SECURE AND PRIVATE ACCESS CONTROL FOR SYSTEMS OF SMART DEVICES

By

Tam Dan Le

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science - Doctor of Philosophy

2019

ABSTRACT

SECURE AND PRIVATE ACCESS CONTROL FOR SYSTEMS OF SMART DEVICES

By

Tam Dan Le

With the emergence of Internet of Things (IoT) technologies and the invasion of smart devices in almost every aspect of our lives, access control that allows only authorized users to access IoT devices becomes an important problem. The limited capabilities of the devices and the distributed nature of IoT environments have presented unique challenges to the design of an effective access control mechanism. First, it should be lightweight enough for the IoT devices to handle due to their resource constraints. Second, the variety of devices and applications and the arbitrary manners of users require the support of fined-grain, flexible access control policies. Last but not least, traditional access control models that are often centralized may not be suitable for distributed IoT. Therefore, a decentralized approach should be considered.

In this dissertation, we propose access control solutions that are not only secure and private but also scalable to meet IoT requirements. Our first design is an authorization protocol that supports flexible delegation for smart home applications. The protocol allows users to create and share various permissions within their authorities to other users. In addition, since simple computation operations are used, the protocol is lightweight and supports fast validation at resource-constrained devices.

Next, the need to support larger environments and the open problem with the exchange of access keys without a central authority motivate us to seek a decentralized solution from blockchain technology, which is originated from the famous cryptocurrency Bitcoin. The advantages of blockchain, which lie in an immutable distributed ledger that is maintained by a peer-to-peer network of untrusted nodes, can bring decentralization to IoT applications. However, applying blockchain to IoT is not straightforward as it was not originally designed for IoT requirements. We address two main issues in blockchain-based access control for IoT

systems. First, since blockchain is a public platform, user privacy is one of the top priorities. Second, resource-constrained IoT devices are often not powerful enough to interact directly with the blockchain but need to rely on certain trusted nodes to retrieve blockchain data.

The first issue of user privacy leads to our design of CapChain, a blockchain-based privacy-preserving access control framework that enables the sharing of access capabilities to multiple devices in a secure and private manner. Then, applying similar techniques to CapChain but also extending the use of blockchain by smart contracts, we design a privacy-preserving service that allows users to create IoT automated tasks by defining one of multiple conditional statements that need to be satisfied before a task can be performed. We set up strict privilege at the triggering party, such that it may not trigger the task any time except only when the conditions are satisfied.

To address the second issue of resource constrained devices, we propose a method for IoT devices to validate blockchain data without solely being dependent on a central server. In our approach, several witnesses on the network can be selected randomly by the devices to validate access control information. Our method is aided by Bloom filters, which are shown to be lightweight for resource-constrained devices.

To my family...

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my advisor Dr. Matt W. Mutka, who introduced me to a new research area in the Internet of Things at the time when I felt lost in my research. Without his invaluable advice, tremendous patience and guidance, the completion of this dissertation would not have been possible.

At Michigan State University, I am grateful to Dr. Abdol-Hossein Esfahanian, Dr. Li Xiao, and Dr. Mi Zhang, who have been supportive of the research that went into these pages. I also thank the Department of Computer Science and Engineering for providing me with teaching assistantships that provided me considerable academic experience and opportunities to learn from my students.

It is a pleasure to be able to acknowledge the considerable institutional and financial support that I received from Vietnam Education Foundation (VEF) to attend Michigan State University. Thanks to VEF, I also met great friends Duyen Huynh, Huan Do, Lieu Hoang, Nhu Nguyen and other VEF fellows, without whom my time at MSU would have been much less enjoyable.

I would like to thank my mother for her unwavering dedication to me and my brother. I thank my father for believing in me and my choice of this career path.

Last but not least, I am extremely thankful to my brother Kien Le for being (and also bearing) with me in the long course of writing this dissertation. His rigorous critiques of my writing were most instrumental in helping to shape the contours of this dissertation. And without his company across many global locations, the past six years would not have been so eventful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

Introduction

Internet of Things (IoT) devices have become pervasive in almost every aspect of our lives, from home automation, health care to industries and transportation. A smartphone will no longer be the only "smart" thing a person can own. Instead, a person may possess plenty of devices and vice versa, a single device can be shared between multiple users. As a result, secure and private access control to IoT devices becomes critical.

Although there has not been a formal definition of IoT, the most common one is that the IoT is a network of connected objects that can collect and exchange data [63, 2]. Figure 1.1 describes a variety of "things". By the definition, a "thing" can be any device with Internet connectivity. However, in this dissertation, we do not consider powerful machines like laptops, smartphones and tablets IoT devices. Instead, we are specifically interested in the range of devices with low computational power and memory such as sensors, actuators, wearables, etc.

IoT environments presents unique challenges to security and privacy. In terms of user experience, due to the diversity of both users and IoT devices, it is difficult to define a flexible access control mechanism that can satisfy all of user's ad-hoc behaviors and demands [51]. On the other hand, in terms of system architecture, centralized solutions present a latent single point of failure and may not be suitable for a heterogeneous environment like IoT. A decentralized/distributed architecture may be more beneficial, however, protecting user privacy among untrusted parties would be challenging. Last but not least, access control

Figure 1.1: A variety of IoT devices [1].

mechanisms must consider the limited capabilities of IoT devices as they cannot afford high-computational operations.

In this dissertation, we pose a research question: can we design access control solutions that are not only secure and private but also scalable to meet IoT requirements? We consider not only smart homes but also larger-scale environments such as smart cities and envision the emerging technology *blockchain* as a potential solution. This chapter introduces the challenges and our motivations for a secure and private access control for IoT environments.

1.1 Research motivations

1.1.1 Flexible and lightweight access control model

A case study on several commercial smart home devices [82] points out that although each studied device employs a different access control mechanism, none provides a convenient method to share access with other users. In a more recent study [87], the authors also express their concerns about the imbalance of power between multiple users who are supposed to have the same role, where some users may have (intentionally or unintentionally) more privileges

than others. We present a common example in the IoT context, which is illustrated in Figure 1.2: Bob is a home owner, thus has a full control of all devices at his home. He is often out of town so he offers his house for lodging on Airbnb. Each time his house is booked, he needs to grant a house key to his renter. Bob may be out of town, but he can create a digital key that provides full control to all devices in the apartment, such as HVAC, lighting system, etc. but only limited access to the surveillance camera and the door lock, and send the key to the renter Dave. As Bob is very cautious, he also authorizes a security company to receive notifications and logs from his door lock, but they are not allowed to enter the house. Dave also has his family who should be able to use the system. It would be more convenient for Dave to grant permissions to his family members instead of Bob. However, as Dave is not authorized to change the passcode of the door lock, he can only delegate the same or lower permission. When the lease expires, Bob revokes the key, including sub-keys created by Dave without any physical contacts. The above scenario addresses several

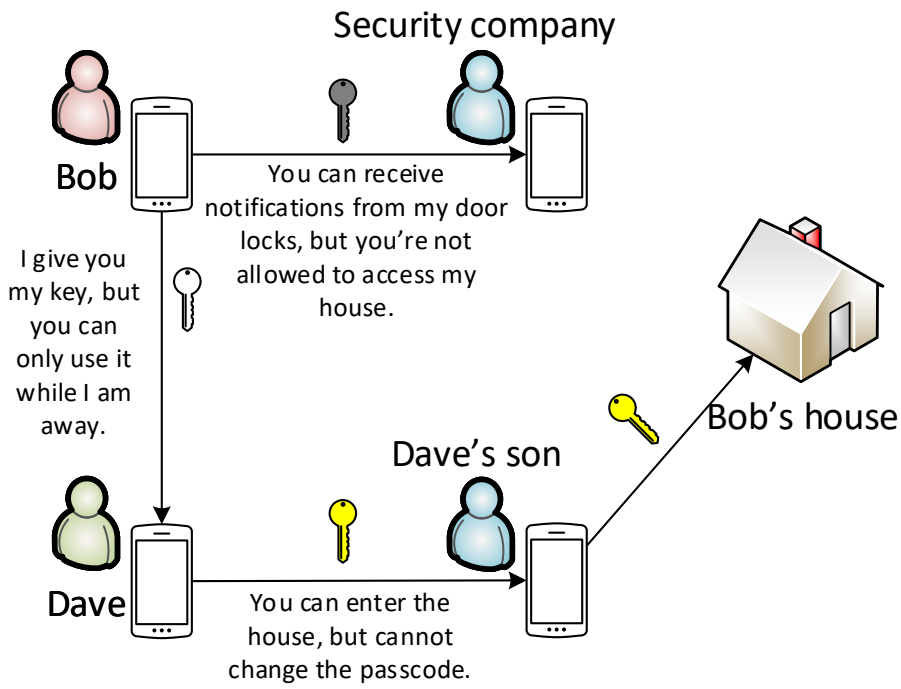


Figure 1.2: A smart home scenario

problems for access control in IoT environments. First it should be scalable and flexible so that a user can create various permissions, depending on his/her needs and capability. As keys can be transferred among multiple users, efficient delegation and revocation should also be supported. Moreover, many IoT devices have limited resources, such as low computational power or battery capacity, therefore they cannot support complicated security mechanisms.

1.1.2 Problems with centralized IoT

In centralized IoT architectures, IoT devices need to rely on some more powerful central entities or outsourced cloud services to manage access control policies [88, 77]. While cloud solutions have been supporting Internet of Things (IoT) services with management, data processing and storage, its centralized architecture also raises a concern about security and trustworthiness of cloud providers due to the importance of the IoT data they can access [76] as well as their privileges over the devices. Reliance on cloud systems can introduce a latent single point of failure if the servers are down or attacked [86].

In terms of delegation, the key transfer is also usually carried out via a server operated by the device manufacturer. For a heterogeneous environment like IoT, the problem becomes more complicated if the delegation occurs across domains. We elaborate this by another scenario: Alice is on the way back to her apartment after work. When she passes the parking gate, it triggers her smart home hub/manager to turn on her HVAC and command her smart kettle to boil some water so that by the time Alice gets home, her apartment is already warmed. In this scenario, Alice needs to allow the parking gate to inform her smart home hub and her hub to control her heater and kettle. While Alice's devices are under her control, the parking gate belongs to the apartment building and thus uses a different infrastructure. Therefore to enable such delegation, a third party is required to communicate between the two parties. For example, IFTTT [1], which stands for If-This-Then-That, is a cloud service that allows users to provide a set of rules to their services. By IFTTT, Alice's

[1] https://ifttt.com/discover

scenario can be described as a rule "if I open the parking gate, turn on the heater". By authorizing IFTTT to access the gate and heater on her behalf, Alice allows IFTTT to act as a bridge to connect the 2 services. IFTTT has been very popular among smart home communities and partnered with a lot of companies. As a result, a large amount of API keys are stored by IFTTT, which is not only a privacy issue to users but also makes IFTTT a potential target to attackers [36].

1.1.3 Decentralized IoT with blockchains

Companies and researchers have recently sought for an alternative decentralized solution from blockchain, the core technology behind the famous cryptocurrency Bitcoin. A blockchain is a distributed public ledger that records transactions of digital assets. What makes blockchain a timestamped, immutable database is a consensus protocol called *Proof-of-Work* (PoW) that guarantees every untrusted participant in the network behaves properly. With blockchain, IoT applications can be built on a decentralized, trustless peer-to-peer network without the need of a central authority [29]. In a blockchain context, Alice's scenario can work as follows: the interaction between Alice and the parking gate results in a transaction on the blockchain, then Alice's devices can verify the existence of such transaction and perform the required operations.

However, applying blockchain technology to IoT is not straightforward. Blockchain is currently facing problems with scalability, latency and high computation due to PoW. These problems become more prominent in IoT contexts as the resources of many IoT devices are very limited. As a result, the IoT devices cannot interact directly with the blockchain network and need to rely on one or several gateways, which may remove the benefit of decentralization. More importantly, since most blockchain systems operate in public mode, i.e. transactions on blockchains are visible to everyone, privacy is not considered. On the other hand, access control information is one of the most sensitive and should not be exposed in public. While privacy has been a concern with extensive studies in blockchain communities, solving privacy

with consideration of IoT devices is still a challenge as most of the techniques for general privacy-preserving blockchains cannot be afforded on resource-constraint devices.

