 Nodes, where lumen level data, represented as float values, are sensed from light-to-digital sensor ISL29020.[11]

Consumer SO

We implemented a smart object with third smartness level, able to check privacy preference compliance. We adopt the same simulation and data generation strategies used for processor SOs II (see Section 5.3.1), where compliance verification algorithm has been implemented as an additional Java function.

Smart objects network

We recall that the aim of this experiment is to measure the overall time overhead and bandwidth utilization implied by our enforcement mechanism on a network of smart objects. To simulate this scenario, we should implement a real network of smart objects, where each object is modified so as to embed the PEAS update algorithm as well as the compliance checks. However, the lack of standard platforms on which injecting our modified smart objects makes hard to deploy this setting. To overcome this issue, we decided to simulate a network of smart objects by leveraging on Complex Event Processing (CEP) systems [137]. We exploit the graph-based SQL modelling of CEP to simulate a network of objects, where every single operation in the CEP query is interpreted as a distinct smart object. As an example, the graph-based SQL query in Figure 5.3 can be interpreted as a network of two sensing SOs (i.e., the two INPUT operators), four processor SOs (i.e., the join, projection, aggregation and selection operators), and one final OUTPUT operator. This will allow us to estimate the extra time needed to execute queries with the proposed enforcement mechanisms, as well as the extra space required to encode the needed meta-data into each output tuple.

[9] www.espertech.com/esper
[10] www.iot-lab.info
[11] www.digikey.com/catalog/en/partgroup/isl29020/14151

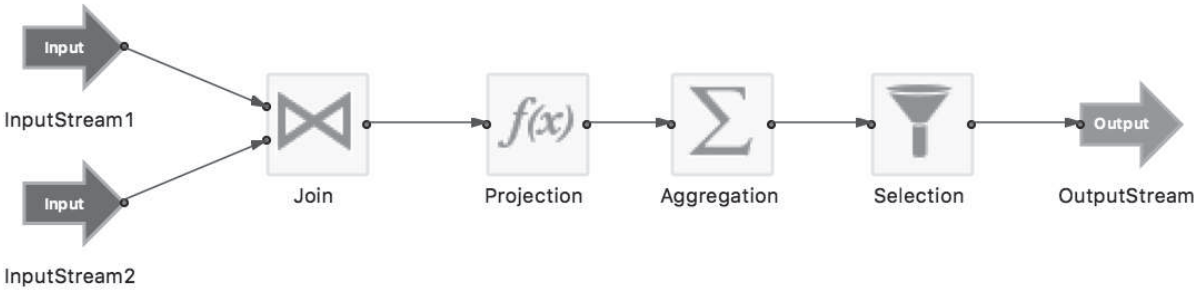

Figure 5.3: Graph-based SQL query

We adopted Streambase as CEP platform, as this product allows us to associate with an SQL operator an additional function, running a Java code, that will be executed by Streambase every time the operator processes an incoming tuple. As such, with Java SE Development Kit 8u73, we defined a set of functions implementing the PEAS update algorithms (cfr. Section 5.2). These functions have been properly associated with INPUT, Join, Selection, Projection, and Aggregation operators of Streambase. Following this design, the compliance verification algorithm should have been associated on the OUTPUT operator. However, since Streambase does not allow this, we implemented a new Streambase operator, to which the function implementing the compliance verification algorithm has been associated. This new defined operator is inserted just before the OUTPUT operator finalizing the query. We have exploited the feed simulation tool from Streambase Studio, that automatically generates and passes test data at specified rates to INPUT operators. Data rate has been fixed to 100 tuples per second for each stream. Number of attributes in each tuple has been regulated according to experiment settings (see discussion on query complexity in Section 5.3.2).

5.3.2 Experimental results

In executing our experiments, we considered four main characteristics that may impact the performance of the proposed solution. These are: privacy preferences complexity, queries complexity, number of sensing smart objects, and percentage of smart objects with associated privacy preferences. However, it was not possible to test all these dimensions in each scenario (see Table 5.1). Indeed, the experiment varying the number of sensing SOs requires to use IoT-Lab dataset, and, as discussed in Section 5.3.1, this dataset has been used only for processor SO II and consumer SO. Experiments have been executed by varying a single dimension and keeping fixed the others with worst case settings. Table 5.1 presents a summary of conducted experiments. In the following, we illustrate the results.

Varying query complexity

Processor SO I: these SOs are only able to perform σ and Π operators. Moreover, since σ operator does not alter PEAS meta-data of processed attributes (cfr. Section 5.2), we only considered the Π operator. We varied query complexity by increasing the number of attributes

Tested dimensions	**Proc. SO I**	**Proc. SO II**	**Consumer SO**	**Smart Object network**
Varying complexity of PP	✓	✓	✓	✓
Varying query complexity	✓	✓	✓	✓ *
Varying number of sensing SOs		✓	✓	
Varying % of sensing SOs with PP		✓	✓	

Table 5.1: Experiments: ✓ - time overhead, ✓ * time and bandwidth overhead

to be evaluated to perform the Π operator. We recall that, according to the adopted notation, the Π operator is modeled as a set of parameters, each one having a set of attributes to be evaluated. Thus, in this experiment, we have considered a Π operator with a fixed number of parameters (i.e., 3), by varying the number of attributes in each of them. As an example, the simplest query is defined as a Π operator with three parameters and two attributes for each parameter. Given a stream S(a,b,c), an example of simplest query is $\Pi_{(a+b),(c-b),(a\times c)}$. Execution times of queries with different complexity levels with and without the proposed privacy preference enforcement are illustrated in Figure 5.4. Even for the most complex queries, time overhead is less than 0.1 ms and performance overhead is less than 7%.

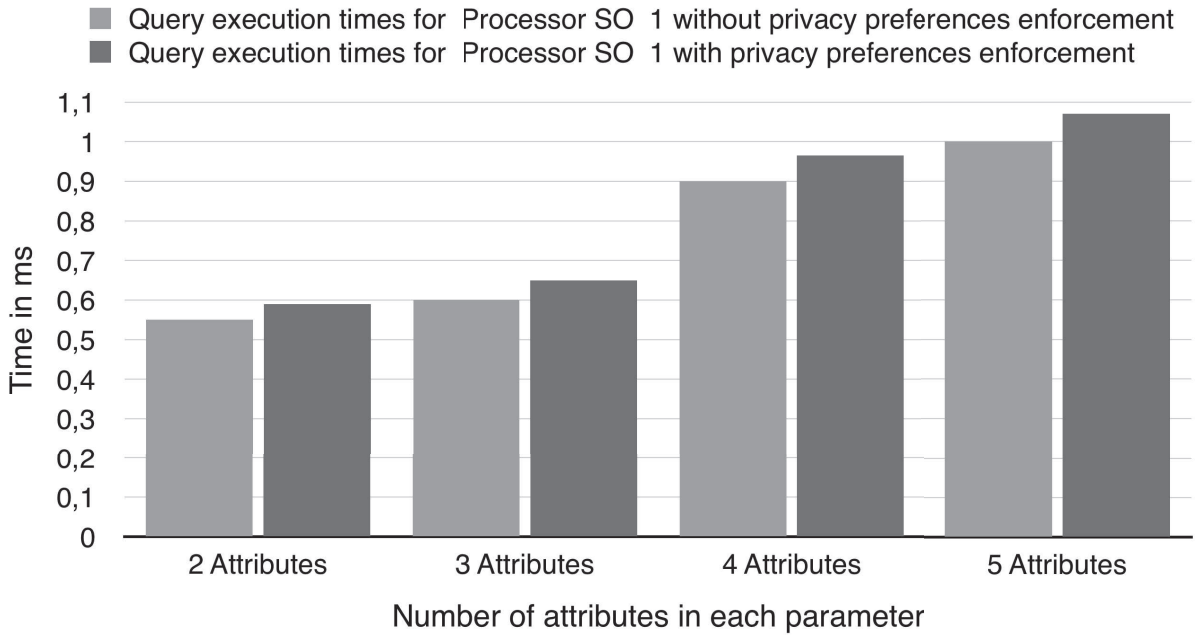

Figure 5.4: Varying query complexity - Processor SO I

Processor SO II: query complexity can be estimated by the number of operators it contains. However, we have also to take into account that smart objects are constrained devices, so not being able to perform too complex SQL-like queries like those done by DBMSs. For instance, the processor SO II has been simulated on Raspberry Pi with only 1 GB RAM. To cope with this limitation, we have considered two sets of queries, with different complexity. The first set, named *Simple Queries*, contains queries consisting only of a Σ operator. We select this aggregation operation as we expect it is the most common task performed by smart objects, that is, aggregating data sensed by sensors. The second set, named *Complex Queries*, contains queries performing a join of data sensed by two clusters of sensing SOs, then a projection over joined attributes, and, finally, an aggregation (i.e., average on 10s window) on projected values. Again this type of queries represents a typical operation for smart objects. Table 5.2 shows query execution time with and without privacy preference enforcement. Even for the most complex queries, the time overhead is less than 0.71 ms and performance overhead is less than 9%.

Queries	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**
Query selectivity	*3,5%*	*1,84%*	*0,85%*	*0,72%*	*0,55%*	*0,37%*	*0,28%*	*0,2%*	*0,15%*	*0,13%*
Extra bits per output tuple	*98*	*140*	*210*	*245*	*294*	*336*	*392*	*455*	*483*	*546*
Bandwidth overload per hour	*3440*	*2576*	*1785*	*1764*	*1617*	*1243*	*1097*	*910*	*724*	*709*

Table 5.4: Varying query complexity - SO network - bandwidth overhead

Processor SO II	**Without PP enforcement**	**Proposed mechanism**
Simple Queries	*5 ms*	*5.36 ms*
Complex Queries	*8 ms*	*8.71 ms*

Table 5.2: Varying query complexity - Processor SO II

Consumer SO: even if consumer SOs are not designed to perform queries, but only compliance verification, we have to note that query complexity might impact the execution of this verification. Indeed, the more complex is the query, the more operations it contains. Thus, a complex query implies, as a consequence, more complex History fields to be evaluated by the compliance verification algorithm. As such, the aim of this experiment is to estimate how query complexity impacts the overhead given by compliance check. For this experiment, we used the same queries generated for processor SO II. Results of experiments are illustrated in Table 5.3. Even for complex queries, the time overhead is less than 0.44 ms.

Scenario	**Simple queries**	**Complex queries**
Consumer SO II	*0.18 ms*	*0.44 ms*

Table 5.3: Varying query complexity - Consumer SO

Smart object network: We simulate a smart object network via a Streambase query, where each single Streambase operator acts as a smart object. Thus, increasing query complexity is like increasing the network complexity. In performing these experiments, we use a set of 10 queries with different numbers and types of operators. In particular, the simplest query, Q1, contains 1 π, 1 σ, 1 $\bowtie$, and 1 Σ operator, whereas the most complex query, Q10, contains 10 π, 10 σ, 10 $\bowtie$, and 10 Σ operators. Results of experiments are illustrated in Figure 5.5. As expected, less complex queries take less time to be processed. However, even in complex network scenarios, the system overhead is always less than 10% which is less than 12 ms.

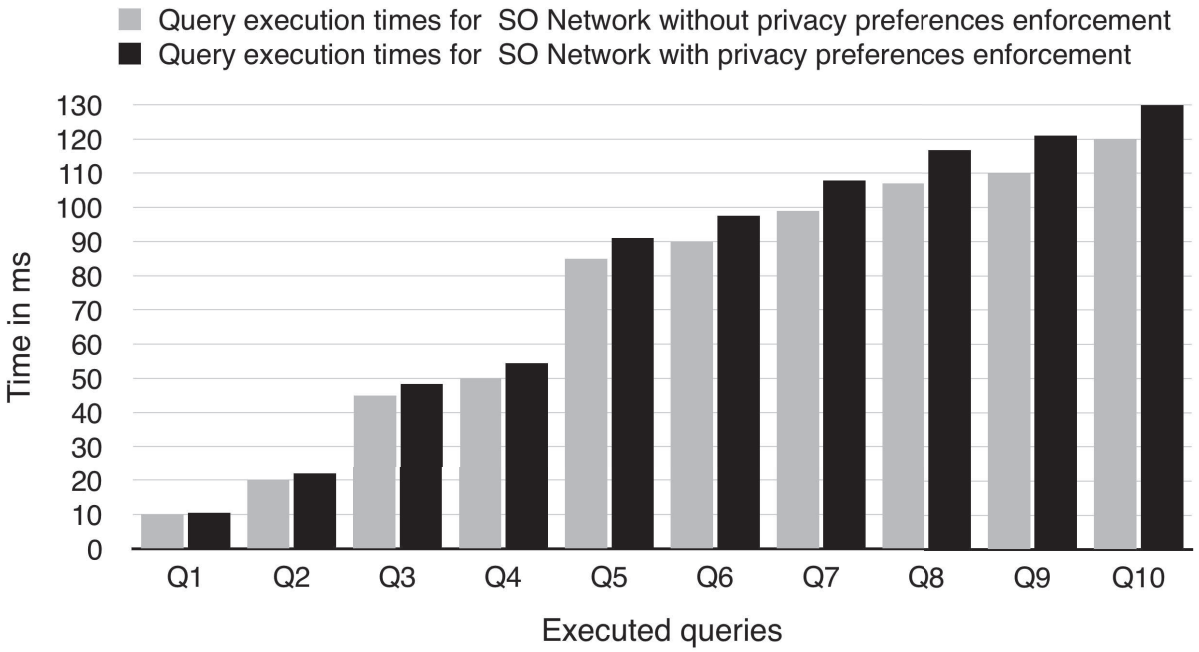

Figure 5.5: Varying query complexity - SO network - time overhead

Regarding bandwidth utilization, we have to note that the overhead is mainly implied by meta-data inserted in the original tuples. We recall that these meta-data are generated and updated by each single operator specified in the query. As such, we expect that a dimension impacting the bandwidth utilization is the query complexity. In addition to query complexity, another element that might impact this overhead is the query selectivity, that can be measured as the ratio of the number of tuples entering and number of tuples returned to/from a query. As query selectivity depends on characteristics of query operators (as an example, in the aggregation operator a bigger window size implies an higher query selectivity), we have modified the benchmark of 10 queries above described so as to tune the operators (e.g., window size in aggregation, selection conditions in Π) to obtain an increasing selectivity.

Number of bits added into output tuple after query execution are illustrated in the second line of Table 5.4. As expected, more complex queries increase the number of bits added to the meta-data. However, it is relevant to note that even the most complex query consumes quite low bandwidth, 0.546 kbits, compared to the network capabilities of devices running similar queries (like Raspberry Pi).[12]

In addition to that, we want to underline that we expect that complex queries will have higher selectivity over input tuples (see first line of Table 5.4), thus less output tuples will be generated. Thus, on the one hand a more complex query increases the number of added bits, but on the other hand, it decreases the number of output tuples and thus the overall bandwidth utilization. To better explain this balance, we run an experiment simulating a smart home

[12]www.pidramble.com/wiki/benchmarks/networking

scenario generating 1,000 tuples per hour, each containing a float data type attribute (e.g., room temperature, used electricity in kw). This implies 32 bits in each single tuple. In case that no queries are executed, that is, all tuples are flooding the network, we have 32 kbit overall bandwidth per hour. From the third line of Table 5.4, we can see that the overall overhead decreases by varying query complexity, aka query selectivity. Therefore, even in the worst case, output tuples generated by queries consume quite low bandwidth, i.e. 3.4 kbits, which translates to overhead less than 11%.

Varying complexity of privacy preferences

We recall that a privacy preference poses a set of conditions to be verified on consumer privacy policy, namely *ip*, *jac*, and *cdc*. The presence of these conditions implies checks and operations both in the compliance verification algorithm as well as in the PEAS generation algorithms. As such, we can see the number of conditions specified in a given *pp* as a dimension to measure *pp* complexity. In view of this, we generated two sets of privacy preferences with two different levels of complexity. In particular, privacy preferences in the first set, named *Simple PP*, have been defined as very simple preferences with a unique condition on the intended purpose field, and thus their compliance verification is easy to be performed. In contrast, privacy preferences in the second set, named *Full PP*, have been defined with a condition on each field (i.e., *ip*, *jac*, *cdc*), so as to make harder the compliance verification. More precisely, privacy preferences have been defined in such a way that their constraints for both *Simple PP* and *Full PP* refer to random elements of the data category and purpose trees, which, in turn, have been populated with synthetic data. In particular, we have considered 100 elements in both the data category and the purpose trees. This test case has been applied to all experiment scenarios, as described in the following.