1.2 Research contributions

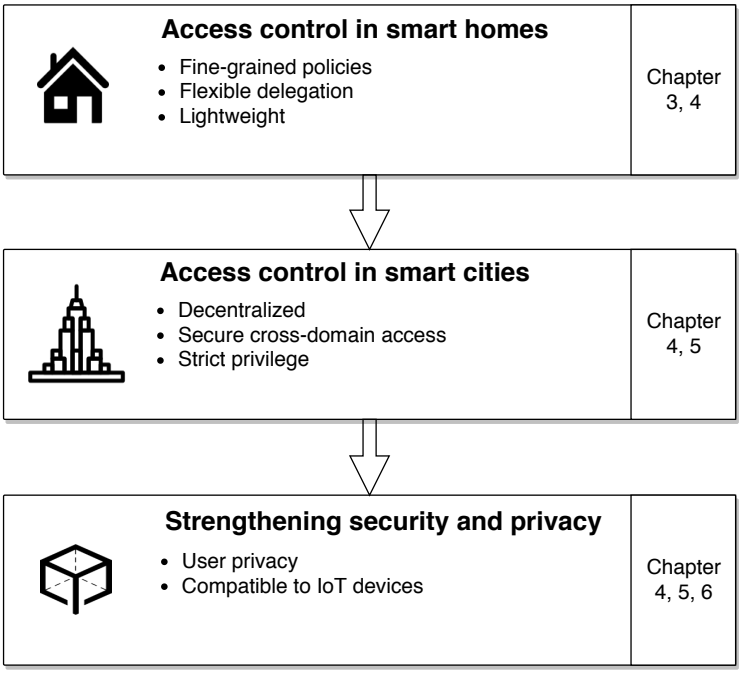


Figure 1.3: Dissertation road map

Figure 1.3 illustrates the road map of the dissertation. First, we consider small and close smart home environments, which require fine-grained, flexible and lightweight access control. Then, we expand to a larger scale, i.e. smart cities, and identify more issues, including the requirement of a decentralized architecture and secure cross-domain access with strict privilege at authorized parties. Choosing blockchain as the decentralized platform, we try to solve the privacy issues and make it more transparent to IoT devices.

In particular, this dissertation contributes in the following:

- *Authorization and delegation in smart home environments.* We propose lightweight authorization protocol with support of a delegation chain for smart home environments, in which a user can easily transfer (part of) his/her access rights in the form of a Bloom

filter. In our mechanism, a trusted chain can be maintained and verified by a single request without the need of an access control list. The security of our protocol is based on the false positive rate of a Bloom filter.

- *Privacy-preserving blockchain-based access control.* We explore the possibility of applying blockchain by designing CapChain - an access control framework based on blockchain that allows users to share and delegate their access rights easily to IoT devices in public but still maintain privacy. To protect privacy, we adapt multiple techniques from anonymous crypto-currency blockchain systems to hide sensitive information, including users' identities and related information about the capabilities.

- *Privacy-preserving automated tasks.* Utilizing user permissions to create IoT automated tasks, we present PPCA, a privacy-preserving service that allows users to create conditional actions in a decentralized platform using smart contracts. PPCA can guarantee strict privilege such that a third party that holds a conditional action cannot perform it when the condition is not satisfied. By generalizing a variety of conditions into simple forms of conditional logic, the conditions can be verified in a privacy-preserving manner on a public blockchain.

- *Block validation for IoT devices in blockchain-based applications.* We propose a lightweight block validation method that helps resource-constraint IoT devices to validate blockchain data without solely being dependent on a central server. The method employs a random selection of witnesses on the blockchain network to help the devices validate access control information. It is aided by Bloom filters, which are shown to be lightweight for resource-constrained devices.

Figure 1.4 summarizes the problem areas and our research interests.

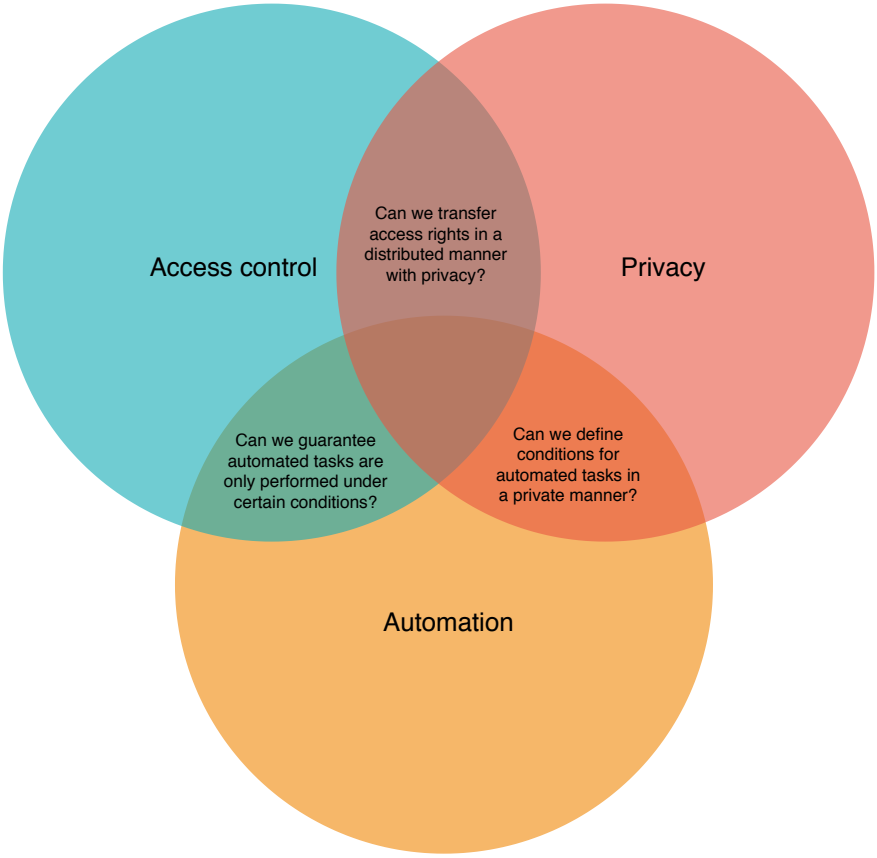


Figure 1.4: A summary of research problems.

1.3 Structure of the Content

We provide background and related work on access control in IoT environments and blockchain technologies in Chapter 2. Chapter 3 targets smart home applications and presents our lightweight authorization protocol with delegation. Chapter 4 discusses the privacy issues of general blockchains and presents the design of CapChain to provide a reliable and flexible key sharing framework. In chapter 5, we extend further the use of blockchains by using smart contracts to support IoT conditional tasks with PPCA. Next, chapter 6 describe our lightweight block validation method for resourc-constraint devices in blockchain-based applications. Finally, Chapter 7 provides a summary and potential future work.

Chapter 2

Background and Related Work

In this chapter we provide two surveys. Section 2.1 is a discussion on access control in IoT environments. It hightlights the current security issues with IoT devices and different IoT access control mechanisms. Then, we review the state of the art of blockchain technology and its applications for IoT in section 2.2.

2.1 Access control in IoT environments

In this section we first review current state of security and privacy of commercial smart home devices. Second, we discuss some representative access control frameworks that follow capability-based access control model. Finally, we discuss authentication and authorization protocols for IoT devices.

2.1.1 Security and privacy of smart home devices

Many commercial smart home products have been studied and found to have insufficient security mechanisms. Examining a smart light-bulb, a power switch and a smoke-alarm, Notra et al. [69] highlight the lack of encryption, appropriate authentication, message integrity checks and privacy implications in those devices and propose a network-level protection. More recently, smart lock systems have also been found to be vulnerable to several threats that allows attackers to obtain unauthorized access and private information about the user's

home [46]. Three categories of attacks are presented, including privacy leakage, unwanted unlocking and revocation/logging evasion [35]. Focusing on a programming framework, Fernandes et al. analyze the Smartthings platform, which support developers to write their own applications called SmartApps to control smart devices. They found that the capability model designed by Smartthings often grants to SmartApps capabilities that were not explicitly requested. As a result, attackers can exploit the overprivilege to create malicious applications and perform unauthorized operations.

Centralized servers have also posed several problems. A recent outage incident with Samsung's SmartThings cloud servers has made users in North America unable to use their smart appliances for hours [14]. In addition, a serious vulnerability due to a flaw in the manufacturer's cloud server architecture has been found in some popular smart cameras [54], which can allow attackers to gain access to all of the cameras.

In terms of access control, Ur et al. conducted a case study on existing commercial home automation devices [82] and point out that although each studied device employs a different access control mechanism, none provides a convenient method to share access with other users. Another user study is presented in [51] by Kim et al, in which a set of intuitive access control policies is proposed. They also address the problem of access sharing in a smart home system, which is very ad-hoc and dependent on user's demands. Based on user interview results, they suggest four group of policies: full control, restricted control, partial control, minimal control.

2.1.2 Capability-based access control model

In [77], Roman et al. addresses the importance of an access control model for distributed IoT that should support granular policies, inexpensive computational overhead for resource-constrained devices and delegation mechanisms that allows sharing between users and across domains. In this sense, capability-based access control (CBAC) is found to be more suitable for IoT environments than other traditional access control models, including access control

list (ACL), role-based access control (RBAC) and attribute-based access control (ABAC) [47]. In [16], Anggorojati et al. present a delegation mechanism based on CBAC that supports machine-to-machine cross-domain communication. However, their model requires each IoT network domain to have a manager to authorize delegation requests.

On the other hand, the CBAC approach proposed by [44] allows users to manage and share their own access control to services and information by issuing capability tokens, which will be presented to the IoT devices for authorization. However, the mechanism to validate a token or a chain of tokens was not addressed. To resolve that issue, Skarmeta et al. [80] propose a distributed CBAC mechanism that is built on public key cryptography and digital signatures. In their approach, a device owner grants his/her users a signed capability token using Elliptic Curve Digital Signature Algorithm (ECDSA). The device then has to verify the token by examining the signature and other access control policies attached in the token. Although this approach works well with one level of delegation, i.e. from one owner to multiple users, it will create more computational overhead for the device when the delegation chain expands to more than 1 hop.

A similar idea of using capability tokens is also proposed by IETF [79] to enable a client to obtain limited access to a resource with the permission of a resource owner. However, it also only supports 1-hop delegation. Moreover, an authorization server is required for delegation as well as registration, which can be a problem of central point of failure.

Hussein et al. [47] introduce a Community capability-based access control framework (COCapBAC), in which an IoT community is formed by IoT devices that shares the same mission, then a more capable device in the community can make decisions on behalf of those with limited resources. Their framework is motivated by the conjunction of IoT devices to achieve a common goal, for example, a speaker often operates together with a TV box in a home theater. How to build such community with trust, however, was not addressed.

2.1.3 Authentication and authorization with delegation

The idea of using smartphones to delegate authority to other users has been introduced very early by Bauer et al. [18] with the Grey project - a decentralized access control system office buildings deployed at Carnegie Mellon University. In this project, they designed a verification process in which a guest can prove his permission to access an office door with help from an authorized user. The framework integrates several technologies to support delegation, user authorization and communication among phones and resources. However, since Grey was built at the early stage of Internet of Things evolution, the design was mainly focused on end-user applications and did not consider the capability of resource devices.

In order to avoid a central token issuer, Dmitrienko et al. introduced SmartTokens - a delegable access control system for NFC-enabled smart phones that allows users to delegate their access rights to other users without involvement of a central authority [31]. Although SmartTokens uses symmetric cryptography which is suitable for constrained devices, this at the same time hinders the delegation scalability because a user must present all delegated tokens through the delegation chain in order to be verified.

TARD [56] is a cryptographic protocol based on one-way hash functions to grant a temporary access to a device. The condition to use the token is specified by a counter which is set up through a hash chain to allow the temporary token to be used only for a certain number of times. Since TARD does not use any complicated cryptography but just hash functions, it is very light-weight and suitable for resource-constrained device. However, the protocol cannot support fine-grained access control policies.

2.1.4 Other approaches

Master key [89] is also an early approach to provide access to digital locks through a single device (the Master Key) which can aggregate all of user's digital forms of access tokens for entity authentication. The advantage of Master Key is, despite that one may interact with many locks, and a lock may interact with many key owners, an user does not need to

remember all of the relationships between keys and locks but is still able to achieve mutual authentication with few messages. However, the design of Master Key requires a shared secret between each pair of key and lock and does not support delegation as lending one or several sub-keys to another user is equivalent to sharing the secrets. In addition, there may be different operations with locks (or smart devices) besides open/unlock (for examples, read data, modification, etc.), which may also have a valid time. Meanwhile Master Key only supports authentication, which means it cannot differentiate these operation requests.

Neto et al. [66] present a suit of cryptographic protocols called Authentication of Things (AoT) that provides authentication and access control during all stages in a device's life-cycle, including pre-deployment, ordering, deployment, functioning, and retirement. The cryptographic primitives of AoT includes Identity-Based Cryptography, Attribute-Based Cryptography and Attribute-Based Access Control model. AoT mostly focuses on the relationship between the device and its owner with an extra feature to allow a guest device to authenticate itself with devices from a foreign domain. Device sharing, however, is not in their coverage.

Fotiou et al. [38] develop an access control framework for IoT that uses symmetric encryption rather to set up a secure communication between users and IoT devices. Their design relies on a trusted third party call Access Control Provider to evaluate access control policies and relay the result back to devices.

2.2 Blockchain technology and distributed ledgers

In this section, we provide a review of blockchain technology which is best known for Bitcoin and notable blockchain systems and academic research that aim to enhance one or several aspects of blockchain. Then we review some non-cryptocurrency blockchain applications. Last, we discuss some blockchain frameworks designed specifically for the IoT.

2.2.1 Notable blockchains

Bitcoin

Bitcoin is the most popular cryptocurrency system that was invented by Satoshi Nakamoto in 2008 [65]. In Bitcoin, transactions are recorded on a public immutable ledger called *blockchain*. A block is a group of validated transactions that is broadcast to the whole network by a special node called *miner*. Each block has a reference to its previous block, forming a chain of blocks (aka blockchain) as illustrated in Figure 2.1. The blockchain is distributed, meaning that every node in the network maintains its own copy of it. To allow the network to reach agreement on the current state of the blockchain, Bitcoin relies on a consensus protocol called *Proof-of-Work* (PoW), which involves a mining process and miners. As mentioned earlier, a miner is in charge of gathering new transactions to a block. However before publishing a new block, the miner has to invest his computational power to compete in solving a puzzle. The winner of the competition is the first one who finds a solution, i.e. the proof of work, for the puzzle and thus has his block accepted by the network. To compensate for the invested computation, Bitcoin rewards miners with some coins together with transaction fees paid by users. The reward incentivizes miners to participate in the mining process and secure the network against double-spend attacks, in which a malicious user tries to spend the same money twice.

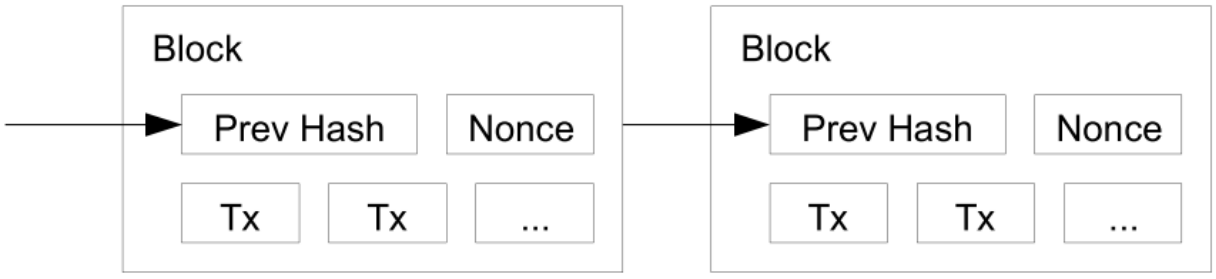


Figure 2.1: Each block refers to the hash of its previous block and forms blockchain (figure borrowed from [65]).

It can be seen that if two miners mine two blocks nearly at the same time, a part of the network will see a different version of blockchain (also known as *fork* in Bitcoin terminology)

than the other one. To solve this problem, Bitcoin applies the *longest chain rule*, i.e. if a node sees more than one version of blockchain, it will choose the longest one and discard the others. Since transactions after being added to a block can still be discarded, Bitcoin's consensus is probabilistic and does not satisfy *consensus finality*, which as defined in [85], is a property that requires that a valid block should never be removed from the blockchain.

Distributed consensus among untrusted parties was first addressed as *Byzantine generals problem* by Lamport et. al. in 1982 [55]. However, despite that a family of *Byzantine fault tolerance* (BFT) state-machine replication protocols has been well-designed and proved to achieve *consensus finality*, which cannot be satisfied by PoW, they do not scale well in terms of network size. Bitcoin, with PoW, is the first peer-to-peer payment system that can provide consensus to thousands of nodes [85]. A workflow of Bitcoin can be summarized as follows:

- To make a payment, a user signs a transaction and broadcasts it.
- Each node in the network validates the transaction and passes it to the rest of the network.
- Miners receive the transaction, group it into a block together with other transactions and start the mining process. Specifically, the puzzle in Bitcoin is to find a number (a nonce) that when being hashed with the block data, produces a certain number of leading zeros in the hash result. The network will accept the block from the miner who first solves the problem.

Ethereum and smart contracts

Ethereum [10] is the second most popular blockchain system after Bitcoin. Although Ethereum also has its own cryptocurrency called *ether* (or ETH), its applications are beyond just a payment system. Ethereum provides a Turing-complete programming language that allows *smart contracts* - programmable computer codes that can automatically run on blockchain. While Bitcoin records money flows on blockchain, Ethereum stores machine

state transitions. Thus, it is guaranteed that smart contracts once triggered will run exactly as they are written without needs of any central authority. To run smart contracts, users need to pay ETH for *gas* - a unit that specifies how much work is needed to perform the contracts. Gas also helps to prevent smart contracts from halting the system by running forever. Smart contracts allows users to build various decentralized applications serving their own purposes such as voting systems, crowd-funding, etc.

Challenges and limitations of blockchains

Bitcoin and other variations of blockchains in general have been facing several challenges:

- Consensus latency: due to forking, users should wait for more blocks coming after the block that contains their newly posted transaction to make sure that it is buried deep enough in a valid chain. As Bitcoin recommends 6 blocks as the minimum number users should wait for, with the current mining rate of 10 minutes per block, it takes users at least 1 hour to be able to confirm their transactions.

- Fixed block size: to prevent denial-of-service attacks, Bitcoin limits its block size to 1 MB, which can support 5-7 transactions per second, originally to avoid DDoS attacks. Compared to other credit card payment systems that can process thousands of transactions per second, such throughput is too low.

- High computational power: since the mining process requires high computational power, the miners often use dedicated hardware which may consume too much energy.

- Privacy: since the blockchain is public, all of the data stored on it is visible to everyone. In case of payment systems like cryptocurrencies, it means that anyone can link and see transactions of others.

2.2.2 Enhancement proposals for blockchains

Alternatives to proof-of-work

Since PoW is computational expensive and produces latency, a new family of protocols called *Proof-of-Stake* (PoS) has been proposed as an alternative. The general idea is that instead of computational power, PoS chooses validators (equivalent to miners in PoW) according to their invested money (aka stakes). As a result, validators can be selected deterministically and transaction confirmation can be made instantly. In early PoS proposals [8, 12, 11], a validator is randomly chosen at each epoch to create a block. However they all faced a common problem called *nothing-at-stake*, where validators can create blocks on multiple chains to maximize their rewards without any cost. PoS was formally studied by Bentov et. al. [19] with analysis of various types of attacks. The nothing-at-stake problem was also addressed by introducing a penalty, so that validators may lose their stakes for their dishonest behaviors.

Most of more recent PoS protocols follow Byzantine Fault Tolerance (BFT) style [10], in which a set of validators are selected to propose blocks, then each validator casts a vote for a certain block. The voting can take several rounds and the final result is everyone agrees on either a given block or no block at all. BFT protocols are often based on different assumptions on the synchrony of the network. Tendermint [53] is a modification of the classic consensus protocol Dwork-Lynch-Stockmeyer (DLS) [34] that assumes a partial synchrony of the network. Aiming specifically at preventing forks, Algorand [42] designed a sortition algorithm to randomly select a set of user/nodes to propose a block and a Byzantine agreement protocol to vote on a final specific block. Ouborous [50] is a provably secure PoS protocol that is proposed at around the same time as Algorand. It implements a secure multiparty coin-flipping protocol to elect a leader that can create a new block. Both Algorand and Ouborous are under the assumption that the network is synchronous. On the other hand, Honey Badger [62] proposes a BFT protocol for asynchronous network that can be applied to cryptocurrencies. However, it involves a fixed set of servers at bootstrapping, making the

system no longer decentralized. It is noteworthy that at the time this dissertation is written, PoS-based blockchains have not been widely adopted yet. While Ethereum is planning to switch to a PoS consensus protocol, the blockchain implementation for most of the discussed protocols (except Tendermint) are still in an early stage of development and there has not been an official release yet.

Privacy-preserving blockchains

Bitcoin is built on UTXO (Unspent Transaction Output) model. A UTXO specifies an amount to be sent to a Bitcoin address. A transaction refers to one or multiple UTXOs from previous transactions as inputs and creates new UTXOs, allowing anyone to track the money flow from a specific address as all of the transactions are visible on the blockchain. There are several cryptocurrency projects that aim to solve the privacy problem. Two representatives are Monero [67] and Zcash [78] with two different approaches.

Monero hides the input(s) of a transaction by mixing it with several unrelated UTXOs from other transactions and produces a ring signature that proves the validity of the hidden input(s). The identity of the receiver and the transferred amount are also hidden using Elliptic Curve Cryptography and commitment scheme. The level of anonymity of Monero depends on the size of the mixed ring.

On the other hand, Zcash provides complete anonymity with zkSnarks - a variance of zero-knowledge proof. However, the computation of the proof is quite expensive and can take up to minutes.

In terms of smart contracts, there are two main approaches to solve the lack of confidentiality and privacy: i) using cryptographic techniques such as zero-knowledge proofs, secure multi-party computation [52, 90]; ii) using trusted execution environments (TEE) such as Intel SGX [22, 28]. Like Zcash, while the use of cryptographic techniques can provide more privacy and confidentiality, it requires intensive computation. On the other hand, the latter approach using TEE requires special hardware and trust on an attestation service

(which is Intel in the case of SGX). Following none of these two approaches, Arbitrum [49] is a platform for scalable and private smart contracts that only publishes hashes of contract states on the blockchain. Although Arbitrum is shown to significantly reduce the on-chain computation cost compared to Ethereum, its performance on a real blockchain network was not evaluated. In addition, how Arbitrum performs on low-power devices is also not clear.

Non-blockchain cryptocurrencies

IOTA [75] and Byteball [30] are two emerging cryptocurrencies that use directed acyclic graph (DAG) instead of blockchain as the data structure. In both systems, if a node wants to send a transaction, it has to validate/confirm and create a reference to 2 or more other transactions, thus a directed acyclic graph is formed instead of a chain.

IOTA is designed specifically for IoT, with the goal to create a machine-to-machine payment system without mining and transaction fees. IOTA achieves consensus by a Monte-Carlo Markov Chain algorithm to select transactions for reference. It is not mentioned in the whitepaper but IOTA currently implements a coordinator to prevent the so-called 33% attack, which is controversial as a the network becomes centralized.

Byteball detects double-spend by setting up a total order among transactions through a main chain. Consensus on the main chain relies on a set of witnesses who are well-known and have good reputation. Although witnesses can be changed by users, this still involves certain centralization.

2.2.3 Non-cryptocurrency blockchains

Business ledgers

All cryptocurrencies that have been mentioned so far are all operated on *public* blockchains/ledgers, which means nodes are free to join and leave. On the other hand, *private/permissioned* blockchains require nodes to be identified or register before joining the network. Therefore, they get more attention from business and financial organizations. Since

the number of nodes is much smaller than in public blockchains, permissioned blockchains often use BFT protocols. Hyperledger [1] is an open-source distributed ledger framework created by the Linux Foundation with the goal of supporting business transactions for various organizations, including banks, stock exchanges, manufacturers, etc. Under the Hyperledger umbrella project is Hyperledger Fabric [2], a modular permissioned blockchain developed by IBM. The Fabric supports smart contracts and pluggable consensus protocols, including a variant of PBFT protocol [27].

Corda is a non-blockchain decentralized database designed for semi-private networks [9]. In Corda, there is no global ledger but each party stores transactions that are only relevant to them. Corda differentiates two types of consensus: validity consensus in which involved parties in a transaction are responsible for the validity of the transaction and uniqueness consensus, which resolves the double-spend problem using a notary network service. Notaries are in charge of transaction ordering and provide finality by approving transactions submitted to them. Like Hyperledger, consensus between notaries does not follow a specific algorithm but is pluggable.

Other blockchain-based applications

Namecoin [3] is a cryptocurrency that provides decentralized DNS and identities. It is a name/value system that allow users to update and register domain names by paying a fee in Namecoin currency. Inspired by Namecoin, Certcoin [39] is designed as a decentralized public key infrastructure (PKI) with the goal to remove Certificate Authorities (CAs) in the existing PKI. Both Namecoin and Certcoin are built on top of Bitcoin, therefore inherit all Bitcoin's properties, including the blockchain structure and proof of work.

Enigma [90] is a decentralized computation platform with privacy that supports data sharing. In Enigma, data is not stored on blockchain but remains locally at client-side and is managed by an off-chain distributed hash table. The blockchain instead only stores

[1] https://www.hyperledger.org
[2] https://github.com/hyperledger/fabric
[3] https://namecoin.org/

data references and access control policies. Enigma supports privacy through a multiparty computation model, enabling secure computation over data without revealing raw data to any party. Since there is no cryptocurrency in Enigma, the incentives are provided based on security deposits or fees.

MedRec [17] is an electronic health record system that is built based on the idea of Enigma. MedRec allows patients to access their medical information as well as to share their records with different providers. The patient-provider relationships are managed via Ethereum smart contracts, which define data pointers and access permissions associated with the records held by providers and allow patients to locate their data. MedRec incentivizes medical researchers and health care stakeholders to participate in mining by introducing a bounty query that can return anonymized medical data as a mining reward to the miners.

2.3 Blockchain and IoT

IBM introduced a blockchain-based architecture called ADEPT (Autonomous Decentralized Peer-to-Peer Telemetry) [24], in which various types of IoT transactions can be conducted on a universal ledger, including device registration, user authentication and contracts between devices. They categorize IoT devices to 3 groups based on their capabilities: light peers which are low-powered and can only perform minimal transactions, standard peers which are more capable to meet blockchain requirements and peer exchanges which are high-end computers owned by organizations and companies [84].

Access control is one of the research domains for blockchain applications in IoT environments that heavily uses the data immutability feature of blockchains. Focusing on an initial stage when new devices are transferred from manufacturers to owners, Hardjono et al. introduce ChainAnchor [45], a privacy-preserving architecture for IoT device commissioning. Using the zero-knowledge proof scheme Enhanced Privacy ID (EPID) [23], ChainAnchor supports anonymous registration of devices without relying on a trusted third party and leverages a permissioned blockchain for record keeping.