Processor SO I: since FRDM is an hard-coded device with limited capabilities, we assumed that with each attribute in the synthetic dataset the same privacy preference is associated (i.e., every attribute has the same *pp* value). This corresponds to a scenario where the implemented processor SO I receives as input a flow of data generated by a sensing SO owned by a single user, as such all data are marked with the same privacy preference. However, to test how complexity of privacy preferences impacts the local overhead of processor SO I, we run several times these experiments, changing in each execution the privacy preferences associated with attributes, by selecting them both from *Simple PP* and *Full PP* sets.

Processor SO II, Consumer SO, and Smart object network: differently from processor SOs I, privacy preferences are dynamically generated and randomly assigned to each attribute of the tuples entering in the considered smart object.

In performing these experiments we adopt the settings shown in Table 5.5, which also presents the execution time of queries without privacy preferences (i.e., *No PP*), with simple privacy preferences (*Simple PP*), and complex privacy preferences (*Full PP*). As expected, simple privacy preferences imply less time to execute the compliance verification. However, even with complex privacy preferences the overhead is always under 1 ms and performance overhead is always less than 10% for every scenario.

Scenario	Settings	No PP	Simple PP	Full PP
Processor SO I	Complex queries	*0.31 ms*	*0.325 ms*	*0.34 ms*
Processor SO II	Complex queries, 10 sensing SOs 100% assoc. PP	*3 ms*	*3.10 ms*	*3.28 ms*
Consumer SO	Complex queries, 10 sensing SOs 100% assoc. PP	$\emptyset$	*0.08 ms*	*0.34 ms*
SO network	Complex queries	*10 ms*	*10.74 ms*	*10.99 ms*

Table 5.5: Overhead by varying PP complexity

Varying the number of sensing SOs

Increasing the percentage of sensing SOs with an associated privacy preference will increase the number of privacy preferences to be elaborated (i.e., by PEAS update algorithm in Processor SO II and by compliance verification algorithm in Consumer SO). Thus, we expect that this will impact the execution time. In performing these experiments, we executed queries in the set *Complex Queries*, and we associate sensing SOs with privacy preferences taken from *Full PP* set. Results of experiments are illustrated in Figure 5.8 for Processor SO II and in Figure 5.9 for Consumer SO scenarios.

More precisely, in both scenarios, we varied the number of nodes in IoT-Lab testbed (i.e., number of sensing SOs) from 1 to 10, as we assume this range is reasonable for scenarios like the smart home. In performing these experiments, we executed the queries in the *Complex Queries* set, described in Section 5.3.2, and we associate with each sensing SO a privacy preference taken from the *Full PP* set. Results of experiments are illustrated in Figure 5.6 for processor SO II, and in Figure 5.7 for consumer SO scenarios.

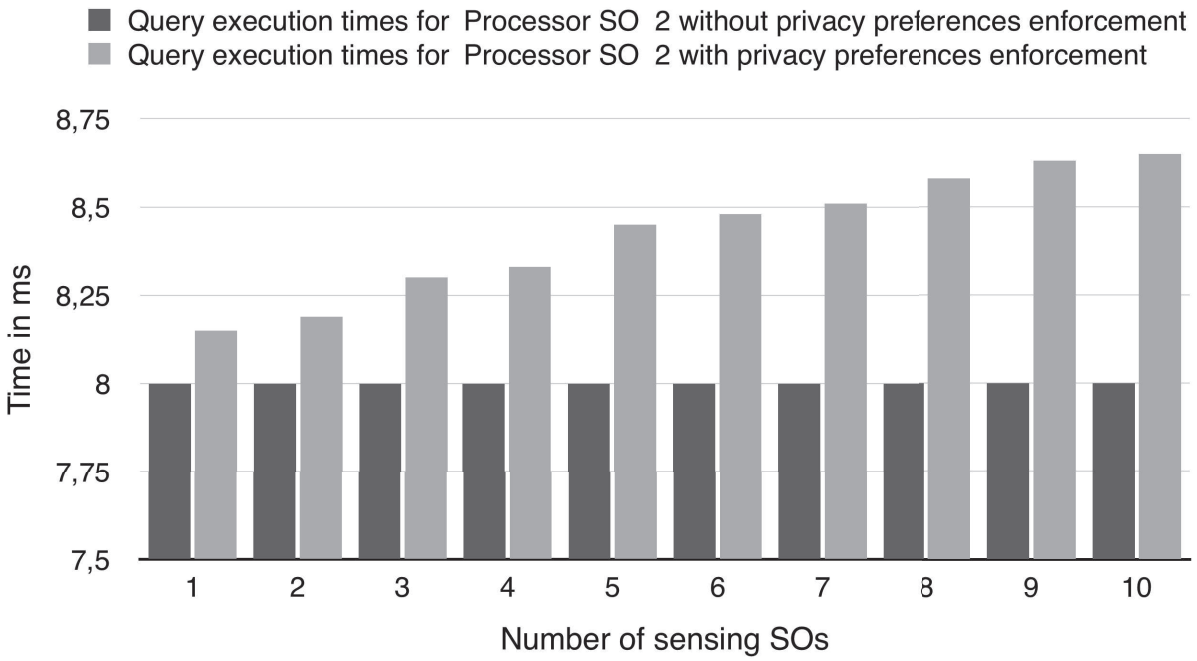

Figure 5.6: Varying the number of sensing SOs in Processor SO II scenario

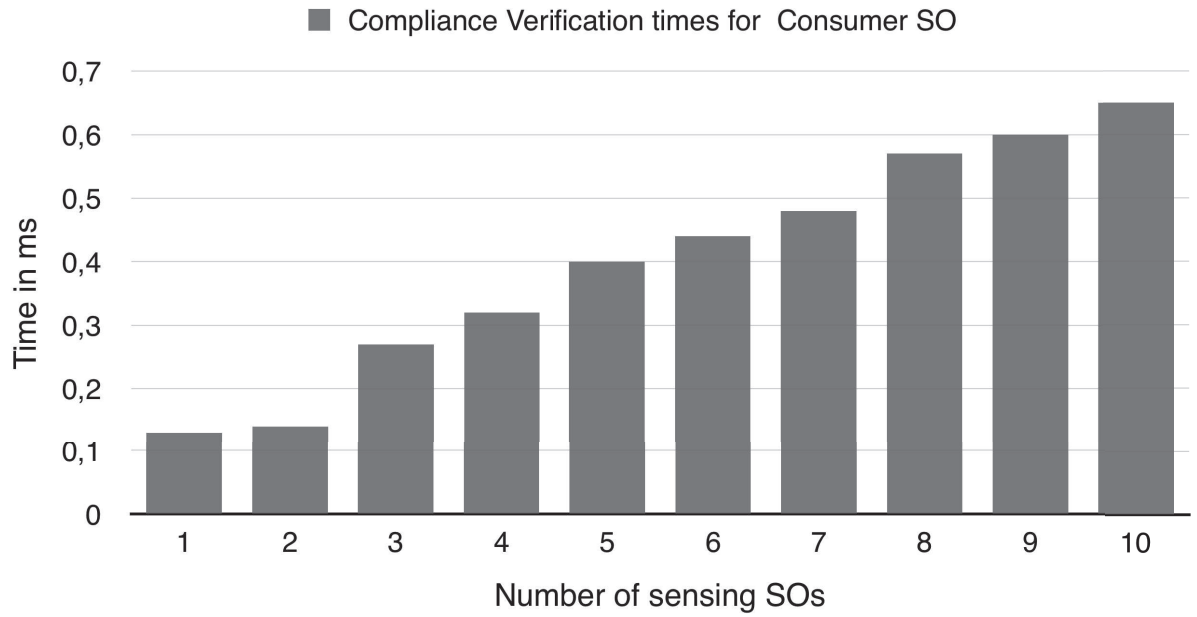

Figure 5.7: Varying the number of sensing SOs in Consumer SO scenario

As expected, for both scenarios having more nodes in the system takes more time to process.

For processor SO II, even in the worst case where we have 10 sensing SOs, overhead is under 0.7ms, thus performance overhead is less than 9%. Compliance check performed by consumer SOs always takes less than 0.7 ms.

Varying percentage of Sensing SOs with privacy preferences

Increasing the percentage of sensing SOs with an associated privacy preference will increase the number of privacy preferences to be elaborated (i.e., by PEAS update algorithm in Processor SO II and by compliance verification algorithm in Consumer SO). Thus, we expect that this will impact the execution time. In performing these experiments, we executed queries in the set *Complex Queries*, and we associate sensing SOs with privacy preferences taken from *Full PP* set. Results of experiments are illustrated in Figure 5.8 for Processor SO II and in Figure 5.9 for Consumer SO scenarios.

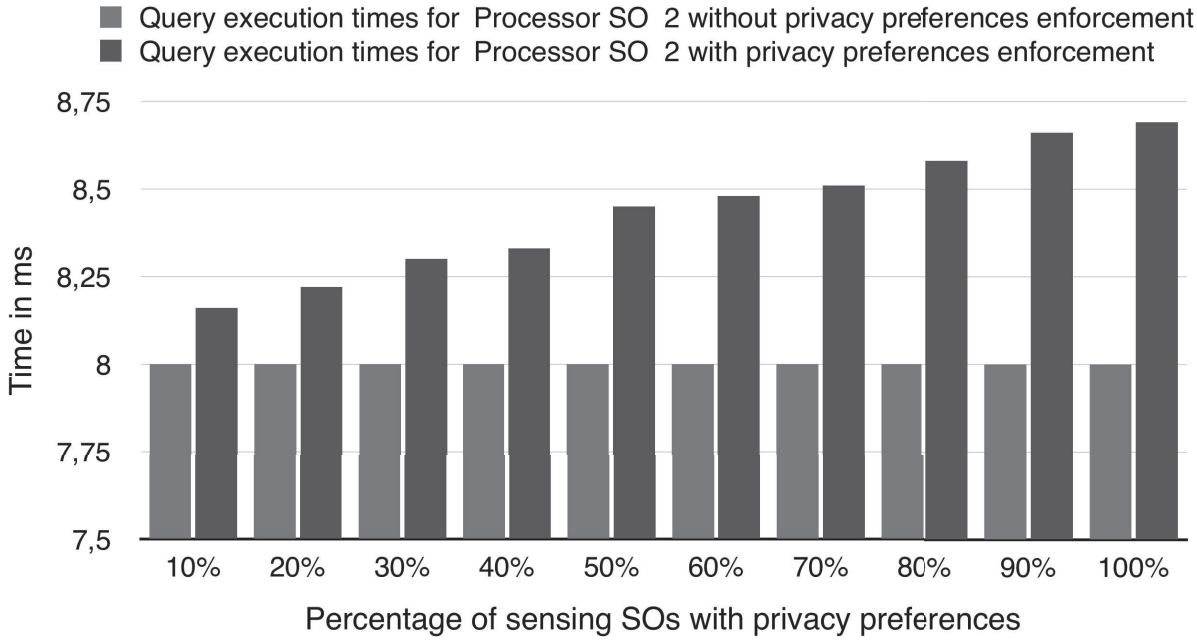

Figure 5.8: Varying number of sensing SOs with associated a PP in Processor SO II scenario

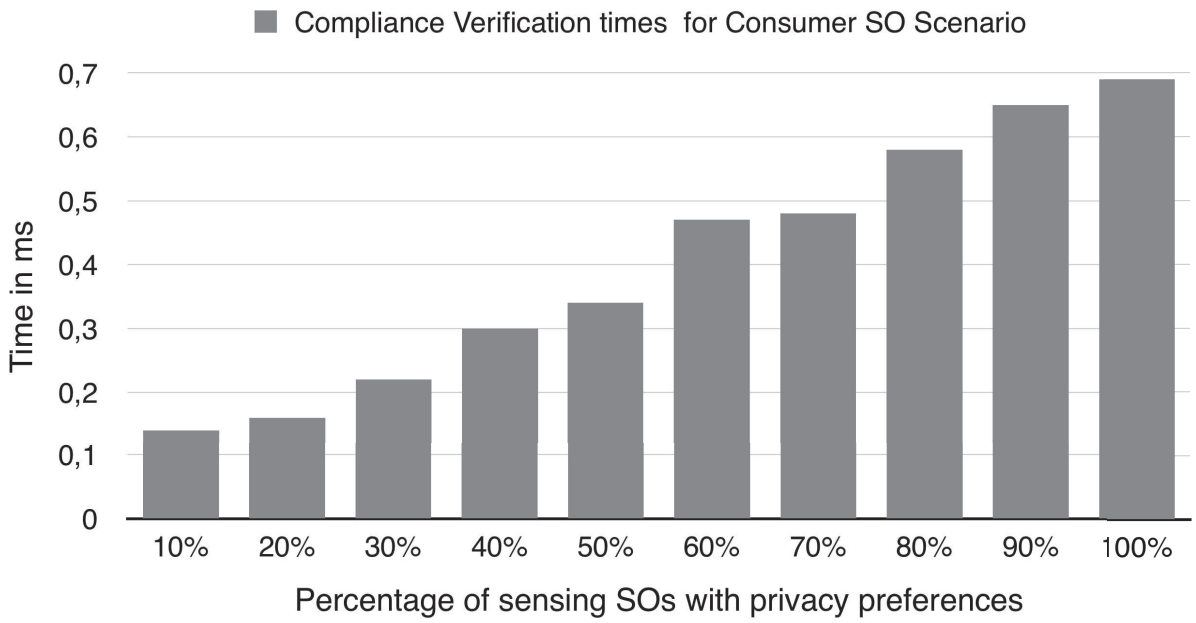

Figure 5.9: Varying number of sensing SOs with associated a PP in Consumer SO scenario

As expected, for both of the scenarios having less nodes with privacy preferences implies less time to process. For Processor SO II, even in the worst case, where all nodes have an associated privacy preference, overhead is under 0.7ms thus performance overhead is less than 9%. Compliance check performed by Consumer SO always takes less than 0.7 ms.

5.3.3 Discussion on experiments results

To test the performance of the proposed mechanism, we tested four dimensions in four scenarios, characterized by networks of smart objects of different smartness levels and roles. In evaluating experiment results, we focus on the time and bandwidth overheads imposed by proposed mechanism. More precisely, the time overhead to processor SO I, processor SO II and smart object network is measured as the ratio of the extra time required to perform PEAS derivation (see Algorithm 1). Experiments show that this is always less than 10%. For consumer SO, the time overhead is estimated as the the time required to perform compliance check (see Algorithm 2). Experiments show that this is always less than 1 ms. On the other hand, bandwidth overhead is measured as the ratio of the bandwidth overload for privacy enforcement to the normal system execution (for a certain period of time), which is always less than %10. Given that, both time and bandwidth overheads given by proposed mechanism are reasonable.

We conclude that benefits of using the proposed mechanism (i.e., users' privacy enforcement in a decentralized setting) outweighs low performance degradation due to time and bandwidth overheads. This proves feasibility of the proposed privacy enforcement framework.

Chapter 6

Blockchain-based P2P Botnet Detection for IoT

Increasing popularity of IoT has made IoT devices a powerful amplifying platform for cyber-attacks [75]. As discussed in Chapter 1, they represent a rather easy target to attackers and the weakest link in the security chain of modern computer networks. As proof of this, a recent study from HP found that more than 70% of IoT devices do not have passwords with sufficient complexity and use unencrypted network services, resulting in being easy targets for attackers.[1]

In such a vulnerable environment, attackers can easily gain access to insecure IoT devices, and inject malicious softwares, *malwares*, to control them or to steal confidential information [112]. Today, one of the most relevant threat posed by malwares in IoT is represented by malicious *botnets*.A botnet is a collection of compromised Internet computers being controlled remotely by attackers for malicious and illegal purposes [132]. For example, some recent Distributed Denial of Services (DDoS) attacks on *Krebs on Security* and *DYN* were due to a malware, named Mirai [8], that uses IoT devices as botnets to generate extensive amount of network traffic, more than 1 Tbps. Additionally, such botnets have been commoditized by malicious parties, known as *booters* [70], that offers DDoS as a service. Booters exploit compromised IoT devices to send attack packets to a target victim, in order to interrupt its service or shut it down. Given that, botnets capable of using tens of thousands of IoT devices pose huge threats to online services' security and privacy.

Botnets. Let us examine in more details the main elements of botnets. Briefly, a typical botnet consists of [132]: i) several *bots*, that is, infected machines running the bot executable; ii) a *Command and Control (C&C) server*, able to control every bot; and iii) a *botmaster*, which is the malicious party controlling the botnet via the C&C server. Early botnets followed a centralized architecture, where the botmaster manages bots via the central C&C server. To increase resilience of their attacks against defence mechanisms, more recent botnet architectures evolved into decentralized P2P architectures. Today, decentralized P2P botnet topologies are able to utilize regular bots as C&C servers [132], thus eliminating the single point of failure

[1] `go.saas.hpe.com/l/28912/2015-07-21/32bhy3/28912/69168/IoT_Report.pdf`

problem. On the other hand, this makes P2P botnets harder to being detected and stopped, as botmasters are able to send attack commands through various channels.