Dorri et al. [32] propose a lightweight blockchain-based architecture for IoT data sharing in smart home environments where a private ledger for access control policies is maintained locally by a smart home manager (SHM) while transactions are stored on a public blockchain formed by multiple SHMs and higher-resource devices. To avoid PoW, the authors use a distributed trust protocol for consensus. The lack of incentives and analysis, however, does not guarantee that every party will follow the protocol and behave honestly.

In terms of access control, FairAccess [70] is a blockchain-based access control framework that allows the transfer of access tokens via transactions. However, FairAccess did not address the privacy problem. In order for the network to validate a token, FairAccess requires the access control policy to be included in the token transaction. Since the blockchain is public, the policy can reveal much information about the device and its users, i.e. the type of the device and the type of access. In addition, since the device itself is not capable to store the entire blockchain, it must acquire the data from a trusted node, which means there still needs some kind of centralized authority in the network.

WAVE [15] is a decentralized authorization system to manage permissions to IoT resources. WAVE supports non-interactive delegation with fine-grained access control policies using smart contracts. While privacy is not the focus, WAVE is still able to protect sensitive user information in the permissions using identity-based encryption. However, WAVE uses Raspberry Pi, which has sufficient computation power to run as a blockchain node, in their proof-of-concept. As a result, less powerful devices must still require some trusted gateways to interact with the blockchain network.

ControlChain [72] is a blockchain-based architecture for access authorizations to IoT resources. It supports relationship establishment between users and devices as well as rule and context definition. According to the authors, the resource-limited IoT devices can receive updates from ControlChain and blockchains in general with the support of other devices with the same owner, which means some trusted nodes are still needed. In addition, there is also no evaluation on real IoT devices. In [33], Dukkipati et al. propose a framework that applies

the attribute-based access control model to blockchain. The framework is validated using a use case in smart cities. However, the authors did not address how the IoT devices can be brought to the scenario.

Focusing more on the commercial side, Slock.it [3] provides a blockchain platform that allows users to rent, sell and share their assets using smart contracts. Access to the assets can be granted if certain rules are satisfied (e.g. the renter makes a payment). To support low-power IoT devices, Slock.it introduces a network of nodes called INCUBED that sends signed block hashes to the IoT devices. The nodes are incentivized by making a deposit which they can lose if they are found to cheat by other nodes.

2.3.1 Issues with IoT access control blockchains

Privacy

In public blockchains, all of the blockchain data is visible to everyone. On the other hand, access control information is critical and should not be exposed. Although identities in blockchains are often pseudonyms, they are not anonymous. As a result, user activities can be easily tracked. As discussed in section 2.2.2, solving privacy is already a challenge for regular blockchains, which requires either complicated cryptographic techniques or special hardware. Therefore, with resource-constrained IoT devices, the problem is even more challenging.

Limitation of IoT devices

Most of the mentioned blockchain platforms for IoT access control are based on the same principle: the occurrence of a specific event/transaction on the blockchain needs to be verified before access to an IoT devices can be granted. For example, for Slock.it, a user can rent his/her house by creating a smart contract on a blockchain. By making a payment through the smart contract, the renter can then receive a key/token that later can be used to unlock the house. In this case, the smart lock needs to be able to recognize the transaction between its owner and the renter on the blockchain in order to grant access to the renter.

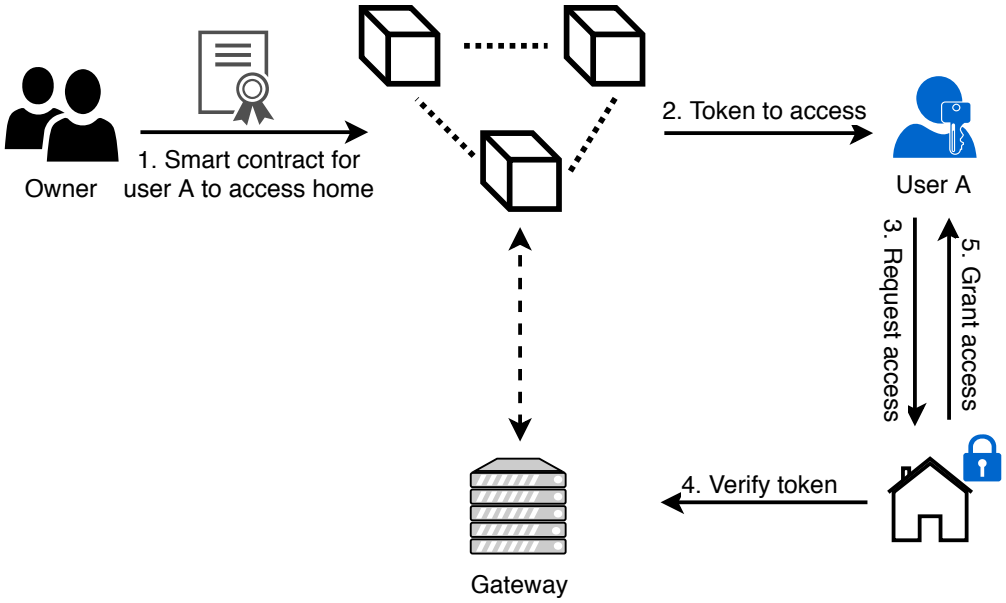


Figure 2.2: A use case where an owner transfers the key to his/her house to user A via a smart contract. The door lock confirms the smart contract via a gateway that connects to the blockchain network and grants access to the user accordingly.

The validation of blockchain data requires nodes to store the entire blockchain, whose size may be a problem. To solve this, Bitcoin also provides Simplified Payment Verification (SPV) [65], which allows nodes that do not have sufficient storage to store only block headers. Although SPV works well for cryptocurrency nodes, it cannot be adapted to IoT devices with very low memory, data storage and bandwidth. Currently in Bitcoin there are over 500 000 blocks with 80 bytes per block header. Ethereum has over 6 000 000 blocks with 508 bytes per block header [4]. Hence, block headers require hundreds of megabytes. In addition, although SPV nodes do not store the complete blockchain, they are still peers that receive block headers from others. On the other hand, low-cost devices often have very limited transmission rates and are not capable of maintaining connections and frequently exchanging data with multiple peers. As a result, resource-constrained nodes have no choice but rely on some more powerful nodes that can interact directly with the blockchain to retrieve data. Figure 2.2 demonstrates the earlier use case of Slock.it where there is a smart home manager that acts as a gateway to inform the smart lock about the contract between

[4]From https://blockchain.com and https://ethstats.net, as of Aug 2018

the owner and renter.

2.4 Summary

In this chapter, we provide a comprehensive survey of state of the art of access control and blockchain technology for IoT. While there have been various mechanisms for authentication and authorization, the key sharing, especially through a chain of delegation, is often neglected, which can hinder user experiences. Additionally, since blockchain is an emerging technology, its applications to IoT domains are still in an early stage. In particular, user privacy and the role of IoT devices in blockchains are still open problems that need to be explore.

Chapter 3

Access Control with Delegation for Smart Home Applications

3.1 Introduction

Motivated by the secnario in Chapter 1, we identify the importance of having flexible access control in smart home environments so that users can create/transfer their permissions in a convenient way. As discussed in section 2.1, traditional access control models such as role-based access control (RBAC) and attribute-based access control (ABAC) are not suitable for IoT. In role-based access control (RBAC), users are assigned to certain roles with different access levels. However, since the behaviors of users can be very arbitrary, it is difficult to define consistent roles that meet all of users' autonomous demands and quickly update policies to adapt with the frequent changes of permissions [44]. In attribute-based access control (ABAC), access is granted to users if they hold a certain set of attributes. For example, in a health database, a boolean expression $(Doctor \vee Nurse) \wedge Psychology$ defines a policy that a medical record may only accessible to a doctor or a nurse in Psychology department. Although ABAC is more flexible than RBAC in terms of defining access control policies, the attributes are also pre-defined and applied to a group of users rather than an individual, which is a common case in smart home environments.

In capability-based access control (CBAC), a capability token with certain access to a

device can be created by the device's owner or an authorization server and then granted to a user, therefore it can provide more flexibility than RBAC and ABAC. However for CBAC, at each step of capability delegation, the parent capability should be included in the new one to make sure the delegated access does not violate the permission hierarchy, therefore as the delegation chain expands, not only the token becomes larger due to the piggyback information, but the number of issuer identities that needs to be verified also increases. This results in high overhead at the device, in terms of both communication and computation.

In this chapter, we design a lightweight delegation mechanism using Bloom filters. A user can easily delegate a permission to another user by adding the corresponding operation to his current Bloom filter as long as the operation satisfies the permission hierarchy of the system. By leveraging the property that items can only be inserted but are difficult to remove from the Bloom filter, we can create a trusted chain within a small packet that a server can easily verify without complicated cryptographic algorithms. Although the Bloom filter has a false positive rate, by setting appropriate parameters, we show that a sufficient security level can be achieved.

3.1.1 Chapter organization

The chapter is organized as follows. Section 3.2 discusses the use of Bloom filters as permissions and limitations of previous work. Section 3.3 presents the details of our protocol for smart home environments. A security analysis is provided in section 3.4. Section 3.5 presents the performance evaluation. A discussion of the protocol is presented in section 3.6. Section 3.7 concludes the chapter.

3.2 Bloom Filter As Delegable Permission

A Bloom filter [21] is a compact probabilistic data structure that can check for membership of a given item. It is represented by a bit array of length m which is initialized to all zeros. An item can be added to the filter by setting k bits, whose positions are generated by

passing the item to k hash functions. A Bloom filter can tell that a given item is definitely not in the set if any of its corresponding k bits is 0. On the other hand, if all of these bits are 1, the item may or may not belong to the set with a false positive rate.

Foley et al. [37] introduces an approach to implement permissions as Bloom filters. A Bloom filter format of a permission x is a filter consisting of all permissions equal or higher than x. By keeping the root permission as a secret, it is easy to create subsequent permissions but difficult to obtain a higher permission. A permission x is considered to be dominated by permission y, which denoted as $x \leq y$ if permission for y also implies permission for x. For example, the Vera lock system supports users the ability to control, monitor (i.e. access logs), configure the lock and receive notifications in the form of 3 roles: administrator, guest and notification-only [13]. While an administrator/owner has full control of the system, a guest may not be allowed to change its configuration. A notification-only user (e.g. a security staff) cannot access the lock but is able to receive notifications such as events or alarms. By this role set up, a permission lattice $P = \{root, control, monitor, notify\}$ can be created as in Figure 3.1 [1], where each permission can be represented by a string or some bit value. Since *root* is the highest permission, all values except *root* can be made public while *root* is kept as a secret between the owner and the lock. Let $\lceil p \rceil$ denote the set of all permissions higher or equal to p, i.e. $\lceil p \rceil = \{x | x \geq p\}$ and $\mathcal{B}(X)$ denote the Bloom filter to which items from X are added. Suppose Carol rents Bob's apartment and is given permission to control the door lock, the Bloom format of her permission will be $\mathcal{B}(\lceil control \rceil) = BF(\{root, control\})$. If she wants her father John to be able to receive notifications from the lock, she can create a notification-only permission by adding $monitor, notify$ and delegate $\mathcal{B}(\lceil notify \rceil) = BF(\{root, control, monitor, notify\})$ to John without knowing *root*. When John requests for notification, he must present his permission to the lock, which then verifies if all values *root*, *control*, *monitor*, $notify$ are presented in the permission. It should be noted that the presence of *root*, *control*, *monitor*, does not

[1] Here we assume control and monitor are independent permissions. The policies to create a permission lattice depends on manufacturers.

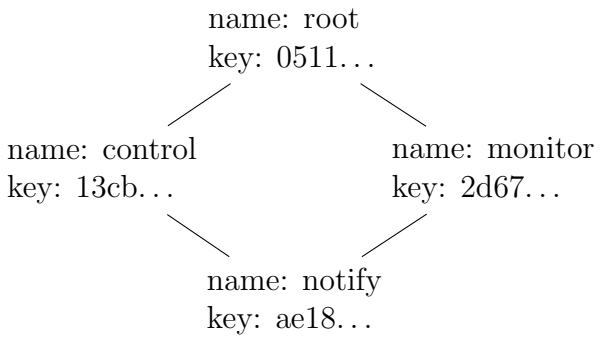


Figure 3.1: Example of a permission lattice of a smart lock

mean John has those permissions; instead it implies that his notification permission was granted by a legitimate user who has authorization to delegate such permission.