AutoBotCatcher. The design of AutoBotCatcher is driven by the consideration that bots of a same botnet frequently communicate with each other and form communities [39, 149]. As such, the purpose of AutoBotCatcher is to dynamically analyze communities of IoT devices, formed according to their network traffic flow (see problem statement in Section 6.1), to detect botnets. Specifically, it is a blockchain-based P2P botnet detection mechanism for IoT that makes use of two main actors, namely: *agents* and *block generators* (see Section 6.1 for more details). Where, agents are entitled to monitor IoT network traffic flows in their subnets, and send collected traffic information as blockchain transactions. In contrast, by using collected network traffic flows, block generators (i.e., trusted big entities in IoT domain) aim at modeling *mutual contact* information of IoT devices (i.e., connections between IoT devices) and generating mutual contacts graph. This graph is then exploited to detect communities (see Section 6.2.1).

In particular, AutoBotCatcher uses Louvain method [19] to perform community detection on mutual contacts graphs. Since mutual contact information of IoT devices evolves over time, new snapshots of the mutual contacts graph are periodically generated. To this end, AutoBotCatcher exploits states of a BFT blockchain in order to store snapshots of the mutual contacts graph (see Section 6.3). Due to their ability to achieve distributed consensus in processing high throughput of transactions, AutoBatCatcher employs BFT blockchains over PoW blockchains. AutoBotCatcher's BFT blockchain is a permissioned blockchain, where a set of pre-identified block generators generate blocks and participate in the consensus process (see Section 2.2.2). Thus, network data stored on the blockchain is only accessible to block generators.

Discussion. In Chapter 3.2, we reviewed literature on the botnet detection systems. Specifically, we classified botnet detection approaches into two groups as *Network traffic signature based approaches* and *Group and community behaviour based approaches*. AutoBotCatcher differs from such approaches discussed in several ways. First, unlike DPI based approaches, to perform botnet detection AutoBotCatcher requires to trace only high level meta-data about network flow traffic (i.e., source and destination addresses), as such it is effective against encrypted C&C channels. Second, unlike network traffic signature based approaches, even if botmasters randomize network traffic by changing packet sizes, communication frequency etc., botnet community structures do not alter, since the same set of commands has to be shared with the same set of bots. Third, unlike other group and community behaviour based approaches, AutoBotCatcher performs dynamic community detection by exploiting blockchain, so it is able to detect emerging and unknown botnets, and to take preventive measures against them.

The remainder of this chapter is organized as follows. In Section 6.1, we present the considered problem statement and the main entities involved in AutoBotCatcher. We provide background information on mutual contacts graph, and community detection approaches in Section 6.2. In Section 6.3, we introduce the blockchain paradigm defined for AutoBotCatcher. We detail the design of AutoBotCatcher in Section 7.3.

6.1 System Model

In this section, we introduce the problem statement and the main entities of AutoBotCatcher.

Problem Statement. We assume to have a network of IoT devices, gateways, and some external hosts that communicate with IoT devices (such as device vendors' servers, cloud services). IoT devices are connected to the Internet, they sense and process data, and communicate with other IoT devices or external hosts. Botmasters compromise IoT devices and make them part of their botnets for malicious purposes, such as performing DDoS attacks. On the other hand, gateways, such as Dell Edge Gateways[2], are located in network boundaries and monitor the Internet traffic to/from IoT devices within their networks, referred to as their *subnet*. We assume that gateways are trusted, as such, they behave as expected, and cannot be compromised by botmasters.

AutoBotCatcher targets P2P botnets, where all bots potentially can be utilized as C&C servers by botmasters, and performs community analysis on network traffic flows of IoT devices to detect botnet communities. We assume that a botnet community is a group of compromised IoT devices that frequently communicate with each other and with the same set of botmasters. AutoBotCatcher relies on *mutual contacts* information of IoT devices, which refers to shared connections between a pair of IoT devices and/or other hosts. For example, let us assume *Host A*[3] is connected to *Host C*; given that, if *Host B* is also connected to *Host C* then *Host A* and *Host B* share a mutual contact, that is, *Host C*. As discussed in [39] and [149], bots of a P2P botnet communicate with at least one mutual contact with very high probability, therefore mutual contacts can be exploited for botnet detection [39]. Given that, AutoBotCatcher exploits mutual contact information of IoT devices in performing botnet community analysis (see Section 7.3).

Our assumptions on the threat model are as follows: IoT devices can become part of a botnet anytime; new types of botnets may emerge in the network; botmasters encrypt C&C channels, and therefore DPI techniques are not suitable; botnets tend to hide their operations and botmasters try to stay as stealthy as possible [132], where botnets are able to manipulate characteristics of bot traffic, and thus they are able to make network flow traffic signature based defense approaches ineffective; botnets are in their waiting stage, where bots are joined to the C&C network and wait for commands from the botmaster, thus their malicious activity may not be easily observable. The goal of AutoBotCatcher is to dynamically identify IoT devices and other hosts in the network that are part of botnets.

Entities. AutoBotCatcher consists of two main entity types:

– *Agents:* They are typically gateway devices that are deployed in the network boundaries. AutoBotCatcher's agents monitor network traffic flows of IoT devices in their subnet and take actions, such as: generating network-data transactions (NTs) (see Definition 14) and forcing infected devices to shut down. In AutoBotCatcher agents are considered trusted. As such, we assume they behave as expected, and can not be compromised and act maliciously.

– *Block Generators:* This role is played by big entities in the IoT domain, such as device

[2]dell.com/us/business/p/edge-gateway

[3]Here, *Host* refers to both IoT devices and other hosts in the network

vendors, Internet service providers (ISPs), security/privacy regulators, Block generators collaborate to achieve large-scale defense and protection from botnet threat without trusting each other with the help of a BFT blockchain. For effective botnet detection and prevention, collaboration of different device vendors and ISPs is very important, as recent IoT botnets, such as Mirai and Hajime, infected IoT products from various vendors.[4] In fact, malwares behind those botnets are adaptable to the various device architectures, such as ARM and Intel, and to different products, and they were effective in all around the world. We assume that block generators have enough computing and network resources available to devote to P2P botnet detection process operations (see Section 7.3).

6.2 Background

In this section, we provide background information needed to understand the rest of the chapter. To this end, first, we explain the mutual contacts graph concept; and then, we provide an overview of community detection approaches.

6.2.1 Mutual Contacts Graph

In AutoBotCatcher, mutual contacts graphs are exploited as a graph based data representation of the mutual contacts of IoT devices and other hosts in the network. We denote a mutual contacts graph as $G = (V, E)$, where IP addresses of IoT devices and other hosts are *vertices* (V). Vertices share an *edge* (E), if they have at least one mutual contact. Edges are bidirectional and weighted, where number of mutual contacts between vertices is the weight of the edge between them. As such, by referring to the example of mutual contacts given in Section 6.1 and assuming that there are only three hosts in the network, *Hosts A, B, C* are vertices in the mutual contacts graph, where *Host A* and *Host B* share an edge with weight 1, as they share the mutual contact *Host C*. In AutoBotCatcher, the whole topology of the mutual contacts graph is represented by a 2 dimensional weighted adjacency matrix, referred to as *mutual contacts matrix (MCM)*, whose element MCM_{ij} indicates the number of mutual contacts between vertices i and j. For all vertices i and j, $MCM_{ij} = MCM_{ji}$.[5]

6.2.2 Community Detection

Bots of the same botnet use similar C&C channels and share the same messages [139, 149], as such they are much likely to share many mutual contacts [39] than legitimate P2P hosts. Therefore, P2P bots show community behaviours and form community structures that can be useful for botnet detection. Given that, AutoBotCatcher performs *community analysis* on mutual contacts graphs.

Community detection methodology. In AutoBotCatcher, accurate and fast detection of communities in the mutual contacts graph is of great importance for proper botnet detection.

[4] symantec.com/connect/blogs/hajime-worm-battles-mirai-control-Internet-things
[5] This is due to the fact that mutual contact relation applies to both of the vertices.

Literature offers several community detection approaches for systems modeled as graphs (see [52] for more details). The main objective of these methods is to find good partitions on the graphs as possible communities in the network. How to measure goodness of a partition is an important concern in designing community detection methods. In general, a *quality function* is used as a quantitative criterion that assigns a number to each partition of a graph to rank partitions based on their score [52]. Notably, *modularity* is the most popular quality function, where achieving high modularity translates to a better partition of the graph, thus better community structure (for further discussion on the methodology please refer to [97]). Given that, AutoBotCatcher exploits a modularity-based *Louvain* method [19], that is, an hierarchical greedy algorithm trying to improve modularity for community detection.[6] Louvain method has many advantages that makes it a good choice for AutoBotCatcher, such as: it is faster and achieves higher modularity than other methods; and it is able to process large networks in a short time.[7]

6.3 The Blockchain Paradigm

We devote this section to explain how blockchain is used in AutoBotCatcher.

Transactions. To detect botnets, AutoBotCatcher employs community detection analysis on the mutual contacts graph. In particular, to generate mutual contacts graphs, AutoBotCatcher needs to collect and analyze meta-data information about network traffic flow of IoT devices (i.e., IP addresses). To this end, we exploit blockchain as a shared data store to audit meta-data, which are thus modelled as blockchain transactions sent by agents to the *transaction pool*, a shared data store hosted by all block generators, that holds unprocessed transactions. Particularly, each agent is connected to one block generator to send transactions, where block generator that receives a new transaction disseminates new transactions to other block generators.

Meta-data are encoded into *network-data transaction (NT)*, formally defined as follows:

Definition 14 ***Network-data Transaction (NT).*** *Let $NetFlow$ be the network traffic flow defined as $NetFlow = (IP_{src}, IP_{dest})$, where ăIP$_{src}$ and IP_{dest} are the source's and destination's IP addresses, respectively. Let $Device_{addr}$ be the unique public key of the agent that sends the transaction, and let $Tx - Pool_{addr}$ be the public key of the transaction pool that agent is connected to send the transaction, A network-data transaction is a tuple: $NT = \langle Device_{addr}, IP_{src}, IP_{dest}, Tx - Pool_{addr} \rangle$*

Blocks. Transactions are bundled into blocks that are generated by block generators.[8] In what follows, we symbolize a block as b_m, where m represents the block number. Given that,

[6] Despite initially being designed for unweighted graphs, it can be easily adapted to weighted ones.

[7] It took 12 minutes for a network containing 39 million vertices and 783 million edges [19] with AMD dual opteron 2.2k, 24 GB of RAM.

[8] Number of transactions in a block is regulated according to the block size and transaction size settings of the blockchain protocol.

the block structure is as follows:

$$b_m = \{NT_x, \ldots, NT_y\} \tag{6.1}$$

Where, NT represents a network-data transaction, and x and y are the transaction number. During the execution of AutoBotCatcher, one of the block generators is elected as a leader to generate a block. Block generation interval, symbolized as τ, represents the amount of time between consecutive block generations. Upon generation of a block, at least two-thirds of block generators should send acknowledgements to the block generator regarding their approval on the block.[9] Once acknowledgements have been obtained and τ is expired, block generator of the next block generates the new block.

Rounds. In AutoBotCatcher, P2P botnet analysis is periodically performed upon generation of a set of blocks. We refer to such periods as *rounds*. Each round takes certain amount of time, symbolized as Δ. Value of Δ depends on both the amount of time needed for block generators to perform botnet detection operations on the blocks (e.g., mutual contacts graph extraction, botnet community detection, etc.), network quality (e.g., network connection delays), and blockchain protocol (e.g., transaction throughput etc.). A round is symbolized as Γ_{Δ_t}, where Δ_t is a timestamp specifying the starting time of the round. When the round Γ_{Δ_t} ends, the next round $\Gamma_{\Delta_{t+1}}$ starts. In round Γ_{Δ_t}, botnet detection is performed on NTs sent during the previous round $\Gamma_{\Delta_{t-1}}$. More precisely, NTs sent from timestamp Δ_{t-1} to timestamp Δ_t (which corresponds to execution time of round $\Gamma_{\Delta_{t-1}}$) are processed in round Γ_{Δ_t}.

State. In general, in blockchain the *state* notion can be used to represent financial balances of users (e.g., Bitcoin), or, in a broader setting, it can represent anything that can be modeled as result of the computer programs (i.e., on Ethereum blockchain [136]). In our setting, the state is a mapping between IoT devices' and other hosts' IP addresses (that are subject to botnet detection) and communities (each marked as botnet community or benign community). In AutoBotCatcher, similar to the Ethereum protocol [136], the state is not stored on the blockchain, rather it is maintained on an efficient Merkle tree implementation,[10] such as Patricia Merkle Trees.[11] Merkle trees are hash based data structures, where each node is the hash of its children or hash of the data, if the node is a leaf. Main benefits of using Merkle trees to store states are: they are immutable data structures, thus we are able to secure entire system states with cryptographic dependences; and, they allow the blockchain protocol to trivially revert to any old state by simply altering the root hash [136].

In AutoBotCatcher, the state of the blockchain on timestamp Δ_t, symbolized as σ_{Δ_t}, consists of: the snapshot of the latest version of the mutual contacts graph G_{Δ_t}, that is generated from the network-data transactions; the set of all communities and IP addresses of hosts associated with that communities, extracted from the mutual contacts graph by block validators, symbolized as $CommSet_{\Delta_t}$; and, all blocks of the blockchain. Given that, the state of the blockchain on timestamp Δ_t is represented as follows:

[9] If the block generator does not get approval from at least two-thirds of block generators, that block will not be added to the blockchain, and a new block generator will be selected to generate a block.
[10] Exploiting Merkle trees in our blockchain setting requires a simple state database backend.
[11] github.com/ethereum/wiki/wiki/Patricia-Tree

$$\sigma_{\Delta_t} = (G_{\Delta_t}, CommSet_{\Delta_t}, [B_0, \ldots, B_m]) \tag{6.2}$$

In Figure 6.1, we present an example of rounds and states, where each round includes generation of three blocks.

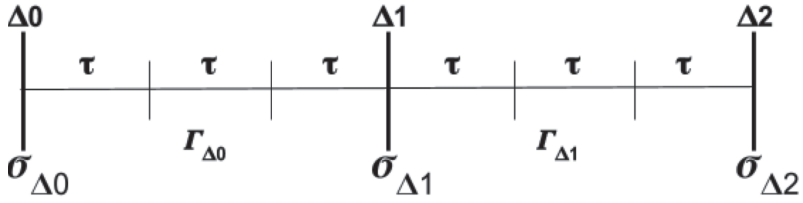

Figure 6.1: Round and state relation

State transitions. In blockchain, state transition refers to achieving a new valid state after executing a set of transactions on the previous state. AutoBotCatcher uses the community set and snapshot of the mutual contacts graph of the previous blockchain state to dynamically perform botnet community detection. Therefore, in AutoBotCatcher, a state transition occurs upon finishing a round of botnet detection operations (see Section 7.3 for more details). Given that, for every round Γ, one state change occurs upon execution of the set of blocks. Particularly, upon execution of Γ_{Δ_t}, the state changes from σ_{Δ_t} to $\sigma_{\Delta_{t+1}}$. We define the state transition as a function of rounds, which takes previous state and new blocks as input, as follows:

$$\sigma_{\Delta_{t+1}} = \Gamma\left(\sigma_{\Delta_t}, [B_m, \ldots, B_{m+z}]\right) \tag{6.3}$$

Where σ_{Δ_t} is the previous state (see Equation 6.2), and $[B_m, \ldots, B_{m+z}]$ is the set of new blocks, where m and z represent block number (see Equation 6.1).

Consensus. Blockchain protocol adopted by AutoBotCatcher uses Byzantine Fault Tolerant (BFT) methodology to achieve consensus. In BFT blockchain, in order to achieve consensus, at least two-thirds of the block generators have to agree on the latest state proposed by one of the block generators. In our setting, this translates to agree on: the same set of blocks processed; same mutual contacts graph; and same community mapping of IoT devices.

6.4 System Architecture

AutoBotCatcher exploits a BFT blockchain as a backend module to perform dynamic and collaborative botnet detection on large scale networks. An overview of the execution flow of AutoBotCatcher is presented in Figure 6.2 and discussed in what follows.

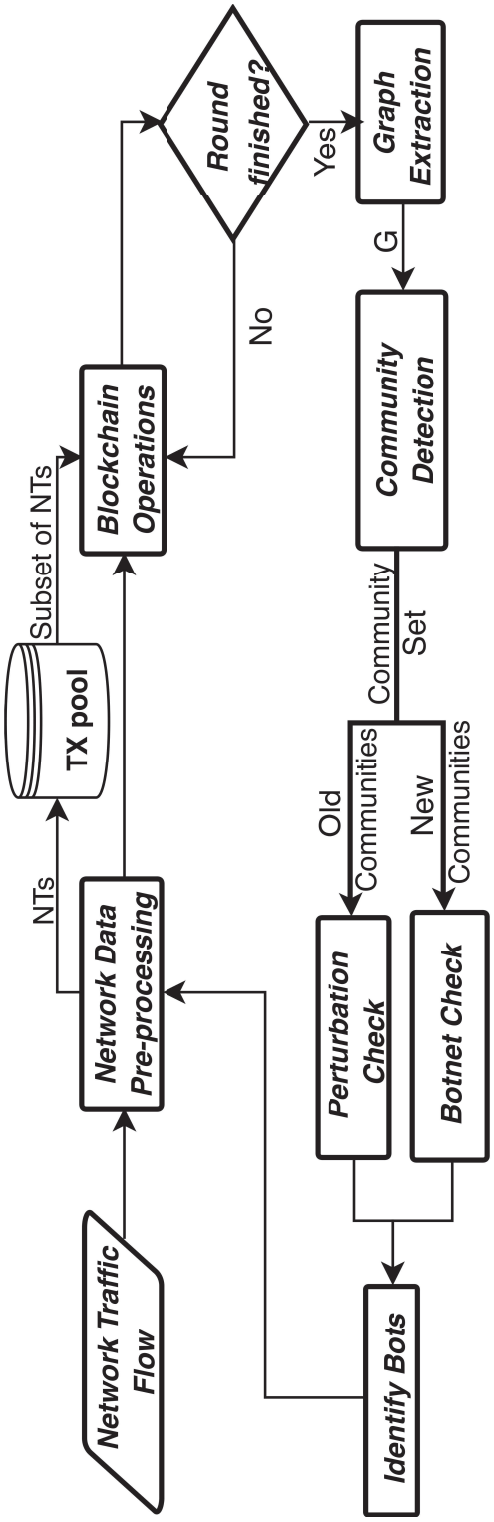

Figure 6.2: AutoBotCatcher system flow

Network data pre-processing. This task is executed by agents for monitoring network traffic flow of IoT devices and taking actions according to that. In performing such operations, agents maintain two types of IP address lists, namely *blacklist* and *whitelist*. Blacklists contain IP addresses that have been previously detected as part of a botnet. On the other hand, whitelists contain predefined and trusted IP addresses,[12] such as IP addresses of the vendor's servers, for all IoT devices in their subnet.

More precisely, an agent constantly sniffs network traffic flows of IoT devices in its subnet; the agent does not take any action for network traffic of an IoT device with a whitelisted IP address; for all other network traffic flows, the agent forms network flow transactions (NTs) (as given in Definition 14), if a network flow is with a blacklisted IP address, then agent quarantines that IoT device either by forcing it to shut down or by cutting all of its network connections.

Blockchain operations. This task is executed by block generators for the generation and relay of blocks that include NTs. To perform these operations in a round Γ_{Δ_t}, first, one block generator is selected to generate one or more blocks.[13] That block generator takes the subset of NTs from the transaction pool, it forms a block and broadcasts the block to all block generators. We recall that each block has to receive approval from at least two-thirds of the block generators in order to be valid, and to achieve consensus on its validity in a BFT blockchain setting.

Graph extraction. This task is executed by block generators, aiming at elaborating the network traffic flow data, that is, the NTs processed in a round Γ, to generate/update the mutual contacts graph G. Particularly in round Γ_{Δ_t}, block generators create the mutual contacts matrix $MCM_{\Delta_{t+1}}$, representing $G_{\Delta_{t+1}}$ that results from the state $\sigma_{\Delta_{t+1}}$, by updating MCM_{Δ_t}, i.e., G_{Δ_t} of state σ_{Δ_t}, with the NTs in blocks sent from Δ_{t-1} to Δ_t.

Operations to transit from MCM_{Δ_t} to $MCM_{\Delta_{t+1}}$ are: *updating weights of edges* between vertices that communicated in round Γ_{Δ_t}; *adding vertices and edges*, if new IoT devices connect to a device network, or a new host IP address communicates with an IoT device; *removing vertices and edges*, if some IoT devices or hosts do not exist anymore.

Community detection. This task, executed by block generators, performs dynamic community discovery (DCD) on the mutual contacts graph G.

AutoBotCatcher performs dynamic community detection with Louvain method on snapshots of the mutual contacts graphs from consecutive states of the BFT blockchain. Yet, as presented in [12], even small changes in consecutive network snapshots may cause Louvain method to generate two alike community structures, which would eventually cause AutoBotCatcher to lose track of the communities and thus degrade its execution. Therefore, to have a more stable community structure between timestamps, AutoBotCatcher initializes the Louvain algorithm for community detection at timestamp Δ_{t+1} with the communities found in Δ_t as proposed by Aynaud *et al.* in [12]. More precisely, in round Γ_{Δ_t}, AutoBotCatcher performs community detection on newly extracted mutual contacts graph $G_{\Delta_{t+1}}$, by feeding community set of IoT devices, $CommSet_{\Delta_t}$ from the last state δ_{Δ_t} to Louvain method.

[12]We assume that hosts corresponding to IP addresses in whitelists are secure, and do not pose any threat to IoT devices.

[13]For the sake of brevity, we do not detail the election process, but any leader election process performed in a typical distributed system is suitable for our setting.

Perturbation check. This task is performed by block generators to check updates (i.e., IP address additions and removals) on communities that already exist in the previous timestamp.[14] Particularly, for botnet communities, IP addresses of new bots are inserted to the list called *additions to blacklist*, whereas IP addresses of bots that are no longer part of a botnet are inserted to the list called *removals from blacklist*. Upon block generators achieve consensus on the new state, block generators share those lists with agents (see task bot identifier).

Botnet check. Executed by block generators, this task is responsible for classifying new communities as either botnet or benign communities. Botnet detection methodology used by AutoBotCatcher is based on two observations: 1) bots connect with each other for command exchange, thus they have more mutual contacts than benign communities [149]; 2) botmasters or attack targets communicate with many nodes, referred to as pivotal nodes [134], so that they have very high number of mutual contacts. Therefore, according to the first observation, AutoBotCatcher calculates the average number of mutual contacts per IP address for all communities. If the result is higher than a pre-defined mutual contacts threshold θ, that community is marked as *candidate botnet* community. Secondly, AutoBotCatcher checks for pivotal nodes in candidate botnet communities. Particularly, for all IP addresses in a candidate botnet community, AutoBotCatcher sums their rows in the mutual contacts matrix (MCM). If one or more pivotal nodes exists in the candidate botnet community, it is marked as botnet community by the block generator. Upon block generators achieve consensus on the new state, block generators share IP addresses in botnet communities with next task for bot blacklisting.

Identify bots. The last task is responsible for updating the bot blacklists of agents, and it is executed by block generators after a state change. It updates the blacklists of all IoT devices with the help of smart contract transactions for addition and deletion of IP addresses on agent's local blacklists.[15]

[14] If a community changes more than a threshold ϕ, calculated as the ratio of total changes to number of edges and vertices, that community will be considered as a new community, and next task will be executed on it.

[15] As it is not the main focus of our work, we do not detail how this mechanism works, such as transaction structure, which is left as future work.

Chapter 7

Hybrid Blockchain Architecture for IoT

IoT has the potential to make several environments smarter more connected, profitable, and efficient, which typically requires the connection, concerted operation and management of a distributed large number of loosely coupled smart devices that need to identify and trust each other. While this should ideally map to a decentralized hardware and software platform, current solutions are mostly based on centralized infrastructures which have many disadvantages, as presented in Chapter 1.

Decentralization, if achieved, would have many advantages over centralized IoT infrastructures, as discussed in Chapter 1. This, from a distributed systems point of view, means achieving distributed consensus. A promising decentralized platform for IoT is blockchain. In fact, when it comes to IoT, blockchain can be used to store critical machine-to-machine communications, sent as blockchain transactions, ensuring accountability and security of the stored data. It can also provide identity and proof of provenance of IoT devices with its cryptographic functions. While the literature offers some examples of blockchain technology in IoT, such as [15, 64, 78], up to now there is no de facto standard solution.

Indeed, one of the biggest challenges in the integration of blockchain into IoT is scalability. In fact, due to the massive number of devices and resource constraints, deploying blockchain in IoT is particularly challenging. The optimal blockchain architecture has to scale to many IoT devices (they become the peers on the blockchain network), and it should be able to process a high throughput of transactions.

Hybrid-IoT, the platform designed in this work, exploits both PoW blockchains and BFT protocols. First, PoW blockchains are used to achieve distributed consensus among many IoT devices, the peers on the blockchain. To measure and qualify that, we define a set of PoW blockchain-IoT integration metrics, and we evaluate the performance of Hybrid-IoT subject to varying blockchain block sizes and block generation intervals, device locations, and number of peers. Since we first observed that PoW blockchains containing few hundreds of geographically close IoT devices have high performance (i.e., high transaction throughputs) and low block propagation delays, the first step in Hybrid-IoT consists of generating multiple PoW block-

chains. Those are generated according to a set of rules, referred to as sweet-spot guidelines, that combine best practices in designing sub-blockchains. As a second step, Hybrid-IoT leverages on a BFT inter-connector framework, such as Polkadot[1] and Cosmos[2], in order to achieve interoperability among sub-blockchains. In this work, we only deal with the first step, that is, analyzing the performance and security of the PoW sub-blockchain setting.

Furthermore, in Hybrid-IoT we define three roles for IoT devices according to their capabilities, and test the performance of the system with a set of experiments and simulations. Moreover, we extensively test security of our approach by acknowledging that generating sub-blockchains may generate security vulnerabilities. We would like to note that, in designing Hybrid-IoT we assumed that IoT devices with fixed location and network connection.

We stress that, in this work, we make extensive use of Bitcoin clients and a Bitcoin simulator to conduct the performance analysis. The Bitcoin client approach is used to test the Hybrid-IoT architecture and design. The real focus here is the PoW sub-blockchains design with the sweet-spot guidelines and the type of tests and analysis performed here would not differ with other types of PoW protocols that allow smart contracts (this would affect only the types of transactions submitted to the peers).

In the literature, there are few papers targeting application of blockchain to IoT (cfr. Chapter 3.3.1). However, application of blockchain to IoT has been mainly limited to application specific tasks (e.g. firmware updates of IoT devices [78]), whereas the goal of our work is to decentralize IoT by exploiting blockchain.

Limitations related to scalability of PoW blockchains have been widely studied [43, 40] and different methodologies have been proposed to overcome such limitations [79, 122, 50, 13]. Particularly, one notable approach is using block Decentralized Acyclic Graphs as reconstruction of classical longest chain protocols [79, 122]. Another significant proposal is Bitcoin-NG protocol [50] by Eyal *et al.* Bitcoin-NG protocol proposes use of two types of blocks in the Bitcoin blockchain, namely; key blocks and microblocks, to achieve higher transaction throughput. Differently, [13] Back *et al.* proposes the use of multiple interoperable blockchains, referred to as pegged sidechains, to allow assets in different ledgers to be transferred between each other. We would like to underline that, Hybrid-IoT can benefit from such proposals by adopting their methodologies to increase throughput and enhance interoperability of PoW sub-blockchains. integration of their methodologies throughput of PoW sub-blockchains can increase.

The remainder of this chapter is organized as follows. We define PoW blockchain-IoT integration metrics in Section 7.1. In Section 7.2, we extensively evaluate PoW blockchain-IoT integration and define sweet-spot guidelines for sub-blockchain generation. We detail the design of Hybrid-IoT in Section 7.3. In Section 7.4, we evaluate performance of the sub-blockchains. Security of our approach is discussed in Section 7.5.

[1] polkadot.network
[2] cosmos.network

7.1 PoW Blockchain-IoT Integration Metrics

We identify five relevant dimensions that an optimal blockchain PoW implementation for IoT should be subject to: *scalability*, *security*, *decentralization*, *efficiency* as observed metrics, and *network bandwidth* as a controlled parameter. In what follows, we analyze those dimensions (see Table 7.1 for a summary).

Scalability. Scalability in IoT is the capacity to be changed in size or scale in terms of number of devices, hardware characteristics and functional and non-functional requirements, while maintaining quality of performance. For blockchain, this translates to have a peer to peer network that can scale up in terms of number of peers and throughput, as number of transactions per unit of time.

Security. Security is a critical dimension in IoT, especially considering recent large scale attacks ,like Mirai and WannaCry.[3] While in this work we do not deal with device intrusions, the issue of data integrity for IoT devices is an important problem to be solved [80]. While data integrity is by design preserved by a PoW blockchain, the issue of the longer chain attack still exists [55]. In order to measure this, we consider the maximum amount of total work in the PoW sub-blockchain as a metric.

Decentralization. Decentralization in IoT is critical to improve security and privacy and achieve autonomous execution, as noted in Section 2. In peer-to-peer overlay networks, like blockchains, decentralization is measured by the number of properly functioning peers [40]. In a blockchain, a peer needs to be up to date with the most recent block before generating a new block to be accepted by blockchain consensus. Hence, we define the metrics for measuring decentralization as the number of functioning peers on the network. We also define a lower bound of functioning peers, to be 90% of the total, to guarantee proper functionality of the blockchain for its IoT application.

Efficiency. Efficiency in IoT can be defined as an optimal utilization of hardware resources and energy. Therefore, in order to achieve that, the IoT devices on the blockchain should optimally utilize resources and energy to maintain and progress the blockchain. Among others, an obstacle to that is the issue of forks and stale blocks in PoW blockchains [55]. Specifically, stale blocks do not contribute to the security of the blockchain and transactions in stale blocks are considered as unprocessed by the network, requiring wasted effort to generate them.[4] Hence, we define our metrics for efficiency as the stale block generation ratio and we establish a upper bound for performance to be $\approx$1%.

Network bandwidth. Network bandwidth is a one to one map between the IoT network and its corresponding blockchain network. It is defined by the the IoT devices downlink and uplink rates. For example IEEE 802.15.4 and NarrowBand-IoT standards set 250 Kbps data transfer peak rates for machine to machine communication, whereas in LTE Cat M1 and LTE Cat 0 standards it is 1 Mbps. In this work, in order to avoid network overloads and consequent bottlenecks with high information traffic, we set an upper bound of 250 Kbps as total of uplink

[3] siliconrepublic.com/machines/iot-devices-botnets-autonomous-cars

[4] In Ethereum blockchain they are included to the blockchain as uncle blocks, however they do not count towards total difficulty of the blockchain [55].

Dimensions	Metrics
Scalability	◇ Maximum no of IoT devices as peers ◇ Maximum transaction throughput
Security	◇ Maximum work in the blockchain
Decentralization	◇ $\frac{\text{90\% block propagation time}}{\text{block generation interval}} \leq 1$
Efficiency	◇ Stale block generation rate $\approx 1\%$
Network bandwidth	◇ Avg network traffic of a device $\leq$ 250 Kbps

Table 7.1: PoW Blockchain - IoT Integration Metrics

and downlink rates.

7.2 PoW Blockchain-IoT Integration Evaluations

In this section, we evaluate the performance of the integration of PoW blockchains in IoT, subject to the dimensions and metrics defined in Section 7.1. To this end, we use and further extend (by adding different device location setups) the Bitcoin simulator[5] presented in [55], (see Section 7.2.1). We perform three evaluations (see Section 7.2.2): one by varying block size and block generation intervals (see Section 7.2.2); one by varying device location (see Section 7.2.2); one by varying the number of IoT devices (see Section 7.2.2). We present results as an average of 5 experimental runs. We use the findings of this section to define the concept of sweet-spot guidelines that drives the generation of sub-blockchains (see Section 7.2.3).

7.2.1 Simulator Setting

The Bitcoin simulator is built on *ns-3* discrete-event network simulator. It allows to model a Bitcoin network with a set of consensus and network parameters such as: block generation interval; block size; number of nodes. Connections between nodes are established using point-to-point channels, by considering latency and bandwidth as the two main characteristics (cfr. [55] for further information). We have extended the simulator with three different device location setups, namely *the Netherlands*, *Europe*, and *World*, by adopting real world network latency data.[6] In the Netherlands setup, devices are located in six cities of the Netherlands: *Alblasserdam, Amsterdam, Dronten, Eindhoven, Rotterdam* and *The Hague*. In the Europe setup, devices are located in six European cities: *Brussels, Athens, Barcelona, Izmir, Lisbon* and *Milan*. Finally, in the World setup devices are located in 7 globally distributed cities: *Dhaka, Hangzhou, Istanbul, Lagos, Melbourne* and *San Diego*. We equally distribute regular and miner among the cities in the respective setups.