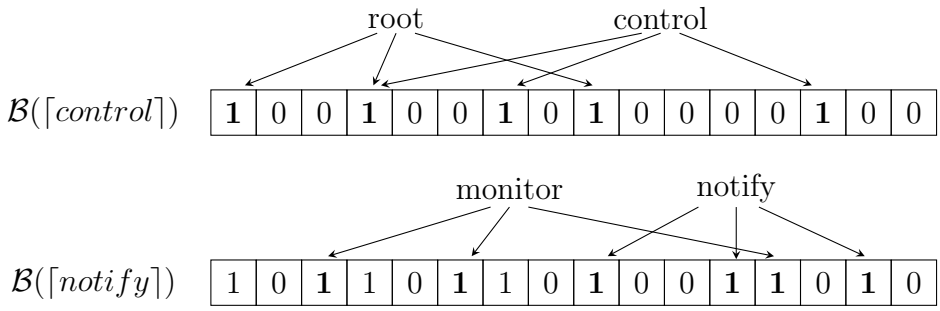


Figure 3.2: Example of how to create 'notify' permission from 'control' permission

The above takes advantage of the non-reversibility property of BFs: items can be added but cannot be removed from the filter as they may share the same bits. Fig. 3.2 describes Carol's $\mathcal{B}(\lceil control \rceil)$ and John's $\mathcal{B}(\lceil notify \rceil)$ using a 12-bit BF. John can try to remove $notify$ from his filter to obtain $\mathcal{B}(\lceil control \rceil)$. However, since *root* and $notify$ share the same bit at position 4, removing $notify$ by setting all of its bits back to 0 will also invalidate *root*. Since *root* is unknown, John cannot know which bits should be retained, thus fails to get the permission. Similarly, Carol is not able to remove *control* as well since it overlaps with *root* at bit 6. The above takes advantage of the non-reversibility property of Bloom filters: items can be added but cannot be removed from the filter as they may share the same bits. Therefore it is difficult to forge a higher privilege from a current Bloom permission. For example, Figure 3.2 demonstrates Bloom permission $\mathcal{B}(\lceil control \rceil)$ and $\mathcal{B}(\lceil notify \rceil)$ using

16-bit Bloom filter where each item is set by 3 bits. Suppose a malicious user who is given $\mathcal{B}(\lceil notify \rceil)$ and wants to obtain $\mathcal{B}(\lceil control \rceil)$, he can do that by removing $notify$ from his filter. However, as seen in the figure, $root$ and $notify$ share the same bit at position 9, therefore removing $notify$ by setting its corresponding bits back to 0 will also invalidate $root$. Since $root$ is unknown, the attacker cannot know which bits should be retained, thus fails to get the permission. Similarly, if one is given the ($\lceil control \rceil$), he/she is not able to remove $control$ as well since it overlaps with $root$ at bit 4.

The probability that there is at least one overlapping bit in a Bloom filter is

$$Pr\{overlap\} = 1 - Pr\{nonoverlap\} = 1 - \frac{m!}{(m-nk)!m^{nk}} \tag{3.1}$$

where m is the length of the filter, n is the number of items in the filter and k is the number of bits used to index an item to the filter.

Although the authors of [37] claim that a high probability of overlapping can be achieved with appropriate m, n, k, in practice it is not enough to secure the filter. Figure 3.3 shows the probability of overlapping when there are 2 items ($n = 2$) with $m = \{512, 1024\}$ and $k = \{1, 2, \ldots, 50\}$. With a 1024-bit filter, k should be at least 45 to have a sufficient probability of overlapping. However, even if there is overlapping, guessing the overlapping bits is still not difficult. If k is known, one can easily guess the number of overlapping bits. Moreover, since there are only 2 items, it is highly likely that only 1-2 bits overlap, resulting in only around 1000 possible cases. To make it difficult enough, the ratio of bits that are set should be at least 15-20%. However, too big k will fill up the filter quickly and thus prevent scalability. We propose a solution to this problem in the next section.

To solve the above problem, we extend [37] by generating all of the items' values dynamically for each delegation so that the same item i will be inserted using two different sets of bits in two different Bloom permissions, depending on the permission information such as user ID, permission name, etc. As a result, any attempt to remove an item will not be successful since a valid permission would use a different set of bits than the one resulted

from the removal. We provide more details in the following section.

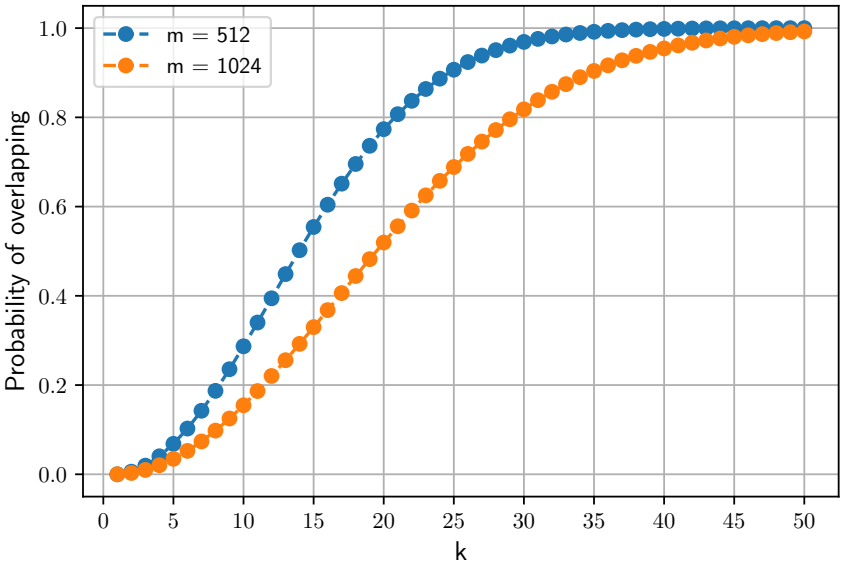


Figure 3.3: Probability of overlapping with 512 and 1024-bit Bloom filters containing 2 items

3.3 Delegable permissions

3.3.1 Preliminaries

We consider 4 entities in our domain:

- Owner: can own multiple resource devices and have the highest authorization to those devices.

- Resource device: a device that provides certain service to users with valid permissions. It has a built-in permission lattice to represent the access control policy. Each node in the lattice is defined by a pair $\{name, key\}$ (example in Figure 3.1) whose key is kept secret between the owner and the device. This is different from the original Bloom permission in [37] where only the value of the highest permission is secret.

- Normal user: who is not the owner and can obtain an access right to a resource from an owner or an authorized user. If the permission is given by an authorized user, it

needs to be activated at the device before usage.

- Authorized user: is given authorization (by the owner or another authorized user) to issue lower permission than his/her own permission to another user. In order to request a certain access to a resource, a user needs to present a valid certificate. The certificate can be issued by an owner or an authorized user, however in the latter case, the requested access level cannot be higher than what the authorized user is holding.

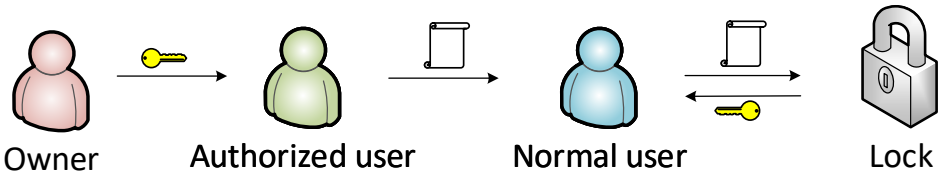


Figure 3.4: Delegation procedure

3.3.2 Overview of the delegation

Our scheme involves 5 procedures:

- $\mathcal{B}_p \leftarrow \texttt{PermGen}(p, id, t)$: to issue to user id a BF permission $\mathcal{B}_p$ that expires at time t.
- $\mathcal{B}_p^{del} \leftarrow \texttt{DelPermGen}(p, id, t)$: to issue to user id a delegation permission $\mathcal{B}_p^{del}$ that expires at time t. Users can use $\mathcal{B}_p^{del}$ to issue authorization certificate $\mathcal{B}_q^{auth}$ where $q < p$.
- $\{0 \text{ or } \mathcal{B}_q\} \leftarrow \texttt{Activate}(B_q^{auth})$: to check an authorization certificate $\mathcal{B}_q^{auth}$ and return BF permission $\mathcal{B}_q$ if the certificate is valid, otherwise return 0.
- $\{0 \text{ or } 1\} \leftarrow \texttt{Verify}(\mathcal{B}_p)$: to check a BF permission $\mathcal{B}_p$ and return 1 if the permission is valid and grant access accordingly, otherwise return 0.
- $K \leftarrow \texttt{KeyDerive}(B, salt)$: to derive a key K from a BF permission $\mathcal{B}$. Here the BF permission is used as master key or key material to generate a one-time session key K for secure message exchange between the user and device.

- $\texttt{Enc}(K|m)$ and $\texttt{Dec}(K|c)$: symmetric encryption and decryption algorithms using key K.

The reason not to use BF permissions directly as access tokens but for encryption keys is three-fold. First, a BF permission should not be presented to the device in plain text to avoid replay attacks. Second, an attacker may create a filter with many 1s to increase the probability of matching. However, by using symmetric keys, the device regenerates the BF permissions by themselves, which means there will be no redundant 1s, hence that kind of attack can be avoided. Lastly, an access request is often associated with certain data. For example, in a request to lock/unlock a door or change the AC temperature, the lock action and the temperature are sensitive data that may expose user privacy. Therefore, using BF permissions as keys both helps to avoid revealing the permission as well as protect user data. Table 3.1 provides notations used in our scheme.

Table 3.1: Notations and definitions

$(P, \leq)$	a permission lattice
$\lceil p \rceil$	set of all permissions higher than or equal to p i.e. $\lceil p \rceil = \{x \in P \mid x \geq p\}$
$\lfloor p \rfloor$	set of p's successor permissions, i.e. permissions lower than p, i.e. $\lfloor p \rfloor = \{x \in P \mid x < p\}$
$\mathcal{B}(X)$	Bloom filter to which items from set X are added
$\mathcal{B}_p$	BF permission that allows any access lower than or equal to p, i.e. $\mathcal{B}_p = BF(\lceil p \rceil)$
$\mathcal{B}_p^{del}$	BF delegation permission that is used to create permissions lower than p
$\mathcal{B}_p^{auth}$	BF authorization certificate for permission p
K_p^{auth}	authorization key derived from $\mathcal{B}_p^{auth}$

3.3.3 Bloom filter index

To distinguish BF permissions of different users, we define a public value $PID = \{perm\_name, userid, exp\_time\}$ that serves as a unique identifier of a permission. For exam-

ple, $\{control, Alice, 191231\}$ identifies Alice's *control* permission, which expires on Dec 31, 2019.

To insert a permission to a BF, we apply the secure index mechanism [43]. Given a pseudo-random function f, the insertion of permission p to BF permission $\mathcal{B}$ with id PID works as follows:

1. Compute $x = f(p.name, p.key)$
2. Compute $y = f(PID, x)$
3. Divide y to k chunks $\{y_1, \ldots, y_k\}$, each of which serves as an index to the Bloom filter.

The permission generation is described by the procedure `PermGen` in Algorithm 1.

Algorithm 1 Bloom permission generation

function PermGen(p, id, t)
  $PID \leftarrow p.name||id||t$
  $\mathcal{B}_p \leftarrow \emptyset$
  for each x in $\lceil p \rceil$ **do**
    $y \leftarrow f(PID, f(x.name, x.key))$
    Insert x to $\mathcal{B}_p$ using y
  end forreturn $\mathcal{B}_p$
end function
function DelPermGen(p, id, t)
  $PID \leftarrow p.name||id||t$
  $m \leftarrow f(PID, p.key)$
  $\mathcal{B}_p^{del} \leftarrow \emptyset$
  for each x in $\lceil p \rceil$ **do**
    $x' \leftarrow x$
    $x'.key \leftarrow f(m, x.key)$
    $y' \leftarrow f(PID, f(x'.name, x'.key))$
    Insert x' to $\mathcal{B}_p^{del}$ using y'
  end for
  $\lfloor p' \rfloor \leftarrow \emptyset$
  for each x in $\lfloor p \rfloor$ **do**
    $x' \leftarrow x$
    $x'.key \leftarrow f(m, x.key)$
    $\lfloor p' \rfloor \leftarrow \lfloor p' \rfloor \cup \{x'\}$
  end forreturn $\mathcal{B}_p^{del}, \lfloor p' \rfloor$
end function

3.3.4 Protocol Details

Permission delegation

Figure 3.5 depicts our delegation protocol. In the first delegation from the owner, Alice is transferred permission a in the form of Bloom filter $\mathcal{B}_a$. If Alice is authorized for delegation, she is given another Bloom filter called *delegation permission* $\mathcal{B}_a^{del}$, by which she can delegate parts of her rights by inserting lower permissions than a to it. The generation of delegation permission, described by the procedure `DelPermGen` in Algorithm 1, is identical to the normal Bloom permission, except that it works on a new set of permission values, which are derived from the original ones by hashing them with the permission ID. The set of successor permissions of a but with the new derived values is then transferred to Alice. Hence, given $\mathcal{B}_a^{del}$ and $\lfloor a' \rfloor$, Alice can create a sub permission, e.g. $b < a$, by adding corresponding items to $\mathcal{B}_a^{del}$. Since this new Bloom filter is created by Alice who is an authorized user, we call it an *authorization permission* and denote it as B_b^{auth} to distinguish it with the delegation permission $\mathcal{B}_a^{del}$ which is created by the owner.