[5] github.com/arthurgervais/Bitcoin-Simulator
[6] wondernetwork.com/pings

In order to use the simulator for our evaluations we categorize IoT devices within two roles: miners and regular devices. The number of connections per miner device and regular device follows the distribution as in [87]. Regular devices only check and propagate the blocks they receive, whereas miner devices also generate new blocks. The ratio of miner over number of nodes is set to ca 7%, with the remainder taking the role of regular devices. This is justified by some Bitcoin statistics [87] and by the fact that we consider only a small subset of IoT devices to have enough resources to take part in the mining process.

Network latency plays a critical role in performance due to the intrinsic nature of peer to peer information propagation (i.e., block and transaction). Hence, to evaluate how geographical locations of the devices affect network latency, we exploit *the Netherlands*, *Europe*, and *World* device location settings of the simulator.

Bandwidth capacities of IoT devices obviously affect information propagation time in the blockchain. To have a realistic bandwidth setup, we adopt the bandwidth benchmarks of Raspberry Pi devices.[7] Hence, we adopt an upper bandwidth limit of 100 Mbps (variations within that limit are allowed due to connection type) and we realistically simulate bandwidth capacities with a distribution from testmy.net.[8] That results in a varying download bandwidth between 0.1 Mbps and 100 Mbps with a 5 Mbps average, and a varying upload bandwidth between 0.02 to 20 Mbps with a 1Mbps average.

7.2.2 Evaluation Results

Evaluation I: Block sizes and block generation intervals

We evaluate the effect of block sizes and block generation intervals with the simulator with the Netherlands setup. We adopt a six block generation cycle with the following intervals: 10 minutes, 5 minutes, 1 minute, 30 seconds, 10 seconds, and 5 seconds. For every block generation cycle, we vary block sizes as: 10 KB, 50 KB, 100 KB, 500 KB, 1 MB, 5 MB, 10 MB. We fix the number of IoT devices to 250, with 18 devices with miner roles, according to the 7% ratio in Section 7.2.1. Experiment results are presented in Table 7.2.

Network bandwidth. Not surprisingly, using bigger blocks and/or having short block generation intervals increase the average network traffic. In that, big blocks (e.g., 5 MB) comply with the network bandwidth metric's bound when block generation interval is long enough (e.g., 5m), whereas for small blocks (e.g., 10 KB) even short block generation intervals (e.g., 5s) are suitable.

Security. Obviously, using shorter block generation intervals increases the number of blocks generated. However, we observe that this is not proportional, especially when the block size is bigger than 100 KB. Similarly, in experiments with 1 minute or shorter block generation interval settings, increasing the block size decreases the number of generated blocks. This is due to bandwidth exhaustion of devices. Therefore, according to the bounds of security metric, using small blocks (e.g., 10 KB) in short block generation intervals (e.g., 5s) is more appropriate to increase number of genuine blocks.

[7] pidramble.com/wiki/benchmarks/networking
[8] testmy.net/country

Decentralization. According to the decentralization metric's bounds, 90% block propagation time should be lower than block generation interval. Due to their restricted bandwidth capabilities, IoT devices have to spend more time to propagate big blocks, and that in turn breaches the 90% block propagation time bound. In parallel, we observe that, when using big blocks (e.g., 10 MB), block generation interval should be long enough (e.g., 10m) to satisfy the decentralization lower bound. For example, when small blocks (e.g. 10 KB) are used, the decentralization bound can be satisfied with shorter block generation intervals (e.g., 5s). Therefore, in order to achieve decentralization, block sizes and block generation intervals should be set carefully.

Efficiency, scalability. Short block generation intervals and/or using big blocks leads to higher stale block rates, as bandwidth resources of IoT devices are exhausted in propagating the blocks. In order to achieve low stale block rates, with a short block generation interval setup, only small blocks can be used. Bigger blocks (e.g., 1 MB) can be used with long block generation intervals. The bigger the block is, the longer block generation should be used to satisfy the low stale block generation bound. Moreover, we observe that block sizes bigger than 1 MB are not suitable for IoT, since it leads to high stale block rates, even with a long block generation interval setup. Achieving a low stale block rates positively impacts transaction throughput. In our experiments the highest throughput achieved is 30.1 transaction per second in using 500 KB blocks with 1 minute block generation interval setting with 1.71% stale block rate.

Findings: blocks smaller than 1 MB should be used; block generation intervals should be as short as possible; block size and block generation intervals should be set carefully to ensure low stale block rates and high decentralization.

Evaluation II: Device Locations

We evaluate the effect of device locations by varying the network latency among IoT devices. In order to simulate that, we use the Bitcoin simulator with three location settings (the Netherlands, Europe, and World), as in Section 7.2.1. Since from Evaluation I, the optimal block size should be less than or equal than 1 MB, the block size is fixed at 500 KB on average. We adopt a six block generation cycle with the following intervals: 10 minutes, 5 minutes, 1 minute, 30 seconds, 10 seconds, and 5 seconds. We fix the number of IoT devices to 250, where 18 of them are miners. Experiment results are presented in Table 7.3.

Network bandwidth, security. For all location setups, in each block generation interval setting, average network traffic per device and number of generated genuine blocks are highly correlated. Particularly, only 1 minute or longer block generation intervals comply with the bound for the the network bandwidth metric (the average network traffic should be less than 250 Kbps) for all location settings. Hence, a 1 minute block generation interval is the most suitable according to the security metric bound, since it has the highest number of genuine blocks. With those, every locations setup shows a similar behavior.

Scalability, decentralization, efficiency. For any setup that we tried the outcome with shortest block propagation delays, lowest stale block rates, and highest transaction throughputs is the Netherlands setup. For example, with 1 minute block generation interval, a PoW

Block Size	Block. Gen. Intrvl(s)	Total Blocks	Stale Blocks	Genuine Blocks	Stale Rate	90% Prop. Delay(s)	Avg Traffic (Kbps)	Thrghpt (TX/s)
10 MB	**10m**	10.8	0.43	10.4	3%	360	276	69.3
	5m	18.8	0.9	17.9	8.83%	755	723	119.5
	1m	45.6	16	26.93	35.07%	2162	21215	197.5
	30s	51.2	26.6	24.6	47.99%	2412	49520	164.1
	10s	57.7	41.2	16.5	71.38%	2560	151046	110.2
	5s	64.2	48.2	16	75.00%	2665	273777	107.1
5 MB	**10m**	10.2	0.26	9.9	2.6%	168	134	33.1
	5m	19.9	1	18.9	5.3%	180	288	63
	1m	67.3	12.1	55.2	17.99%	1888	8718	184
	30s	73.4	26.8	46.6	36.52%	2105	28528	155.5
	10s	84	56.5	27.5	67.18%	2472	100255	92
	5s	91.4	68.5	22.9	74.91%	2512	190671	76.4
1 MB	**10m**	12.3	0	12.3	0%	31	26	8.2
	5m	22.3	0	22.3	0%	32	53	14.9
	1m	92.4	3.4	89	3.71%	37	438	59.3
	30s	165.9	8.5	157.4	5.15%	818	3243	104.9
	10s	219.6	94.1	125.5	42.86%	1812	30259	83.7
	5s	232.5	128.7	103.8	55.37%	2183	69059	69.2
500 KB	**10m**	9.6	0	9.6	0%	15	13	3.2
	5m	18.6	0	18.6	0%	15	26	6.2
	1m	92.1	1.6	90.5	1.71%	17	136	30.1
	30s	165.2	9.2	156	5.56%	18	639	52
	10s	346.5	101.4	245.1	29.25%	1665	14762	81.7
	5s	350.6	161.1	189.5	45.96%	1972	41378	63.2
100 KB	**10m**	9.3	0	9.3	0%	3.2	2	0.6
	5m	23	0	23	0%	3.2	5	1.5
	1m	99	0	99	0%	3.2	27	6.6
	30s	186.4	4.4	182	2.35%	3.2	54	12.1
	10s	537.3	22.8	514.5	4.25%	3.4	447	34.3
	5s	954.5	124.5	830	13.04%	99	7249	55.3
50 KB	**10m**	11	0	11	0%	1.6	1	0.4
	5m	18.6	0	18.6	0%	1.6	2	0.6
	1m	96.3	0	96.3	0%	1.6	14	3.2
	30s	187.0	0.7	186.3	0.35%	1.6	28	6.2
	10s	562.0	10.2	551.8	1.82%	1.7	84	18.4
	5s	1120.4	43.4	1077	3.87%	1.8	931	35.9
10 KB	**10m**	10.3	0	10.3	0%	0.4	0.3	0.1
	5m	21.6	0	21.6	0%	0.4	0.7	0.2
	1m	101	0	101	0%	0.4	3.5	0.7
	30s	193.3	0	193.3	0%	0.4	7	1.3
	10s	598.6	0	598.6	0%	0.4	21	4
	5s	1166.3	19.9	1146.4	1.71%	0.4	42	7.6

Table 7.2: Evaluation I: Block sizes and block generation intervals

Block Gen. Intrvl(s)	Sce.	Total Blocks	Stale Blocks	Genuine Blocks	Stale Rate	Mean Delay (s)	Avg Traffic (Kbps)	Thrghpt (TX/s)
10m	**N**	9.6	0	9.6	0%	7	13	3.2
	E	9.9	0	9.9	0%	16	13	3.3
	W	9.9	0	9.9	0%	20	13	3.3
5m	**N**	18.6	0	18.6	0%	6	26	6.2
	E	16.4	0	16.5	0%	13	27	5.5
	W	15.5	0	15.5	0%	17	27	5.2
1m	**N**	92.1	1.6	90.5	1.71%	17	136	30.1
	E	93.6	4.8	88.8	5.14%	18	140	29.6
	W	96.5	7.9	88.6	8.22%	20	140	29.5
30s	**N**	165.2	9.2	156	5.56%	18	639	52
	E	170.5	21.9	148.6	12.87%	38	527	49.5
	W	169.5	23.9	145.6	14.11%	52	592	48.5
10s	**N**	346.6	101.4	245.2	29.25%	314	14762	81.7
	E	314	104.6	209.4	33.31%	355	15237	69.8
	W	331	136.8	194.2	41.34%	392	16777	64.7
5s	**N**	350.7	161.2	189.5	45.96%	815	41378	63.2
	E	301.2	156	145.2	51.80%	918	43931	48.4
	W	303.3	161.1	142.2	53.12%	1000	45021	47.4

Table 7.3: Evaluation II: Device Locations

blockchain using the Netherlands setup achieves a throughput of 30.1 per second and complies with the efficiency bound (stale block rate is 1.71%) and decentralization bound (90% block propagation time is 17 seconds). Whereas, in Europe and World settings, the block generation interval needs to be at least 5 minutes to satisfy the same bounds. With those, both the Europe and World setups can only achieve a throughput of 5 transaction per second.

Findings: blockchains containing IoT devices that are geographically close to each other achieve higher throughput with low stale block rates.

Evaluation III: Number of IoT devices

We evaluate the effect of varying the number of IoT devices with two experiment types: experiment (A): we fix the PoW difficulty; experiment (B): we fix block generation interval. Each setup is run for 100 minutes. In both types we vary the number of IoT devices from 83 to 1250 and assumed a fixed block size of 500 KB. In PoW blockchains, the block generation interval depends on the ratio of the difficulty of the PoW puzzle over the total mining power of the system [92]. Hence, with the PoW difficulty fixed, we vary the block generation intervals inversely proportionally to the number of miners. On the other hand, with a fixed block generation interval of 1 minute, the difficulty of the PoW puzzle is varied proportionally to number of

No of Miners/ Total	Block Gen. Intrvl(s)	Total Blocks	Stale Blocks	Genuine Blocks	Stale Rate	90% Delay (s)	Avg Traffic (Kbps)	Thrghpt (TX/s)
6/83	**3m**	29.9	0	29.9	0%	7	44	9.9
12/166	**1.5m**	76.7	1.3	75.4	1.9%	8	88	25.1
18/250	**1m**	92.1	1.6	90.5	1.71%	17	136	30.1
36/500	**30s**	177.7	15.6	162.1	8.8%	41	1168	54
54/750	**20s**	237.2	32.9	204.3	13.87%	147	3485	68.1
72/1000	**15s**	216.5	39.4	177.1	18.2%	291	5791	59
90/1250	**12s**	212	47.5	164.5	22.43%	498	8265	54.8

Table 7.4: Evaluation III: Number of IoT Devices - Experiment (A): Fixed difficulty setting

miners (the difficulty of the PoW puzzle is α for 6 miners and 15α for 90 miners).

Experiment (A). Results are in Table 7.4.

Metrics. Having more IoT devices with shorter block generation intervals leads to generate more blocks, leading to an increase in throughput and average network traffic per device. That causes extensive bandwidth consumption that generates long block propagation delays, which leads to high stale block rates. Hence, experimental variations containing 83, 166 and 250 devices satisfy the efficiency, network bandwidth, and decentralization bounds. When it comes to scalability and security bounds, scenario containing 250 devices is the optimal setups as it produces more genuine blocks, and achieves the highest throughput and scales to more devices.

Experiment (B). Results are in Table 7.5.

Metrics. In all the experimental variations, 90% block propagation times are less than 1 minute block generation interval, thus satisfying the decentralization bound. Similarly, average network traffic per device is less than 250 Kbps, satisfying the network bandwidth bounds for all experimental variations. However, only experimental variations containing 83, 166 and 250 devices satisfy the efficiency bound with low stale block rates. Among them, experiment containing 250 devices is the optimal setup according to the security and scalability bounds, as it achieves the highest throughput and scales to more devices.

Findings: PoW blockchains containing few hundreds of IoT devices achieve higher transaction throughput; the optimal number of IoT devices as blockchain peers is around 250.

7.2.3 Sweet-spot Guidelines

After Evaluations I,II and III, we can conclude that PoW blockchains containing few hundreds of IoT devices in close geographical proximity achieve the highest performance. Therefore, in order to design a blockchain architecture for IoT, we propose to deploy multiple PoW blockchains as sub-blockchains for IoT, organized according to pools of IoT devices. We adopt the following guidelines, referred to as sweet-spot:

- Sub-blockchains should contain few hundreds of IoT devices.
- Sub-blockchains should contain IoT devices that are geographically close and frequently

No of Miners/ Total	Total Blocks	PoW Puzzle Difficulty	Stale Blocks	Genuine Blocks	Stale Rate	90% Delay (s)	Avg Traffic (Kbps)	Thrghpt (TX/s)
6/83	96.1	α	0,8	95.3	0.85%	13	135.76	31.7
12/166	96.3	2α	1.8	94.5	1.19%	14	133.37	31.1
18/250	92.1	3α	1.6	90.5	1.71%	17	136	30.1
36/500	93.03	6α	3.99	89.04	4.29%	26	122.99	32.86
54/750	93.41	9α	4.49	88.92	4.8%	28	102.03	29.64
72/1000	93.03	12α	4.42	88.61	4.75%	28	102.99	29.53
90/1250	92.77	15α	4.98	87.78	5.36%	39	107.01	29.26

Table 7.5: Evaluation III: Number of IoT Devices - Experiment (B): Fixed interval setting

communicating with each other.

• Block size and block generation intervals should be set to ensure low stale block rates, and high decentralization and scattering of mining power.

• Blocks smaller than or equal to 1 MB should be used.

• Block generation interval should be as short as possible.

In the next section, we design the architecture of Hybrid-IoT, based on the sweet spot guidelines, by leveraging on multiple PoW sub-blockchains.

7.3 Hybrid-IoT: Hybrid Blockchain Architecture for IoT

Hybrid-IoT consists of multiple PoW sub-blockchains that achieve distributed consensus among IoT devices that are peers on the blockchain. Sub-blockchains are generated according to the sweet-spot guidelines defined in Section 7.2.3. In order to connect the sub-blockchains, Hybrid-IoT uses a BFT inter-connector framework (e.g., Polkadot and Cosmos) that guarantees inter-blockchain transactions.

System execution. The transaction flow in Hybrid-IoT is as follows: transactions on the PoW sub-blockchains are processed and included in blocks that are added to their respective sub-blockchain upon PoW consensus; when a transaction among two distinct sub-blockchains happens, that is picked by the BFT inter-connector framework; the BFT inter-connector framework checks the transaction correctness and authenticity; after a positive response, the BFT inter-connector framework transfers the transaction to the target sub-blockchain's transaction pools that hold unprocessed transactions; last, the transaction is processed and included in a newly generated block in the respective sub-blockchain, upon PoW consensus. The reasons for the choice of a BFT inter-connector framework lies in the intrinsic capability of BFT consensus protocols to achieve high throughput with a low number of peers.[9] Hence, that should allow

[9]For example, Tendermint protocol used by Cosmos network is able to process thousands of transactions per second [26], whereas PoW sub-blockchains are able to process few dozens of transactions according to our evaluations presented in Section 7.2.

to connect few sub-blockchains with an adequate throughput for inter-blockchain transactions. Moreover, by maintaining low latency in the transmission of inter-blockchain transactions, the BFT inter-connector framework allows the connection of a new sub-blockchain without deferring application execution. An example of Hybrid-IoT architecture containing two sub-blockchains is shown in Figure 7.1.