It is noteworthy that any delegation with the same permission b from Alice will result in the same authorization permission $\mathcal{B}_b^{auth}$, therefore to make each delegation unique, Alice chooses a random secret x and computes an *authorization key* K_b^{auth} that is derived from x and B_b^{auth}, i.e. $K_b^{auth} = f(\mathcal{B}_b^{auth}, x)$. In the second delegation to Bob whose ID is id_B, Alice generates an authorization certificate *cert* by encrypting the new Bloom permission ID associated with Bob, i.e. $\{b, id_B, t_B\}$ together with the value x and sends the certificate together with key K_b^{auth} to Bob. The use of both certificate and authorization key is two-fold. The encryption of Bob's permission ID by Alice's key prevents Bob from altering the information. On the other hand, sending the certificate alone is vulnerable to replay attack, when any malicious party who is able to overhear the message can just resend it to obtain Bob's permission. Bob's authorization permission serves as a temporary session key that is only known to Bob, thus can prevent the attack.

Owner		Alice		Bob
To delegate a, expiring at t_A:				
$\mathcal{B}_a \leftarrow \texttt{PermGen}(a, id_A, t_A)$	$\xrightarrow{\mathcal{B}_a}$	Save $\mathcal{B}_a$		
If is_authorized(id_A):				
$\{\mathcal{B}_a^{del}, \lfloor a' \rfloor\} \leftarrow \texttt{DelPermGen}(a, id_A, t_A)$	$\xrightarrow{\mathcal{B}_a^{del}, \lfloor a' \rfloor}$	Save $\mathcal{B}_a^{del}, \lfloor a' \rfloor$ To delegate b, expiring at t_B: $\mathcal{B}_b^{auth} \leftarrow \mathcal{B}_a^{del} \cup \mathcal{B}(\lceil b' \rceil - \lceil a' \rceil)$ Choose $x_b \in \{0,1\}^*$ $K_b^{auth} \leftarrow \texttt{KeyDerive}(\mathcal{B}_b^{auth}, x_b)$ Choose $x_a \in \{0,1\}^*$ $K_a \leftarrow \texttt{KeyDerive}(\mathcal{B}_a, x_a)$		
		$cert_B^A \leftarrow \texttt{Enc}(K_a \vert \{b, id_B, t_B, x\})$	$\xrightarrow{K_b^{auth}, cert, x_a}$	Save $K_b^{auth}, cert_B^A, x_a$

Figure 3.5: Permission delegation

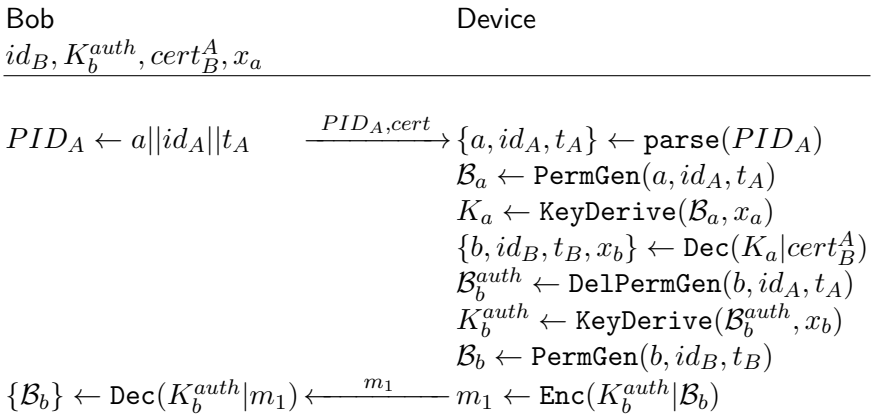


Figure 3.6: Permission activation

Activation protocol

Figure 3.6 depicts the activation protocol, which involves the normal user and the device. To activate permission b given by Alice, Bob first sends his authorization certificate from Alice $cert_{Alice \rightarrow Bob}$ and Alice's corresponding permission information $perm\_id_A$. The device hence can compute Alice's permission $\mathcal{B}_a$, decrypt the certificate to get all information needed to compute the authorization key K_b^{auth} and Bob's permission B_p, which is then sent back in encrypted form by K_b^{auth}.

Verification protocol

As the activation protocol, verification also involves secure message exchange using the Bloom permission as an encryption key. To request a service under permission b, Bob encrypts the request data (for example, open/close door or set AC temperature to to X) and sends it with his public permission information, based on which the device computes the corresponding key $\mathcal{B}_b$ to decrypt the message. If the permission is valid, the device executes the request and sends back the encrypted response to Bob. Both parties can also use challenges for mutual authentication as in the activation protocol as well. The protocol is depicted in Figure 3.7.

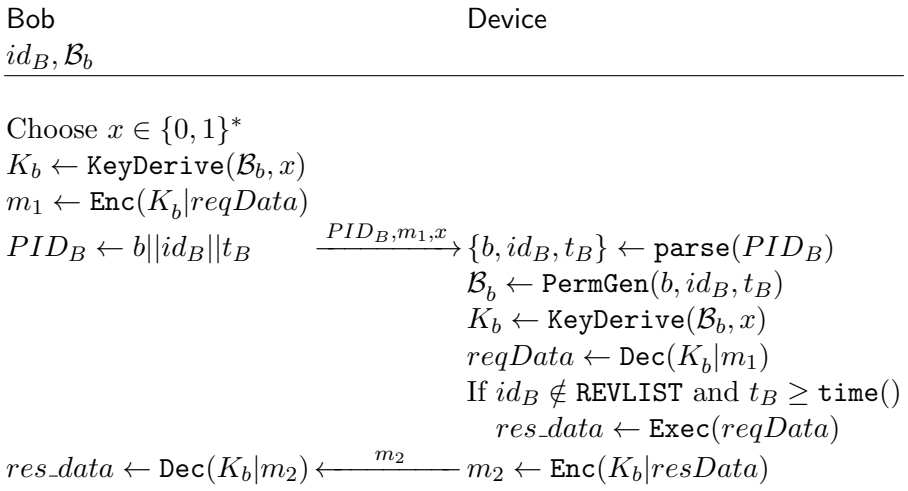


Figure 3.7: Permission verification

3.4 Security Analysis

Our analysis is under an assumption that the exchange of Bloom permission between users is secure, i.e. no malicious entity can overhear or steal the permissions. This is a reasonable assumption as the exchange is carried out on users' smart phones or PCs, which are capable for secure exchange.

Since Bloom permissions are used as symmetric encryption key, the security of our scheme depends on the difficulty of forging such keys. As stated in section III, there are 2 possible attacks that are analyzed in [37] to infer a higher permission from 1 or more permissions: (1) removal attack where the attacker tries to remove items from the Bloom filter and (2) aggregation attack by computing the intersection between 2 or more filters. The removal attack doesn't work in our scheme since each item is hashed with the Bloom permission id that contains the lowest permission. For example, given a Bloom permission $\mathcal{B}_b$, an attacker wants to remove b to obtain $\mathcal{B}_a$ where $a > b$. However in $\mathcal{B}_b$, permission a is associated with $b.name$ while in a valid $\mathcal{B}_a$ it will be associated with the name of itself. Therefore any attempt of removal will result in a meaningless Bloom filter as it is no longer match with the attacker's desired condition. Likewise, aggregation or collusion between users

would not work either and as a result, the only possibility left is to do an exhaustive search to find the filter.

The difficulty of the search depends on 3 parameters of the Bloom filter: the filter size m, the number of items n and the codeword length k. Since the bits are generated from pseudo random functions, we assume that they are under a uniform distribution. After inserting n items using k bits per item, the probability that a bit $b_i, i = 1 \dots m$ is still 0 is

$$P(b_i = 0) = \left(1 - \frac{1}{m}\right)^{kn} \tag{3.2}$$

Therefore the false positive rate of Bloom filter is

$$fpr = \left[1 - \left(1 - \frac{1}{m}\right)^{kn}\right]^k \approx \left(1 - e^{kn/m}\right)^k \tag{3.3}$$

In addition, the average number of bits that is set to 1 after n insertions is

$$s = m \times P(b_i = 1) = m\left[1 - \left(1 - \frac{1}{m}\right)^{kn}\right] \tag{3.4}$$

hence the search space to find a Bloom permission is $S = \binom{m}{s}$.

Equation (3.3) shows that as the delegation chain expands, the false positive will increase since more items will be added to the filter. Therefore, given a fixed m, we want to find a value of k that can provide large enough search space when n is small while still keep a reasonable false positive rate when n is large. We choose 2 as the lower bound of n since *root* is always present in any BF permission[2]. For the upper bound that basically is the total number of nodes in the permission lattice, we choose 20 by intuition as we believe an IoT device would not have more than 20 different types of permissions. Table 3.2 shows fpr and search space S and with 3 sizes of the BF and different values of k. While fpr is monotonically increasing, S will start to decrease when s exceeds $\lceil \frac{m}{2} \rceil$. If we take the difficulty of finding a 128-bit key (whose search space is 3.4×10^{38}) as the minimum security

[2]Since only the owner holds secret keys, it is unnecessary for them to use BF permission consisting of only *root* to control their devices, thus we omit the case when n = 1.

threshold, then according to the table, except for $m = 256$ when S drops to 3.15×10^{30} with $n = 20$, k in the range of $[16, 32]$ can provide both good false positive rate and search space.

The fpr is the probability that an item is recognized despite not being in the set. Thus, it implies the probability that a user can successfully claim his given permission to be another one that he is not granted. It should be noted that the calculated false positive rate by eq. (3.3) is only for 1 item, while a valid BF permission requires all items to be recognized without any redundant bit. To claim a false BF permission, not only the lowest but all n items should be false positives since every item is associated with the lowest one when the permission is generated. Thus, for a BF permission containing n items to be falsely accepted, the actual probability is fpr^n, which exponentially increases to be negligible as n increases.

Table 3.2: Search space and false positive rate

BF size (m)	#bits per item (k)	S with n=2	fpr with n=20
256	5	2.79×10^{17}	0.00353568
	10	2.37×10^{28}	0.00219446
	15	1.91×10^{37}	0.0038438
512	5	3.12×10^{20}	1.76×10^{-04}
	10	4.32×10^{35}	1.25×10^{-05}
	15	1.87×10^{47}	5.04×10^{-06}
1024	5	3.34×10^{23}	6.97×10^{-06}
	10	5.48×10^{41}	3.09×10^{-08}
	15	5.00×10^{57}	1.18×10^{-09}

A Bloom permission with 512 bits or more may not be suitable to be used directly as an encryption key (for example, AES only support 256-bit key at maximum). In this case, a more compact form such as a hash of the permission with salts can be used instead, with the raw permission being included in the message.

3.5 Implementation

We implement the protocol on an Arduino MKR1000 board that runs as a smart device and a Java client that runs on a Dell laptop. The Arduino board has a 32-bit low power ARM microcontroller, 256 KB of flash and 32 KB of SRAM. The key size and Bloom filter size are both 256 bits. We choose HMAC-SHA256 as the pseudo random function f and AES-256 with CBC mode as the encryption algorithm. Each item is inserted using 32 bits of indices. The messages are wrapped in a JSON object for human-readability. We run the protocol over TCP but it can be run over an IoT RESTful protocol, for example CoAP.

The permission lattice is implemented as a directed acyclic graph using adjacency list. In our first test, the Arduino simulates a door lock with the permission lattice shown in Figure 3.1, which has 4 nodes in total. The whole program takes 45 KB of flash. Figure 3.8 and 3.9 are the examples of an activation request and an access request in JSON format. Table 3.3 shows the processing time of *control* and *notify* permissions under the two types of requests. Compared to other schemes such as [80] and [66], our results are significantly smaller. While [80] is capability-based and [66] is attribute-based, they both rely on Elliptic Curve Cryptography with point multiplication as the base operation. Using the ArduinoLibs [3] library, we found that a point multiplication takes 700 ms on our hardware. On the other hand, our scheme can take less than 100 ms.

To test for scalability, we also implement a simple lattice with 20 nodes and measure the processing time with different number of nodes added to the filter. The average processing time is given in Table 3.3b. Though the activation process has to perform more operations than the access request, it still takes less than 200 ms even with a large number of items present in the filter.

[3] https://rweather.github.io/arduinolibs/crypto.html

```
{
  "type": "access",
  "userid": "Alice",
  "perm_name": "control",
  "expiry": 20180430,
  "req_data": "tnJB7Pr4kpONg...",
  "IV": "rxtOeJEQ2..."
}
```

Figure 3.8: JSON format of an access request

Figure 3.9: JSON format of an activation request

Table 3.3: Processing time of request

(a) With the simulated smart lock

	control	notify
Access request	32 ms	38 ms
Activation request	N/A [a]	68 ms

[a] Since *control* can only be given by the owner, there is no activation needed.

(b) With different number of items

n	10	12	14	16	18	20
Access	57 ms	62 ms	69 ms	73 ms	78 ms	83 ms
Activation	121 ms	135 ms	148 ms	161 ms	175 ms	188 ms

3.6 Discussion

In a certificate-based approach, a parent certificate must be piggybacked in every delegation, hence a user needs to send a number of certificates to prove its authorization. However,

in our approach, the size of a Bloom permission is always a constant despite the expansion of the delegation chain. The scalability also supports flexible and fine-grained access control, since users can add various operations to the filter, as long as they do not conflict or violate the permission hierarchy. While the lattice may consume most of storage if the number of nodes is large, all of the key values can be generated from a single seed to save space.