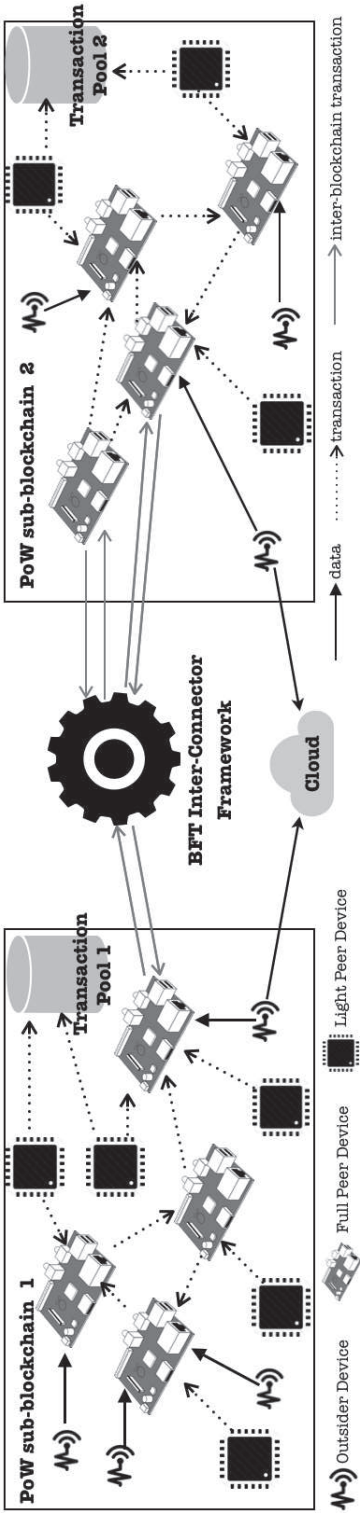

Figure 7.1: Hybrid-IoT

Consensus participation. Blockchains can be categorized into two groups subject to the type of peer access control: *permissioned* and *permissionless* blockchains [147]. In permissionless blockchains, all peers can take part in the consensus, whereas in permissioned blockchains, only pre-defined peers can take part in the consensus process. Hybrid-IoT is a permissioned blockchain system. This is particularly important since the sub-blockchains are based on PoW and the nature of IoT devices can easily lead to malicious cases of majority attack [69]. Indeed, specialized mining hardware could be easily masked as an IoT device and gain enough block mining power to control the PoW blockchain.[10]

Security. While the permissioned nature of Hybrid-IoT can mitigate the risk of longer blockchain attacks [147], IoT is still prone to those, since device capture and device cloning attacks are not a rare occurrence [108]. Usually that would be mitigated by the difficulty of the PoW puzzle. Indeed, in PoW blockchains, for a fixed block generation interval, the difficulty of the PoW puzzle is set proportionally to the total mining power [92]. While the Hybrid-IoT PoW sub-blockchains would have PoW puzzles with low difficulty (there could be only a relatively small number of IoT devices that mine), by keeping a high block generation rate, security vulnerabilities can be prevented (see Section 7.5).

Anomaly resilience. An important issue to consider is the so called blockchain anomaly, presented in [94]: "a long enough delay on the delivery of messages could lead to having the miners to seemingly agree separately on different branches containing more than k blocks each, for any $k \in \mathbb{N}$". While this is theoretically possible, in practice, in a decentralized blockchain, like Bitcoin, it has never materialized in more than few blocks for many years.[11] Hence, in order to prevent the anomaly in Hybrid-IoT, we adopt the same degree of decentralization and scattering of mining power as in Bitcoin. That is assured by the sweet-spot guidelines and it can be further reinforced by the findings in [40].

Remediations. Unlike specialized PoW mining hardwares for cryptocurrencies, such as ASICs and GPUs, IoT devices have limited hardware resources and they are widely energy-constrained devices [42]. As such, IoT devices do not have enough hardware or energy resources to solve very complex PoW puzzles.[12] In Hybrid-IoT, the difficulty of the PoW puzzle is set according to the hardware constraints of IoT devices. Therefore, IoT devices can still perform their application specific tasks, such as data processing, while concurrently continue to mine blocks.

Roles of IoT devices. IoT devices have heterogeneous capabilities, and their roles should reflect their capabilities. Therefore, in Hybrid-IoT, we define three different roles for IoT devices as peers on the blockchain: *full peer roles*; *light peer roles*; and *outsider roles*.

• *Full peer role.* IoT devices that have enough capacity and computing power to perform complex operations, like a Raspberry Pi 3, take the *full peer* role. They have high resources and run full-fledged operating systems like Raspbian. Hence, as peers on the blockchain, they mine blocks and take part in the consensus process in the PoW sub-blockchains. In addition

[10] In the example case of Bitcoin, Raspberry Pi has 0.2 MH/s mining power, whereas a specialized mining device AntMiner S has 14 TH/s mining power.

[11] We crawled orphan blocks through all of the Bitcoin orphan blocks presented in `blockchain.info/orphaned-blocks`.

[12] As of late 2017, it would require more than 1000 years for a Raspberry Pi to mine a single block in Bitcoin.

to that, full peer devices act as gateway devices to connect set of light peer devices to the blockchain network, referred as full peer device subnet. Hence, blocks formed by a full peer device contain its own transactions and transactions sent by its device subnet. The number of light peer devices in the full peers' device subnet is set according to its mining power to guarantee fair block generation rates.

• *Light peer role.* IoT devices that have limited capabilities and computing power, such as Arduino Yun, take the *light peer* role. They have basic operating systems like Alpine Linux, and can connect and participate in the blockchain by performing simple tasks, such as sending transactions. Light peer devices send transactions to the blockchain transaction pool and to the full peer that acts as a gateway. This allows all the full peers to be aware of all the transactions in the sub-blockchain. This acts as a double-check in case a full peer is subject to a malicious attack.

• *Outsider role.* IoT devices that have very limited capabilities by being able only to act as basic sensors, take the *outsider* role. They are not peers on the blockchain, but they can connect to full peers for further data fusion (such as data aggregation). Raw data generated by an outsider is not stored the blockchain to prevent data overload.

7.4 Performance Evaluation

In Hybrid-IoT, as per Section 7.3, light peer devices send transactions to full peer devices, and those will include them in the newly generated blocks. Hence, full peers need to process an heavy transactions load. Therefore, the first performance test for Hybrid-IoT is a stress test, in which, set of light peers repeatedly sends transactions to full peers (see Section 7.4.1). Then, we shift the focus to the different sized PoW sub-blockchains, where full peers take part in the consensus process. Sub-blockchains can be generated with different number of full peers, which affects the time required to achieve consensus and the way in which the full peer manages its resources. Hence, the second type of performance test is done by varying sub-blockchain sizes and measuring the time needed to achieve consensus and the full peers' resource usage (see Section 7.4.2).

7.4.1 Performance Evaluation I: Stress test

We design a DDoS attack simulation for the stress test: 20 light peers take the role of attackers; a full peer takes the role of victim; the attack is conducted for 45 minutes. All the peers are are virtualized with LXC (Linux Containers)[13] containers and have the following configurations:

• Full peer: Ubuntu 14.04 (Trusty) O.S; 512 MB RAM memory, 10% of one Intel Core i7 2.70 GHz CPU; 5 mbit/s ingress and egress network interface limit; bitcoind version 14.02 Bitcoin protocol's full node. We measure CPU utilization, memory usage, and Ethernet activity with nmon.[14]

[13] linuxcontainers.org
[14] nmon.sourceforge.net

• Light peer: Alpine Linux 3.6.0 O.S; 128 MB RAM memory, 2% of one Intel Core i7 2.70 GHz CPU, 1 mbit/s ingress and egress network interface limit; Java SE; JRE 8 update 131 environment; Bitcoin protocol's thin client model developed with bitcoinj library. We monitor CPU usage, memory usage, and Ethernet activity with RRDtool.[15]
We use Bitcoin regtest[16] (regression test) network to execute the stress test. The DDOS attack is executed as follows: a number of attackers, max 20, generate a load of identical and valid transactions of ca 225 bytes at varying frequency (from ca 2tx/s to ca 9tx/s); once the victim receives the load it checks the transactions validity add them to the transaction pool. A load of 108960 transactions was generated with an average of 5448 transactions per attacker. For the sake of brevity, in the figures, we show measurements only for the first 15 minutes and only for the CPU component (the other components have very similar trends).

Victim results. Figure 7.3[17] shows the CPU usage of the victim: 90% of its CPU is exhausted by processing the attackers' load; a similar measurement and graph is observed for its Ethernet activity; memory usage is steady around 150 MBs. The victim manages to receive and process over 40 transactions per second from 20 attackers. We can conclude that the victim successfully manages to perform its blockchain duties without crashing or halting under the heavy load from the attack (here heavy is attributed to the fact that the attackers' resources are exhausted).

Attacker results. Figure 7.2 shows the CPU usage of one of the attackers: nearly 100% of the attacker's CPU is exhausted (it is capable of processing ca 300 bit/s); there is an increase in memory usage from 99 MB (before starting to attack) to 124 MB (during attack), utilizing all of its memory. When one light peer takes the role of attacker, it manages a maximum of 9 transactions per second without crashing. This should help to characterize the capabilities of a light peer to generate transaction loads, regardless of the DDoS simulation performed.

7.4.2 Performance evaluation II: Sub-blockchain size

In order to measure the sensitivity to sub-blockchain sizes, we design four sub-blockchain emulation scenarios (Emulation I,II,III and IV) by varying the number of full peers in the sub-blockchains. In Emulation I the number of full peers is 20, in Emulation II 40, in Emulation III 100, and in Emulation IV 200. Peers are connected to each other in a round-robin way. All the full peers are virtualized with LXC (Linux Containers)[18] containers on an IBM Power 8 server[19] and have the following configurations:

• Full peer: Ubuntu 14.04 (Trusty) O.S.; 512 MB RAM memory; 5% of a single Power8 3.5 GHz CPU; 5 mbit/s ingress and egress network interface limit; bitcoind version 14.02 Bitcoin protocol's full node. We measure CPU utilization, memory usage, and Ethernet activity (traffic

[15] oss.oetiker.ch/rrdtool
[16] bitcoin.org/en/glossary/regression-test-mode
[17] Legend for Figure 2; user: avg CPU utilization for Bitcoin client; system: avg CPU utilization for kernel mode; wait: avg CPU utilization for I/O wait mode.
[18] linuxcontainers.org
[19] `ibm.com/systems/power/hardware/reports/factsfeatures.html`

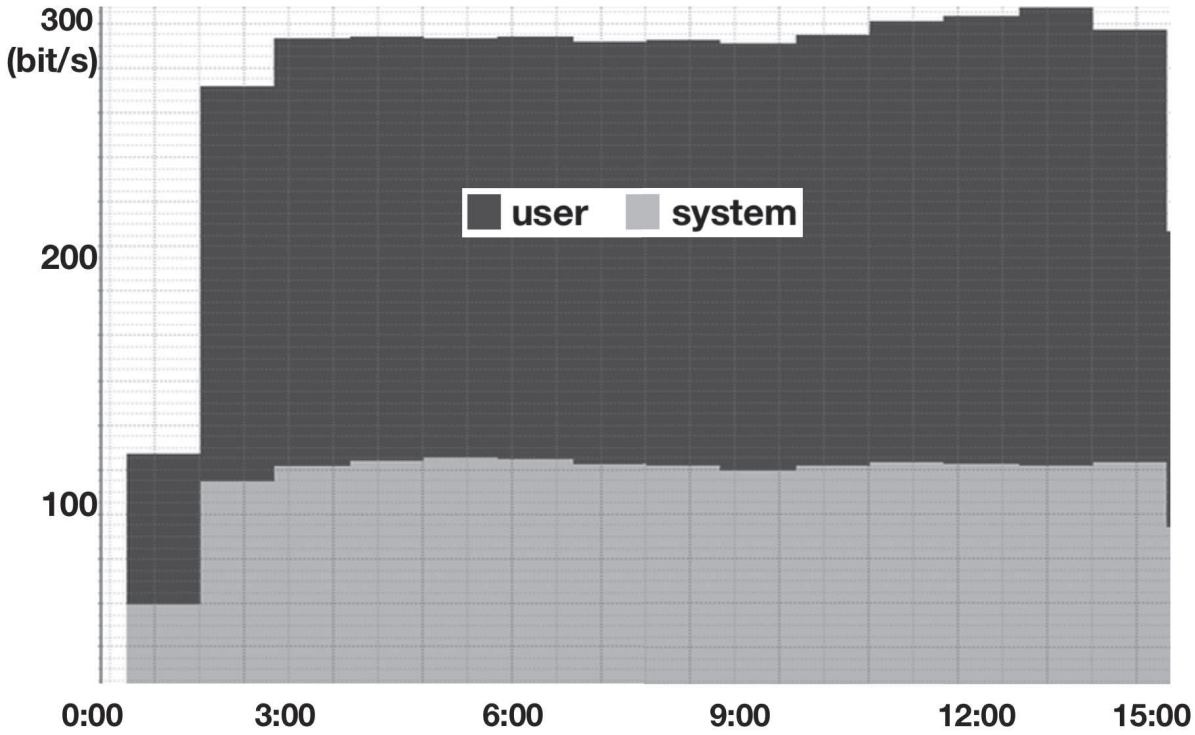

Figure 7.2: CPU utilization of light peer devices in Performance Evaluation I

and packets) with nmon.[20]

We use Bitcoin regtest (regression test) network to execute the emulations. The emulations are executed as follows: in each emulation we submit 11.000 identical transactions (225 bytes) to the network with one full peer; one peer submits to the remaining full peers the 5 blocks with 1 minute block generation interval; the full peers achieve consensus on the submitted blocks. We measure resources utilization at the last peer of the round-robin from the moment at which the submitting peer proposes the first block of the five, till the moment in which all the five blocks are recorded on the local blockchain copy of the last round-robin peer (we refer this as the consensus cycle in the rest of the chapter). We note that we do not employ light peers to generate loads. This is justified by the need of measuring consensus with heavy transactions loads. All measurements are in Table 7.6 or in the text below.

Results. We observe that the consensus cycle is longer with sub-blockchains with more full peers as block and transaction propagation takes longer. We show in Table 7.6, that on average, emulation scenarios with more full peers use less resources. This is because, with sub-blockchains with more full peers, resource utilization is averaged over longer consensus cycles.

[20]nmon.sourceforge.net

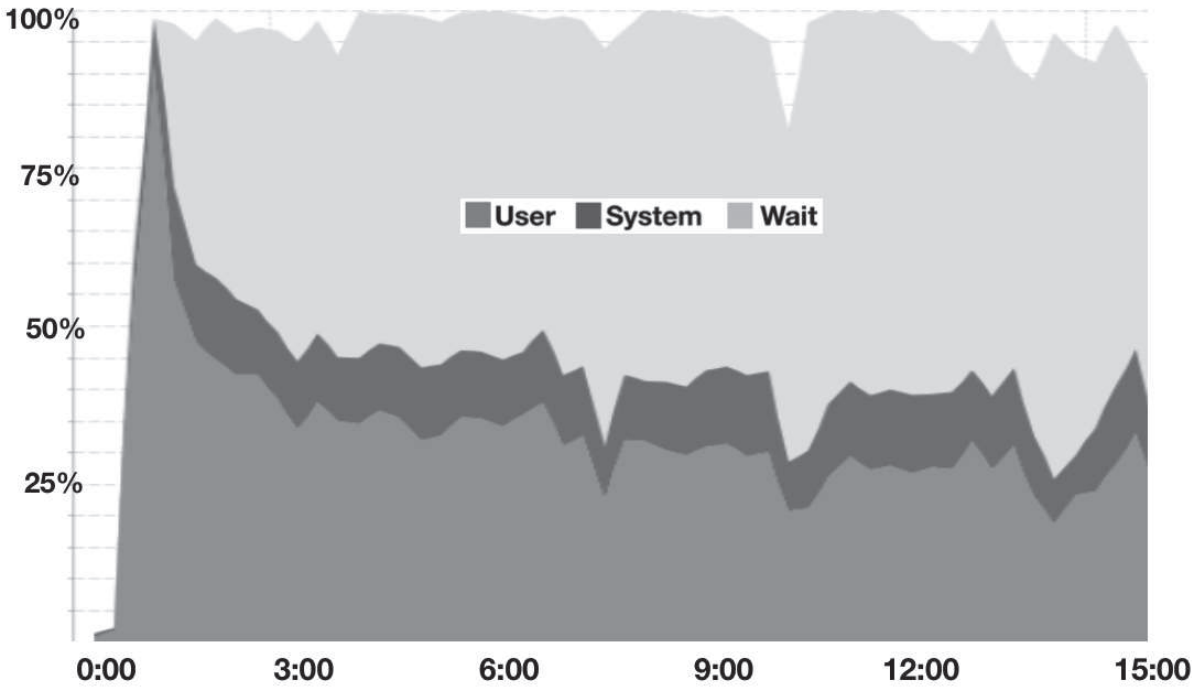

Figure 7.3: CPU utilization of full peer devices in Performance Evaluation I

Metrics	20 peers	40 peers	100 peers	200 peers
Avg CPU usage	6.7%	5.2%	2.8%	2.1%
Avg Memory usage	115 MB	109 MB	109.1 MB	108.7 MB
Avg Ethernet traffic	7.9 KB/s	6.2 KB/s	3.6 KB/s	1.6 KB/s
Avg Ethernet packets	9.4/s	8.5/s	5/s	3.6/s

Table 7.6: Perf Eva II: Performance Statistics

7.5 Security Evaluation

Despite having low difficulty puzzles, sub-blockchains can prevent security vulnerabilities with high block generation, as noted in Section 7.3. We evaluate this by simulating a set of scenarios in which six sub-blockchain setups are compared. This is done with the help of the Bitcoin simulator (as in Section 7.2.3) and by measuring their total work. Total work is defined as the multiplication of the number of genuine blocks by the PoW puzzle difficulty. The sub-blockchains are generated according to sweet-spot guidelines contain subgroups of IoT devices that are geographically close to each other. Hence, we generate the following scenarios:

- Scenario I: 83 peers of which 6 full peers.
- Scenario II: 166 peers of which 12 full peers.