We do not have a user registration phase as user IDs are created by owners and authorized users at each delegation. Since any permission delegated by a non-owner user has to be activated first, a list of users can be managed by the owner by having the device upload new user IDs to a smart home manager or a cloud server. Here we mainly focus on cases like the scenario in Chapter 1, when Bob the owner is not present in his house, thus cannot (and may not need to) anticipate who is going to use his devices since Dave can always bring in his friends and guests. In such cases, it is more important to have control over what type of operations can be performed on the devices, which is handled by Bloom permissions, rather than the identities of users. On the other hand, Bob and authorized users like Dave can observe new users via the user list.

Although our scheme is time-based revocation, i.e. the permission is automatically revoked on the expiry date, an emergent revocation of a user can be done by adding the user's ID to a revocation list. The owner can also revoke a set of users by replacing the set of keys of the permission lattice with new values. For example, if Bob sees that Dave has violated their contract, he can reset the permission keys, thus invalidate Dave's permission as well as all of the delegated permissions.

Since the authorization chain is implied in the Bloom permission, the device does not need to maintain or update any access control list except a revocation list. In case the number of users is so big that the device is not capable to store such list, it can ask for a policy enhancement point (PEP) in the network. Though a PEP seems to defeat our purpose of decentralization, we believe it is necessary to have a centralized authority in a large-scale domain such as an office building.

The Bloom filter is also useful for secured device discovery. Sometimes a device may want to expose itself only to users who have permissions. In this case a new user can use his delegated keys to discover all devices he has access to when he first joins the home network by combining a few bits in each of his permissions to a single filter, then broadcasting it for discovery. The discovery protocol can be similar to the Master Key [89] where users can choose a number of bits they want to send. During the discovery, the user can be informed with device information such as ID and permission lattice.

3.7 Conclusion

We propose a lightweight distributed authorization protocol with support of delegation chain for home environments. Certain access right to a smart device is transferred in form of a Bloom filter with secured hashing to prevent the permission from being forged. By leveraging the inability to remove items in the Bloom filter, a user cannot recreate a permission higher that what he/she is holding, but still is able to transfer lower permissions. Our protocol does not only require no access control list, but can also be implemented with little encryption, in comparison to certificate-based approach, thus is suitable for resource-constrained devices.

In the next chapter, we explore the use of blockchains, which allow secure management and cross-domain sharing of access capabilities, for wider applications that are not limited to smart homes. We introduce CapChain, a privacy perserving blockchain-based access control framework.

Chapter 4

CapChain - A Privacy Preserving Access Control Framework Based on Blockchain for Pervasive Environments

4.1 Introduction

With the growth of IoT devices, a user can hold keys/credentials to several devices in different domains. In most cases, the distribution of those keys to sub-users is carried out through a trusted server, which can be a potential single point of failure. On the other hand, it is also inconvenient for users to have a separate account to store their keys for each application they are using. Therefore, it is desirable to have a global framework that supports reliable and flexible cross-domain key sharing.

With blockchain, a reliable access control list can be effectively created as data cannot be modified or deleted once it is published. However, public blockchains also suffer from a privacy problem since all can access data on the blockchains, which is not a favorable situation for access control as the usage of devices should only be visible to users within their private networks.

We propose CapChain - an access control framework based on blockchain that allows

users to share and delegate their access rights easily to devices in public but still maintain privacy. Our idea is to treat the access rights, which are called capabilities in CapChain, as types of assets that can be transfered between users via transactions. We assume that every IoT device in CapChain has one or several ultimate owners who have full control over the device and are able to generate capabilities based on its own access control policies. The capabilities then can be delegated to other users via transactions on a public blockchain that serves as a public immutable access control list that records the capability delegation. In order to grant access to a certain user, the device needs to verify the existence of the relevant transaction in the blockchain. To protect privacy, we apply multiple techniques to hide sensitive information that can only be visible to relevant users. Our delegation system has the following characteristics:

- Each user's capability has an expiry date for auto revocation. As a rule of delegation, a receiver's capability cannot expire later than the expiry date of the sender's capability.
- Besides auto-expired capabilities, users can still track or revoke the whole chain of delegation originated from themselves if necessary.
- Identities of sender and receiver as well as transaction information are protected.
- Users need only one single master account to receive capabilities from different domains.

With anonymous transactions, CapChain allows users to share their home devices with other users as well as receive keys of other places such as their offices through a global network without worrying about their private information being revealed to the public, thus enables scalability to multiple domains and organizations.

4.1.1 Chapter organization

We present an overview of CapChain and a review of cryptographic techniques that we adopt from existing cryptocurrency systems in section 4.2. The details of CapChain is

provided in section 4.3. Section 4.4 discusses the performance of CapChain. Finally, the chapter is concluded by section 4.5.

4.2 System overview

4.2.1 Capability

We define a capability (denoted by CAP) as a token that represents some access right to an IoT device, and is encrypted by a secret key shared between the device and its owner. For example, a user may have an "operate" capability to open/close a smart lock and another "configure" capability to set up the lock. Capabilities can only be generated and published to the blockchain by the device's owner, then are transferred between users via transactions. A capability can be considered an asset type/currency as in crypto-currency systems, so the owner will have an infinite amount of each capability. On the other hand, capabilities of normal users always have an expiry that cannot be later than the expiry of the sender's capability. Therefore, the expiration time can be treated similarly to the amount of currency. In other words, if we ignore the context of capability, the fact that Alice, who is holding a capability CAP that will be expired in 10 days, can only delegate the same capability that expires in no more than 10 days, is the same as Alice has 10 USD, and thus can only transfer 10 USD at maximum.

4.2.2 Blockchain and capability transactions

The blockchain stores all of capability transactions. It serves as an access control list that records the proof that a user is holding a certain capability and when it will be expired. There are 3 types of transactions:

- $\mathsf{tx}_{\mathsf{publish}}$
- $\mathsf{tx}_{\mathsf{delegate}}$
- $\mathsf{tx}_{\mathsf{confirm}}$

New capabilities are published to CapChain via $\mathsf{tx}_{\mathsf{publish}}$. To delegate capabilities, users need to post a $\mathsf{tx}_{\mathsf{delegate}}$. Just like a Bitcoin transaction, $\mathsf{tx}_{\mathsf{delegate}}$ also consists of input (aka the source capability) and output (aka the destination). $\mathsf{tx}_{\mathsf{publish}}$ can be considered as a special $\mathsf{tx}_{\mathsf{delegate}}$ but without the input. In a $\mathsf{tx}_{\mathsf{delegate}}$ transaction, users has to specify a new expiry date of the delegated capability. As a rule of delegation, the new date cannot be later than the old one from the input capability. To prevent arbitrary delegations without actual usage of capabilities, a $\mathsf{tx}_{\mathsf{delegate}}$ needs to be confirmed by the device or an authority (e.g. the owner) before being used in a further delegation. The confirmation is posted via $\mathsf{tx}_{\mathsf{confirm}}$. Figure 4.4 depicts a chain of delegation, starting from Alice who is the owner of $\mathsf{CAP1}$, then to Bob and Carol.

4.2.3 Authorization workflow

We address 3 entities in the network:

- IoT devices that have low computation and low storage. Depending on the communication type, the device may or may not have direct access to the Internet. In the latter case, it has to rely on a local proxy to look for transactions on the blockchain.

- Proxy: higher-resource devices, such as a PC at home or a server in office. Each domain has at least 1 proxy that acts as an actual node on CapChain.

- Mobile devices such as smartphones or laptops that act as wallets to receive and transfer capabilities.

To request for access, a user can prove his/her capability possession by signing the corresponding transaction. The device then will inquire to the blockchain (either directly or via local proxy) for the presence of the transaction and the capability and grant access accordingly.

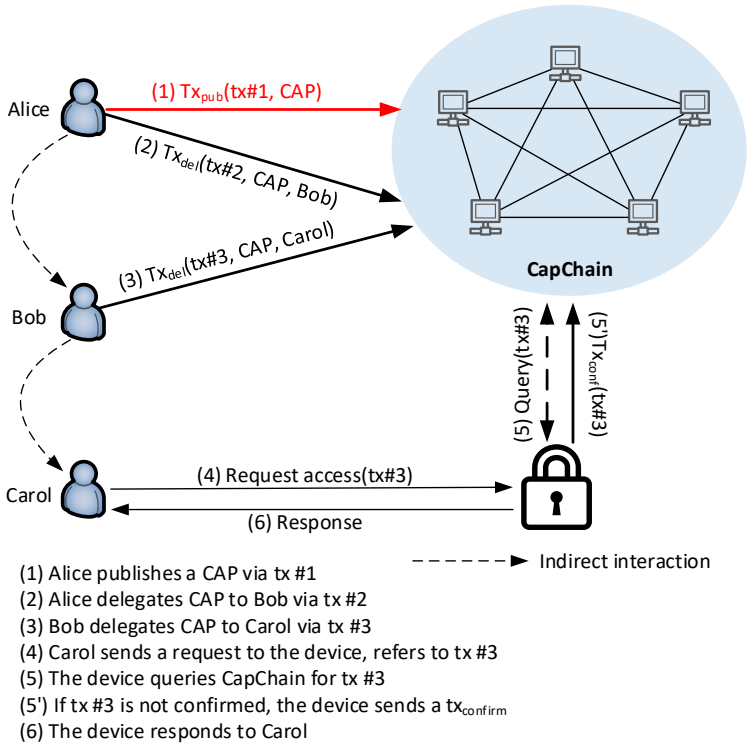


Figure 4.1: System overview

4.2.4 Transaction linkability

Unlike the current anonymous crypto-currency systems where there is no way for the sender to trace where their sent money will be spent further, our users should be able to control all of the delegations made by their successors and revoke them if necessary. Therefore, we utilize *transaction depth* - the number of hops from the current transaction to the root one made by the owner and design a simple hash chain so that given the current transaction depth, users can easily find all of the subsequent transactions. Generally, if a transaction is made anonymously, it is impossible to infer the depth of it. However, since we now attach the depth as a field in the transaction, it should also be hidden from the public network.

In summary, the following information should be obfuscated:

- The identities of the sender and receiver

- The identity of the capability
- The expiry date of the capability
- The transaction depth

4.2.5 Privacy issues with cryptocurrency blockchains

A typical Bitcoin transaction has 2 components:

- Input: a reference to an output from a previous transaction.
- Output: an output specifies the receiver and the amount of money to be sent.

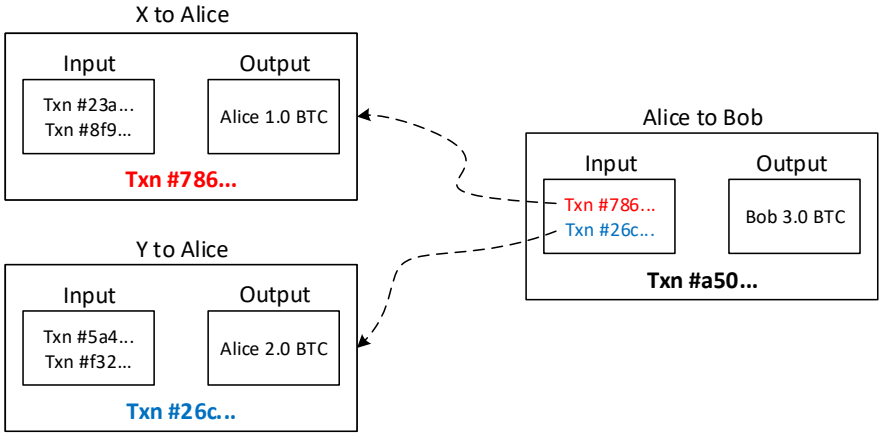


Figure 4.2: Alice sends to Bob 3 Bitcoins, whose inputs refers to two previous outputs.

In order for a transaction to be accepted by the network, it must satisfy the following: 1) the sender is the valid owner of the inputs; 2) all inputs are referred to valid, unspent outputs; 3) all inputs must be consumed by the outputs, which means the total currency value from the inputs and outputs must be equal. Figure 4.2 depicts a sample transaction from Alice to Bob with an amount of 3.0 Bitcoins, in which the input consists of 2 previous outputs destined to Alice, a 1.0 Bitcoin and a 2.0 Bitcoin. Later if Bob wants to spend these 3 Bitcoins, he can make a transaction that refers to his output in Alice's transaction. In fact, Alice and Bob do not use their names but a pseudonym or so-called *address* that

is derived from their public keys to receive bitcoins. However, if someone happens to know the user's identity associated with an address, they can link all transactions that are made from/to this user.

In cryptocurrency systems, the privacy problem can be categorized to three main concerns [4]:

- Privacy of identity: the identities of sender and receiver should not be revealed.
- Privacy of amount: the amount to be transferred should not be revealed.
- Privacy of history: the inputs used in a transaction should not be traced to the previous transactions that created them as well as linked to future transactions.

For the privacy of amount, Maxwell et al. [57] developed "Confidential Transactions," a protocol based on Elliptic Curve Cryptography (ECC) for encrypting the input and output amounts of a transaction in a way that still allows the network to validate that the transaction balances. To tackle all of the three privacy issues, the cryptocurrency Monero [1] introduces Ring Confidential Transactions (Ring CT) protocol [67], a combination of the Confidential Transactions protocol [57] and ring signatures [40]. Monero allows a sender to mix his transaction input with different inputs from others and creates a ring signature that can prove that he knows the private key to one of the public keys in the ring. As a result, the transaction is unlinkable and the amount can be hidden as well.