Block Size	Simulated Scenario	PoW Puzzle Difficulty	Block Interval	Genuine Blocks	Stale Rate	Total PoW
	Scenario I	α	30s	198.6	1.2%	198.6α
	Scenario II	2α	35s	162.6	1.56%	325.2 α
100 KB	Scenario III	3α	40s	133.2	1.7%	299.6α
	Scenario IV	6α	6m	15.2	1.49%	91α
	Scenario V	12α	7m	13.6	1.99%	163α
	Scenario VI	24α	10m	9.2	1.9%	178α
	Scenario I	α	50s	98	1.95%	98α
	Scenario II	2α	55s	74.1	1.2%	148.2α
500 KB	Scenario III	3α	1m	90.5	1.71%	271α
	Scenario IV	6α	10m	10.4	1.98%	62α
	Scenario V	12α	11m	9.7	1.89%	117α
	Scenario VI	24α	12m	8.4	1.96%	203α
	Scenario I	α	150s	37.5	1.3%	37.5α
	Scenario II	2α	165s	34.6	1.2%	69.2 α
1 MB	Scenario III	3α	3m	26.6	0%	79.8α
	Scenario IV	6α	10m	10.5	0.5%	63α
	Scenario V	12α	12m	8.3	0%	99α
	Scenario VI	24α	13m	5.9	1.8%	141α

Table 7.7: Security Experiments' Results

- Scenario III: 250 peers of which 18 full peers.

The scenarios above are generated using the Netherlands setup. We also generate 3 more scenarios using the World setup to have a baseline:

- Scenario IV: 500 peers of which 36 full peers.
- Scenario V: 1.000 peers of which 72 full peers.
- Scenario VI: 2.000 peers of which 144 full peers.

We vary the difficulty of the PoW puzzle proportionally to the number of full peers (see Table 7.7). We also assume that all the peers have the same resources. In order to evaluate the sensitivity to block size we vary the block size with values 100KB, 500KB and 1MB. We configure every scenario with the shortest block generation interval, in order to be compliant with the bounds of the metrics defined in Section 7.1. We present the results in Table 7.7.

Results. As expected, Scenarios I, II and III are able to comply with the bounds of blockchain-IoT integration metrics with shorter block generation intervals than Scenarios IV, V, and VI, and thus they produce more genuine blocks. This trend is more prominent with small block size settings. In fact, with 100 KB blocks, total work of Scenario I sub-blockchain

is more than the total work of Scenario VI sub-blockchain, the latter with a twenty-four times more difficult PoW puzzles than the former. Whereas, with 1 MB blocks, due to a simulated limited bandwidth (inherited from the need to replicate low bandwidth IoT), block generation intervals are longer. With 1 MB block size, total work of Scenario II sub-blockchain is more than the Scenario IV sub-blockchain, the latter with six times more difficult PoW puzzles than the former. Hence, we observe that with smaller blocks we can generate sub-blockchains with less full peers without sacrificing security. We finally observe that, even with low difficulty PoW puzzles, sub-blockchains generated according to the sweet-spot guidelines are able to have more or comparable total work than sub-blockchains with high difficulty PoW puzzles that do not adhere to those guidelines (the Netherlands scenarios have more work with easier PoW puzzles than World scenarios with more difficult PoW puzzles).

Chapter 8

Conclusions

Despite being a disruptive technology that has been applied to several domains, many challenges facing IoT. In this thesis, we address two of these challenges, namely we focused to enhance individuals privacy and improve security of IoT systems under the decentralized model. Towards achieving this goal, first we investigated requirements that needs to be addressed. Then, we proposed frameworks, models and architectures. We used different tools, programming languages, technologies, and, best practices and ideas from different research domains. Significantly, we extensively used blockchain technology to decentralize and improve security of IoT systems. In the following, first we give a short summary of the work that has been presented in this thesis, then we discuss the future work, and finally we give the concluding remarks.

8.1 Summary

We present summary of the work that has been done in this thesis chapter by chapter:

- In Chapter 4, we presented a user-centric privacy enforcement framework as a first step to enhance user control on personal data usage in IoT platforms. The framework gives to the users more control on how their data can be combined and what cannot be automatically inferred from them by analytics processes operating within an IoT platform. The framework has been designed to operate within a reference IoT platform with a general architecture that fits numerous existing IoT systems. The experimental evaluation of the performance has shown a negligible overhead in all considered scenarios.
- In Chapter 5, we presented an extension of the above mentioned core framework for decentralized privacy enforcement for IoT smart objects. The main problem that this framework tackles is privacy enforcement without a central entity.In this framework compliance check of user individual privacy preferences is performed directly by smart objects. Acknowledging that embedding the enforcement mechanism into smart objects implies some overhead, we extensively tested the proposed framework on different scenarios, and the obtained results showed the feasibility of our approach.

• In Chapter 6, we proposed *AutoBotCatcher*, a blockchain based P2P botnet detection framework for IoT. AutoBotCatcher's design was driven by the consideration that bots of the same botnet frequently communicate with each other and form communities. AutoBotCatcher performs community detection on network flows of IoT devices. IoT gateway devices become peers of a BFT blockchain, and, device vendors and security regulators take the block generator role and participate in the consensus process. A BFT Blockchain is exploited to perform dynamic and collaborative network-based botnet community detection on snapshots of the mutual contact graph of IoT devices.

• In Chapter 7, we presented Hybrid-IoT, a novel hybrid blockchain architecture for IoT. In Hybrid-IoT, subgroups of IoT devices becomes peers on PoW sub-blockchains, connected with a BFT inter-connector framework, such as Polkadot or Cosmos. In this chapter, we focused and analyzed the design of the PoW sub-blockchains by defining a set of PoW blockchain-IoT integration metrics that translates IoT issues to blockchain dimensions. The performance evaluation proved the validity of the PoW sub-blockchain design under the defined sweet-spot guidelines. Furthermore, we demonstrated that the sweet-spot guidelines also prevent security vulnerabilities.

8.2 Future Works

Despite all the efforts in the literature and the results presented in this thesis, there are still important concerns on ensuring data privacy and security of IoT systems. Thus, as a future work, we are planning to address those concerns as follows:

• **Future work on data privacy.** We plan to extend the decentralized privacy enforcement framework presented in Chapter 5 according to several directions. First, we plan to extend the privacy model of the proposed framework so as to support multi-dimensional data attributes. Moreover, we recall that the proposed decentralized privacy enforcement mechanism assumes the existence of a set of "derivation paths", that is, a set of inference rules able to predict the data category resulting from the execution of an operator. As a second future work, we aim at investigating how to derive these rules. At this purpose, we plan to exploit existing data fusion schemes proposed for IoT domain [148, 3], where different methodologies have been designed (e.g., probability based, artificial intelligence based) aiming at combining data from multiple sensors to produce more accurate, more complete, and more dependable information that could not be possible to achieve through a single sensor [58]. These methods could be deployed in our framework to model the data flow, and thus derivation paths. Specifically, multi-dimensional data fusion algorithms, such as [3, 148], could be used to extract association rules from data generated by subnet of sensing SOs. Third, we plan to implement a hybrid approach for compliance check, that combines apriori and aposteriori compliance verification, to decrease the performance overhead of the system. We also plan to

improve usability of the system. Indeed, we are aware that an average user of an IoT application may not be skilled enough to understand potential privacy threats and to properly set up his/her privacy preferences. At this purpose, we plan to investigate tools to help users in setting their privacy preferences, based on learning from their habits, as it has been previously done in other domains (e.g., Online Social Networks [51]).

• **Future work on data security.** We plan to extend the blockchain-based P2P botnet detection framework presented in Chapter 6 in many directions. First, we plan to implement AutoBotCatcher by leveraging on BFT based Hyperledger blockchain and exploiting its smart contracts to generate mutual contacts graphs and to manage state changes. Second, we aim to integrate secure multi-party computation platforms, such as Enigma [150], in order to protect privacy of the collaborating parties, while ensuring correct botnet detection. Third we plan to test AutoBotCatcher with several botnet examples, and, extend the current botnet detection model with methods making use of statistical network traffic features. Last direction in future work is to design and implement a trust management module between agents and block generators, in order to make AutoBotCatcher resilient against insider threats.

• **Future work on IoT decentralization.** We consider that integration of blockchain technology to IoT is crucial, as achieving distributed consensus with a blockchain helps to decentralize IoT. Therefore, we plan to extend the hybrid blockchain architecture for IoT proposed in Chapter 7 according to several directions. First, we plan to analyze and stress data volumes in Hybrid-IoT; identify a BFT interconnector framework to test the current design. Second, we plan to work on proving the correctness of Hybrid-IoT design with properly done security proofs. Third, we plan to implement a crash fault tolerant algorithm for the light peers, to address the issue of a full peer subnet losing the connection to light peers in its subnet (due to several reasons, malicious or not). Finally, we plan to analyze the energy footprint of PoW Hybrid-IoT and design a PoW algorithm that is IoT energy friendly.

8.3 Concluding Remarks

As a researcher and enthusiast of IoT, I believe IoT is a disruptive technology and it will have a very bright future, only if, we as researchers, can properly solve the problems facing IoT. If we achieve this properly, IoT will be very beneficial for humankind: it will provide us autonomous systems, smarter environments, better health and energy monitoring and more profitable businesses.

The last but not the least, it is my hope and belief that with this thesis, I made my own humble contribution towards advancing IoT, particularly by addressing its data privacy and security problems under the decentralized model.

Bibliography

[1] Mohammad Aazam, Marc St-Hilaire, Chung-Horng Lung, and Ioannis Lambadaris. Pre-fog: Iot trace based probabilistic resource estimation at fog. In *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*, pages 12–17. IEEE, 2016.

[2] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Taieb Znati. Cloud of things for sensing as a service: sensing resource discovery and virtualization. In *Global Communications Conference (GLOBECOM), 2015 IEEE*, pages 1–7. IEEE, 2015.

[3] Awais Ahmad, Anand Paul, Mazhar Rathore, and Hangbae Chang. An efficient multidimensional big data fusion approach in machine-to-machine communication. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(2):39, 2016.

[4] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[5] Olivier Alphand, Michele Amoretti, Timothy Claeys, Simone Dall'Asta, Andrzej Duda, Gianluigi Ferrari, Franck Rousseau, Bernard Tourancheau, Luca Veltri, and Francesco Zanichelli. Iotchain: A blockchain security architecture for the internet of things. In *IEEE Wireless Communications and Networking Conference*, 2018.

[6] Ross J Anderson. *Security engineering: a guide to building dependable distributed systems*. John Wiley & Sons, 2010.

[7] Elli Androulaki et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. *arXiv preprint arXiv:1801.10228*, 2018.

[8] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *USENIX Security Symposium*, 2017.

[9] O. Arias, J. Wurm, K. Hoang, and Y. Jin. Privacy and security in internet of things and wearable devices. *Multi-Scale Computing Systems, IEEE Transactions on*, 1(2):99–109, April 2015.

[10] Orlando Arias, Jacob Wurm, Khoa Hoang, and Yier Jin. Privacy and security in internet of things and wearable devices. *IEEE Transactions on Multi-Scale Computing Systems*, 1(2):99–109, 2015.

[11] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[12] Thomas Aynaud and Jean-Loup Guillaume. Static community detection algorithms for evolving networks. In *Modeling and optimization in mobile, ad hoc and wireless networks (WiOpt), 2010 proceedings of the 8th international symposium on*, pages 513–519. IEEE, 2010.

[13] 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. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 2014.

[14] Adam Back et al. Hashcash-a denial of service counter-measure, 2002.

[15] Arshdeep Bahga and Vijay K Madisetti. Blockchain platform for industrial internet of things. *J. Softw. Eng. Appl*, 9(10):533, 2016.

[16] Ezedine Barka, Sujith Samuel Mathew, and Yacine Atif. Securing the web of things with role-based access control. In *International Conference on Codes, Cryptology, and Information Security*, pages 14–26. Springer, 2015.

[17] Scott Beach, Richard Schulz, Julie Downs, Judith Matthews, Bruce Barron, and Katherine Seelman. Disability, age, and informational privacy attitudes in quality of life technology applications: Results from a national web survey. *ACM Transactions on Accessible Computing (TACCESS)*, 2(1), 2009.

[18] Timothy J Berners-Lee and Robert Cailliau. World-wide web. 1992.

[19] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[20] Piero Bonatti, Sabrina De Capitani di Vimercati, and Pierangela Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 5(1):1–35, 2002.

[21] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[22] Carsten Bormann, Angelo P Castellani, and Zach Shelby. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62–67, 2012.

[23] Gerben Broenink, Jaap-Henk Hoepman, Christian van't Hof, Rob Van Kranenburg, David Smits, and Tijmen Wisman. The privacy coach: Supporting customer privacy in the internet of things. *arXiv preprint arXiv:1001.4459*, 2010.

[24] Jordan Brown and Douglas M Blough. Distributed enforcement of sticky policies with flexible trust. In *2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS)*, pages 1202–1209. IEEE, 2015.

[25] Glenn Bruns, Daniel S Dantas, and Michael Huth. A simple and expressive semantic framework for policy composition in access control. In *Proceedings of the 2007 ACM workshop on Formal methods in security engineering*, pages 12–21. ACM, 2007.

[26] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, 2016.

[27] Ji-Won Byun, Elisa Bertino, and Ninghui Li. Purpose based access control of complex data for privacy protection. In *Proceedings of the tenth ACM symposium on Access control models and technologies*. ACM, 2005.

[28] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[29] Barbara Carminati, Elena Ferrari, Jianneng Cao, and Kian Lee Tan. A framework to enforce access control over data streams. *ACM Transactions on Information and System Security (TISSEC)*, 13(3):28, 2010.

[30] Barbara Carminati, Elena Ferrari, and Andrea Perego. Enforcing access control in web-based social networks. *ACM Transactions on Information and System Security (TISSEC)*, 13(1):6, 2009.

[31] F Carrez, M Bauer, M Boussard, N Bui, F Carrez, C Jardak, JD Loof, C Magerkurth, S Meissner, A Nettstrater, et al. Deliverable d1. 5–final architectural reference model for the iot v3. 0. *EU Project Internet of Things–Architecture*, 2013.

[32] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[33] Alberto Huertas Celdrán, Félix J García Clemente, Manuel Gil Pérez, and Gregorio Martínez Pérez. Secoman: A semantic-aware policy framework for developing privacy-preserving and context-aware smart applications. *IEEE Systems Journal*, 2016.

[34] C. Y. Chen, J. H. Fu, T. Sung, P. F. Wang, E. Jou, and M. W. Feng. Complex event processing for the internet of things and its applications. In *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*, pages 1144–1149, Aug 2014.

[35] P. Colombo and E. Ferrari. Efficient enforcement of action-aware purpose-based access control within relational database management systems. *Knowledge and Data Engineering, IEEE Transactions on*, 27(8):2134–2147, Aug 2015.

[36] P. Colombo and E. Ferrari. Enhancing mongodb with purpose based access control. *IEEE Transactions on Dependable and Secure Computing*, 2015. in press.

[37] P. Colombo and E. Ferrari. Privacy aware access control for big data: A research roadmap. *Big Data Research*, 2(4), 2015.

[38] Mauro Conti, Roberto Di Pietro, Luigi Mancini, and Alessandro Mei. Distributed detection of clone attacks in wireless sensor networks. *IEEE transactions on dependable and secure computing*, 8(5):685–698, 2011.

[39] Baris Coskun, Sven Dietrich, and Nasir Memon. Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 131–140. ACM, 2010.

[40] Kyle Croman et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[41] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, June 2012.

[42] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4):2233–2243, 2014.

[43] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[44] Steve Deering and Robert Hinden. Internet protocol, version 6 (ipv6) specification. Technical report, 2017.

[45] Ali Dorri et al. Towards an optimized blockchain for iot. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, pages 173–178. ACM, 2017.

[46] Ali Dorri, Salil S Kanhere, and Raja Jurdak. Blockchain in internet of things: Challenges and solutions. *arXiv preprint arXiv:1608.05187*, 2016.

[47] Ali Dorri, Salil S Kanhere, Raja Jurdak, and Praveen Gauravaram. Lsb: A lightweight scalable blockchain for iot security and privacy. *arXiv preprint arXiv:1712.02969*, 2017.

[48] Ali Dorri, Marco Steger, Salil S Kanhere, and Raja Jurdak. Blockchain: A distributed solution to automotive security and privacy. *IEEE Communications Magazine*, 55(12):119–125, 2017.

[49] Mahmoud Elkhodr, Seyed Shahrestani, and Hon Cheung. A contextual-adaptive location disclosure agent for general devices in the internet of things. In *Local Computer Networks Workshops (LCN Workshops), 2013 IEEE 38th Conference on*, pages 848–855. IEEE, 2013.

[50] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *NSDI*, pages 45–59, 2016.

[51] Lujun Fang and Kristen LeFevre. Privacy wizards for social networking sites. In *Proceedings of the 19th international conference on World wide web*, pages 351–360. ACM, 2010.

[52] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[53] José Antonio Galache, Takuro Yonezawa, Levent Gurgen, Daniele Pavia, Marco Grella, and Hiroyuki Maeomichi. Clout: Leveraging cloud computing techniques for improving management of massive iot data. In *Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on*, pages 324–327. IEEE, 2014.

[54] Simson L Garfinkel, Ari Juels, and Ravi Pappu. Rfid privacy: An overview of problems and proposed solutions. *IEEE Security & Privacy*, (3):34–43, 2005.

[55] Arthur Gervais et al. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[56] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *USENIX security symposium*, volume 5, pages 139–154, 2008.

[57] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[58] David L Hall and James Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, 1997.

[59] Huy Hang, Xuetao Wei, Michalis Faloutsos, and Tina Eliassi-Rad. Entelecheia: Detecting p2p botnets in their waiting stage. In *IFIP Networking Conference, 2013*, pages 1–9. IEEE, 2013.

[60] Sayed Hadi Hashemi, Faraz Faghri, Paul Rausch, and Roy H Campbell. World of empowered iot users. In *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*, pages 13–24. IEEE, 2016.

[61] Daojing He, Jiajun Bu, Sencun Zhu, Sammy Chan, and Chun Chen. Distributed access control with privacy support in wireless sensor networks. *IEEE Transactions on Wireless Communications*, 10(10):3472–3481, 2011.

[62] Shuqing He, Bo Cheng, Haifeng Wang, Xuelian Xiao, Yunpeng Cao, and Junliang Chen. Data security storage model for fog computing in large-scale iot application. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 39–44. IEEE, 2018.

[63] Tobias Heer, Stefan Götz, Oscar Garcia Morchon, and Klaus Wehrle. Alpha: an adaptive and lightweight protocol for hop-by-hop authentication. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 23. ACM, 2008.

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

[65] A. Jacobsson and P. Davidsson. Towards a model of privacy and security for smart homes. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pages 727–732. IEEE, 2015.

[66] Sonia Jahid, Prateek Mittal, and Nikita Borisov. Easier: Encryption-based access control in social networks with efficient revocation. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 411–415. ACM, 2011.

[67] J.Zhang, Junjie, Junjie, and Roberto Perdisci. Detecting stealthy p2p botnets using statistical traffic fingerprints. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 121–132. IEEE, 2011.

[68] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, and Jesus Alonso-Zarate. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing*, 3(1):11–17, 2015.

[69] Ghassan O Karame et al. Misbehavior in bitcoin: A study of double-spending and accountability. *ACM Transactions on Information and System Security (TISSEC)*, 18(1):2, 2015.

[70] Mohammad Karami, Youngsam Park, and Damon McCoy. Stress testing the booters: Understanding and undermining the business of ddos services. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1033–1043. International World Wide Web Conferences Steering Committee, 2016.

[71] Jonathan Katz, Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[72] Stephen Kent and Karen Seo. Rfc 4301: Security architecture for the internet protocol. 2005. *URL: https://tools. ietf. org/html/rfc4301 (cited on page 99)*, 2016.

[73] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *Frontiers*

of Information Technology (FIT), 2012 10th International Conference on, pages 257–260. IEEE, 2012.

[74] Sanaz Kianoush, Muneeba Raja, Stefano Savazzi, and Stephan Sigg. A cloud-iot platform for passive radio sensing: challenges and application case studies. *IEEE Internet of Things Journal*, 2018.

[75] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[76] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[77] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[78] Boohyung Lee and Jong-Hyouk Lee. Blockchain-based secure firmware update for embedded devices in an internet of things environment. *The Journal of Supercomputing*, pages 1–16, 2016.

[79] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.

[80] Bin Liu, Xiao Liang Yu, Shiping Chen, Xiwei Xu, and Liming Zhu. Blockchain based data integrity service framework for iot data. In *Web Services (ICWS), 2017 IEEE International Conference on*, pages 468–475. IEEE, 2017.

[81] Caiming Liu, Jin Yang, Yan Zhang, Run Chen, and Jinquan Zeng. Research on immunity-based intrusion detection technology for the internet of things. In *Natural Computation (ICNC), 2011 Seventh International Conference on*, volume 1, pages 212–216. IEEE, 2011.

[82] Jing Liu, Yang Xiao, and CL Philip Chen. Authentication and access control in the internet of things. In *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, pages 588–592. IEEE, 2012.

[83] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[84] Yod-Samuel Martin and Antonio Kung. Methods and tools for gdpr compliance through privacy and data protection engineering. In *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 108–111. IEEE, 2018.

[85] Oleksiy Mazhelis, Antti Hämäläinen, Tomi Asp, and Pasi Tyrväinen. Towards enabling privacy preserving smart city apps. In *Smart Cities Conference (ISC2), 2016 IEEE International*, pages 1–7. IEEE, 2016.

[86] Carlo Maria Medaglia and Alexandru Serbanati. An overview of privacy and security issues in the internet of things. In *The Internet of Things*, pages 389–395. Springer, 2010.

[87] Andrew Miller et al. Discovering bitcoin's public topology and influential nodes. *et al.*, 2015.

[88] Sho Mizuno, Mitsuhiro Hatada, Tatsuya Mori, and Shigeki Goto. Botdetector: A robust and scalable approach toward detecting malware-infected devices. In *Communications (ICC), 2017 IEEE International Conference on*, pages 1–7. IEEE, 2017.

[89] Geoff Mulligan. The 6lowpan architecture. In *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets '07, pages 78–82, New York, NY, USA, 2007. ACM.

[90] Arslan Munir, Prasanna Kansakar, and Samee U Khan. Ifciot: Integrated fog cloud iot: A novel architectural paradigm for the future internet of things. *IEEE Consumer Electronics Magazine*, 6(3):74–82, 2017.

[91] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding p2p bots with structured graph analysis. In *USENIX Security Symposium*, volume 10, pages 95–110, 2010.

[92] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[93] Pratik Narang, Subhajit Ray, Chittaranjan Hota, and Venkat Venkatakrishnan. Peershark: detecting peer-to-peer botnets by tracking conversations. In *Security and Privacy Workshops (SPW), 2014 IEEE*, pages 108–115. IEEE, 2014.

[94] Christopher Natoli and Vincent Gramoli. The blockchain anomaly. In *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*, pages 310–317. IEEE, 2016.

[95] Gabriel Neagu, Ştefan Preda, Alexandru Stanciu, and Vladimir Florian. A cloud-iot based sensing service for health monitoring. In *E-Health and Bioengineering Conference (EHB), 2017*, pages 53–56. IEEE, 2017.

[96] Ricardo Neisse, Gianmarco Baldini, Gary Steri, Yutaka Miyake, Shinsaku Kiyomoto, and Abdur Rahim Biswas. An agent-based framework for informed consent in the internet of things. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pages 789–794. IEEE, 2015.

[97] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[98] NIC Nic. Disruptive civil technologies: Six technologies with potential impacts on us interests out to 2025. *Tech. Rep.*, 2008.

[99] 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*, 9(18):5943–5964, 2016.

[100] Aafaf Ouaddah, Anas Abou Elkalam, and Abdellah Ait Ouahman. Towards a novel privacy-preserving access control model based on blockchain technology in iot. In *Europe and MENA Cooperation Advances in Information and Communication Technologies*, pages 523–533. Springer, 2017.

[101] Siani Pearson and Marco Casassa-Mont. Sticky policies: an approach for managing privacy across multiple parties. *Computer*, 2011.

[102] Otto Julio Ahlert Pinno, Andre Ricardo Abed Gregio, and Luis CE 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. IEEE, 2017.

[103] Pawani Porambage, Mika Ylianttila, Corinna Schmitt, Pardeep Kumar, Andrei Gurtov, and Athanasios V Vasilakos. The quest for privacy in the internet of things. *IEEE Cloud Computing*, (2):36–45, 2016.

[104] Babak Rahbarinia, Roberto Perdisci, Andrea Lanzi, and Kang Li. Peerrush: Mining for unwanted p2p traffic. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 62–82. Springer, 2013.

[105] Y Recommendation. 2060. *Overview of Internet of Things. ITU-T, Geneva*, 2012.

[106] EU Regulation. 679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). *Off J Eur Union*, page L119, 2016.

[107] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[108] Rodrigo Roman, Jianying Zhou, and Javier Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279, 2013.

[109] Sushmita Ruj, Milos Stojmenovic, and Amiya Nayak. Decentralized access control with anonymous authentication of data stored in clouds. *IEEE transactions on parallel and distributed systems*, 25(2):384–394, 2014.

[110] Sherif Saad, Issa Traore, Ali Ghorbani, Bassam Sayed, David Zhao, Wei Lu, John Felix, and Payman Hakimian. Detecting p2p botnets through network behavior analysis and machine learning. In *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*, pages 174–180. IEEE, 2011.

[111] Sherif Saad, Issa Traore, Ali Ghorbani, Bassam Sayed, David Zhao, Wei Lu, John Felix, and Payman Hakimian. Detecting p2p botnets through network behavior analysis and machine learning. In *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*, pages 174–180. IEEE, 2011.

[112] G. Sagirlar, B. Carminati, and E. Ferrari. Decentralizing Privacy Enforcement for Internet of Things Smart Objects. *ArXiv e-prints*, April 2018.

[113] S. Sarkar, S. Chatterjee, and S. Misra. Assessment of the suitability of fog computing in the context of internet of things. *IEEE Transactions on Cloud Computing*, 6(1):46–59, Jan 2018.

[114] Zach Shelby and Carsten Bormann. *6LoWPAN: The wireless embedded Internet*, volume 43. John Wiley & Sons, 2011.

[115] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). Technical report, 2014.

[116] Robert W Shirey. Internet security glossary, version 2. 2007.

[117] S. Sicari, A. Rizzardi, L.A. Grieco, and A. Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76:146 – 164, 2015.

[118] Sabrina Sicari, Alessandra Rizzardi, Daniele Miorandi, Cinzia Cappiello, and Alberto Coen-Porisini. Security policy enforcement for networked smart objects. *Computer Networks*, 108:133 – 147, 2016.

[119] Sabrina Sicari, Alessandra Rizzardi, Daniele Miorandi, and Alberto Coen-Porisini. Security towards the edge: Sticky policy enforcement for networked smart objects. *Information Systems*, 71:78–89, 2017.

[120] Vijay Sivaraman, Hassan Habibi Gharakheili, Arun Vishwanath, Roksana Boreli, and Olivier Mehani. Network-level security and privacy control for smart-home iot devices. In *WiMob*, pages 163–167. IEEE, 2015.

[121] Antonio F Skarmeta, José L Hernández-Ramos, and M Victoria Moreno. A decentralized approach for security and privacy challenges in the internet of things. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 67–72. IEEE, 2014.

[122] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.

[123] Jinshu Su, Dan Cao, Baokang Zhao, Xiaofeng Wang, and Ilsun You. epass: An expressive attribute-based signature scheme with privacy and an unforgeability guarantee for the internet of things. *Future Generation Computer Systems*, 33:11–18, 2014.

[124] Nick Szabo. Smart contracts. *Unpublished manuscript*, 1994.

[125] A. Taivalsaari and T. Mikkonen. A taxonomy of iot client architectures. *IEEE Software*, 35(3):83–88, May 2018.

[126] Chandu Thota, Revathi Sundarasekar, Gunasekaran Manogaran, R Varatharajan, and MK Priyan. Centralized fog computing security platform for iot and cloud in healthcare system. In *Exploring the convergence of big data and the internet of things*, pages 141–154. IGI Global, 2018.

[127] Ilaria Torre, Frosina Koceva, Odnan Ref Sanchez, and Giovanni Adorni. A framework for personal data protection in the iot. In *Internet Technology and Secured Transactions (ICITST), 2016 11th International Conference for*, pages 384–391. IEEE, 2016.

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

[129] Luis M Vaquero and Luis Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.

[130] MQTT Version. 3.1. 1. *Edited by Andrew Banks and Rahul Gupta*, 10, 2014.

[131] Jeffrey Voas. Networks of things. *NIST SPECIAL PUBLICATIONS (SP 800)*, (183), 2016.

[132] Gernot Vormayr, Tanja Zseby, and Joachim Fabini. Botnet communication patterns. *IEEE Communications Surveys & Tutorials*, 19(4):2768–2796, 2017.

[133] Guojun Wang, Qin Liu, and Jie Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 735–737. ACM, 2010.

[134] Jing Wang and Ioannis Ch Paschalidis. Botnet detection based on anomaly and community detection. *IEEE Transactions on Control of Network Systems*, 4(2):392–404, 2017.

[135] Quangang Wen, Xinzheng Dong, and Ronggao Zhang. Application of dynamic variable cipher security certificate in internet of things. In *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, volume 3, pages 1062–1066. IEEE, 2012.

[136] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2016.

[137] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD*. ACM, 2006.

[138] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun, and Hui-Ying Du. Research on the architecture of internet of things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 5, pages V5–484. IEEE, 2010.

[139] Qiben Yan, Yao Zheng, Tingting Jiang, Wenjing Lou, and Y Thomas Hou. Peerclean: Unveiling peer-to-peer botnets through dynamic group behavior analysis. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 316–324. IEEE, 2015.

[140] Yuchen Yang, Longfei Wu, Guisheng Yin, Lijie Li, and Hongbin Zhao. A survey on security and privacy issues in internet-of-things. *IEEE Internet of Things Journal*, 4(5):1250–1258, 2017.

[141] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, and M. Guizani. Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges. *IEEE Wireless Communications*, 24(3):10–16, June 2017.

[142] 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, 2014.

[143] Ting-Fang Yen and Michael K Reiter. Are your hosts trading or plotting? telling p2p file-sharing and bots apart. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pages 241–252. IEEE, 2010.

[144] Junjie Zhang, Roberto Perdisci, Wenke Lee, Xiapu Luo, and Unum Sarfraz. Building a scalable system for stealthy p2p-botnet detection. *IEEE transactions on information forensics and security*, 9(1):27–38, 2014.

[145] Rui Zhang, Yanchao Zhang, and Kui Ren. Distributed privacy-preserving access control in sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1427–1438, 2012.

[146] Jianliang Zhen, Jie Li, Myung J Lee, and Michael Anshel. A lightweight encryption and authentication scheme for wireless sensor networks. *International Journal of Security and Networks*, 1(3-4):138–146, 2006.

[147] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *Big Data (BigData Congress), 2017 IEEE International Congress on*, pages 557–564. IEEE, 2017.

[148] Jin Zhou, Liang Hu, Feng Wang, Huimin Lu, and Kuo Zhao. An efficient multidimensional fusion algorithm for iot data based on partitioning. *tsinghua science and technology*, 18(4):369–378, 2013.

[149] Di Zhuang and J Morris Chang. Peerhunter: Detecting peer-to-peer botnets through community behavior analysis. In *Dependable and Secure Computing, 2017 IEEE Conference on*, pages 493–500. IEEE, 2017.

[150] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.