Since we adopt Monero's Ring CT to CapChain, a brief overview of Ring CT is presented as follows.

ECC terminology

- E: an elliptic curve in the form of $y^2 = x^3 + ax + b$
- G: a generator (or base point) on E

[1] https://www.getmonero.org/

- l: a prime order of the base point G

- $\mathcal{H}$: a cryptographic hash function

- A pair (P, p) is a ECC public-private key pair if $p \in [1, l-1]$ and P is a point such that $P = p \times G$.

Ring Confidential Transactions

For Monero, the privacy of amount requires that the amounts in the inputs and outputs should not be presented in plain text but some encrypted forms while still allow the network to check the balance (i.e. $\sum inputs = \sum outputs$). In other words, a homomorphic encryption C is required to transform the amount values, such that $C(a) + C(b) = C(a+b)$. In Ring CT, in order to hide an amount a, a transaction includes a Pedersen commitment [71] to a, which is computed as follows :

$$C(a, x) = xG + aH \tag{4.1}$$

where x is a secret blinding factor and H is a point on E such that it is difficult to find the discrete logarithm of H with respect to G (i.e. a value n such that $H = nG$). For simplicity, suppose a transaction has 1 input amount a and 2 output amounts a_1, a_2, the commitments to those input and outputs are:

$$C_{in} = xG + aH$$

$$C_{out,1} = x_1G + a_1H$$

$$C_{out,2} = x_2G + a_2H$$

where $x - x_1 - x_2 = z$. If the balance condition is satisfied, i.e. $a = a_1 + a_2$ we will have

$$C_{in} - \sum C_{out,i} = (x - x_1 - x_2)G + (a - a_1 - a_2)H = zG \tag{4.2}$$

which is a commitment to 0 with secret key z and public key zG. Only the sender can know z and sign for this commitment, therefore he can hide his identity by mixing his input with a number of other inputs from the public blockchain to produce a ring signature that proves he is indeed one of the members in the ring. The amount and blinding factor are also encrypted and transfered to the recipient via an ECDH key exchange. To prevent double spending, the signature also includes a key image $I = p\mathcal{H}_p(P)$ where $\mathcal{H}_p(P)$ is a hash to point function, so that any attempt to spend the input twice can be detected by the re-appearance of I in the signature.

However, checking (1) is not safe enough to guarantee that the transaction is valid since the equation still holds for negative values due to overflow. For example, from an input of 5 coins, two illegal outputs can be -3 and 8. In this case, the validation is still successful as $5 = -3+8$, but 8 coins have been created out of nowhere. Confidential Transactions prevent this by *range proof* - a proof that a committed value must be within a valid range without disclosing the actual value. The idea of the proof is also based on ring signature. Readers can refer to the original papers [57, 83, 67] for more details.

4.3 System details

4.3.1 Capability publication

At this phase the owner initializes all capabilities associated with his devices and sends them to the blockchain via $\mathsf{tx}_{\mathsf{publish}}$ transaction. For scalability, an off-chain capability storage similar to [90] can be used to store capabilities so that the blockchain only needs to maintain a reference to each capability. Besides the reference, a capability is also mapped to a unique point $\mathcal{M}$ on an elliptic curve. This point will serve as a capability ID that is used in all delegation transaction.

It is required that the discrete log with respect to two capability points $\mathcal{M}_1, \mathcal{M}_2$ must be difficult to find. Otherwise it would be possible to convert a capability to another. We

compute the capability point $\mathcal{M}$ as follows:

$$\mathcal{M} = \mathcal{H}(\mathsf{CAP}, \mathbf{s})G \tag{4.3}$$

where $\mathbf{s}$ is a shared secret between the owner and the device. The owner can prove his/her ownership by signing the transaction with the private key of $\mathcal{M}$.

It is possible that a malicious user can flood the network with fake capabilities or send those to other users. For the formal case, the publication can be restricted to only valid devices by having manufacturers co-sign the transaction. ChainAnchor [45] provides a mechanism for device commissioning that can be applied to this scenario. For the latter case, if the transfer of the capability involves a payment, then the payment can be locked until the receiver successfully activates the capability.

4.3.2 Transaction destinations

Our delegation involves 3 types of public keys:

- User's primary address: each user joining the blockchain has a public key as his/her primary address.
- Domain's address: a public key V that represents a domain/organization. The corresponding private key v is known to both the local proxy and all devices within the domain.
- One-time sub address: an address that is derived from primary address in order to receive anonymous transactions. A transaction is destined to two addresses: the user and the domain the contains the device.

Deterministic user's sub address

Since it is desirable not to expose the recipient's ID, the primary address shouldn't be used as the destination of the transaction. Unlike CryptoNote or Monero that use a one-time

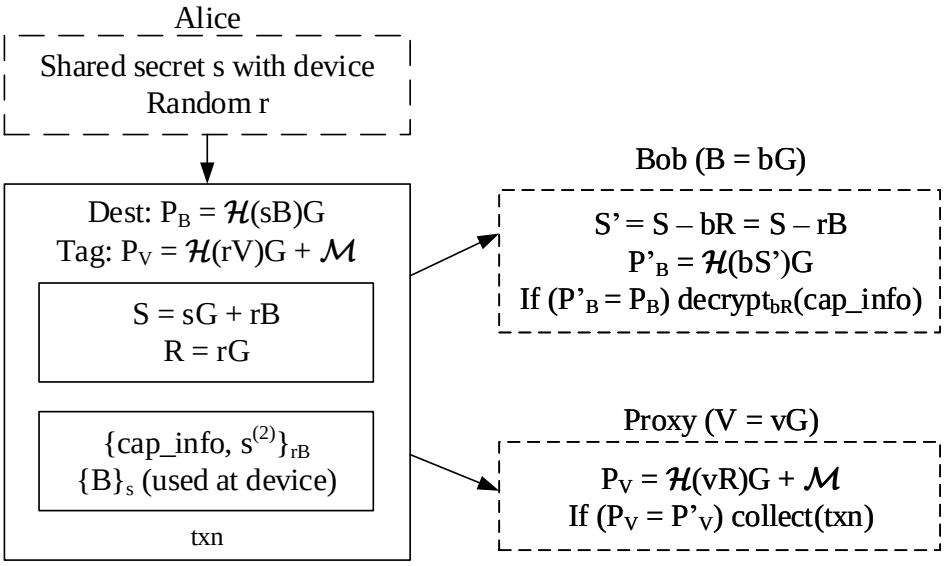


Figure 4.3: Alice computes addresses and attaches necessary information to transaction txn. $\{X\}y$ denotes X is encrypted by key y, cap_info is the capability information, including CAP ID, expiry date, depth and blinding factors used in commitments.

unlinkable address that makes sure only the recipient can spend the money, we want to allow the delegators to link all of the delegation originated from them. Therefore we derive the recipient's sub address in a deterministic way as follows.

We start with the root delegation from the owner Alice whose public key is $A = aG$ to user Bob whose public key is $B = bG$:

- Alice chooses a secret s, which is also shared with her device, and computes the sub address P_B for Bob using Bob's primary address B: $P_B = \mathcal{H}(sB)G$. She also chooses another random secret r and computes $S = sG + rB$, $R = rG$.

- Alice encrypts all of the hidden values and blinding factors used in commitments using one-time shared secret with Bob $rB = brG = bR$ and attaches the encrypted information to the transaction, which is destined to P_B.

- Bob scans every incoming transaction and computes $S' = S - bR = S - rB$, then $P'_B = \mathcal{H}(bS')G$. The transaction is destined to Bob if P'_B equals to P_B.

- Since P_B is a one-time address, Alice encrypts Bob's primary address B using the secret s so that her device can use B for further verification.

Figure 4.3 demonstrates Alice's computation of addresses and how Bob recognizes his transactions.

If Alice wants to allow Bob to delegate the capability further, she will also encrypt a value $s^{(2)} = H(s)$ and attach it to the transaction. Now Bob can derive a sub address for Carol as $P_C = \mathcal{H}(s^{(2)}C)G$.

In general, a user with public primary address X who receives a capability at depth d will have the following sub address:

$$P_X = \mathcal{H}(s^{(d)}X)G \tag{4.4}$$

By computing recipient's addresses using hash chain as above, Alice is not only able to revoke Bob's capability but also Carol and any further user. It is straightforward to revoke Bob's by announcing his sub address P_B. In order to find Bob's transaction destined to Carol, recall that Bob has to provide his corresponding key image $I_B = p_B\mathcal{H}_p(P_B)$ in his ring signature. As the private key p_B is also known to Alice, she can easily compute I_B and look for the transaction that contains this key image and find Carol's sub address P_C. Knowing Carol is at depth 2, Alice can decrypt her primary address C and compute the private sub key p_C, thus the revocation can be continued. Figure 4.4 demonstrates a chain of delegation transactions starting from the owner Alice. It is crucial that Alice's revocation only works if Bob creates Carol's sub address properly using the correct depth and primary address. Although the confirmation process prevents Bob from lying about Carol's primary address, he can use a false depth. For example, instead of using $s^{(2)}$, Bob can raise it by 1 to $s^{(3)}$ and use it for Carol. Carol then tells the device that she is at depth 3 and since the address is still generated from the same seed, the authentication is successful though Alice can no longer trace back Carol unless she checks for every $s^{(i)}$. Therefore to force Bob to use the appropriate depth, the depth value should be attached to the transaction as a

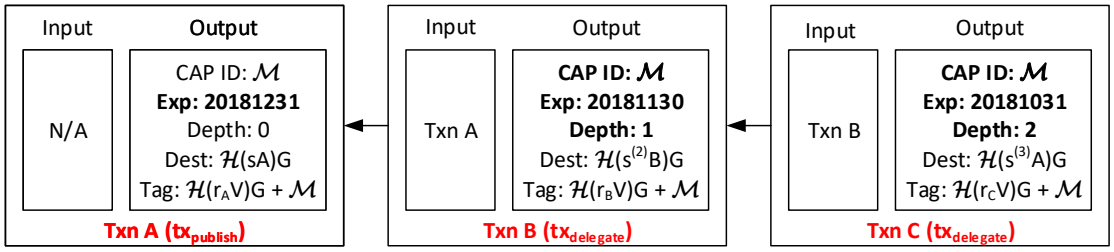


Figure 4.4: An example of a delegation chain from Alice to Bob, then to Carol. **Bolds** are hidden information.

commitment so that Bob cannot change it to other values. We present how to create such commitment in section 4.3.3.

Domain's sub address

Using the same random keypair (R, r) and the capability ID point $\mathcal{M}$, Alice tags the domain to her transaction by computing the domain's sub address

$$P_V = \mathcal{H}(rV)G + \mathcal{M}^2 \tag{4.5}$$

and attaches it with the transaction. The purpose of the domain's address is two-fold. First, similar to recipient Bob, the local proxy can recompute the sub address $P'_V = \mathcal{H}(vR)G + \mathcal{M} = P_V$ so that it may store only transactions destined to its domain. Second, the tag also serves for revocation/confirmation purposes. As stated in section 4.2.2, a $\mathsf{tx}_{\mathsf{delegate}}$ cannot be used for further delegation without a $\mathsf{tx}_{\mathsf{confirm}}$. Since the private key of $\mathcal{M}$ is known to the device, it also knows the private key p_V, hence can sign the transaction as an anonymous confirmation. The confirmation can be relayed by the local proxy, but the proxy is unable to sign the transaction by itself as it does not know the private key, therefore compromise can be mitigated.

[2]In practice, $\mathcal{M}$ should be hidden so the proxy may not know which capability point to use. Therefore the tag instead uses a public obfuscated version of $\mathcal{M}$, which is presented in section C. In either case, only the device or the owner can compute the private key.

4.3.3 Delegation with commitments

Capability ID obfuscation and depth commitment Suppose Alice is at depth $d-1$ of a delegation chain and is holding a capability CAP with ID point $\mathcal{M}$ and expiry time t. To make a delegation to Bob, first she needs to obfuscate her capability ID. We leverage Confidential Assets [73] by publishing an obfuscated ID M_{cap} as below

$$M_{cap} = \mathcal{M} + mG \tag{4.6}$$

where m is a random scalar. After $\mathcal{M}$ is obfuscated, an input commitment to the depth d can be computed as follows:

$$M_{in} = (d-1)M_{cap} + m_1 G \tag{4.7}$$

As the next depth must be d, the output commitment is

$$M_{out} = dM_{cap} + m_2 G \tag{4.8}$$

Since both $\mathcal{M}$ and d are hidden, it is necessary to build a ring signature that can prove that 1) M_{cap} is committed to $\mathcal{M}$ and 2) M_{out} and M_{in} are committed to d and $d-1$, respectively. We create a single ring signature that can prove both statements at the same time as follows. From equation (3) - (5) we have

$$M_{out} - M_{in} - M_{cap} = (m_2 - m_1)G \tag{4.9}$$

$$M_{out} - M_{in} - \mathcal{M} = (m_2 - m_1 - m)G \tag{4.10}$$

Hence

$$2(M_{out} - M_{in}) - M_{cap} - \mathcal{M} = [2(m_2 - m_1) - m]G = \mu G \tag{4.11}$$

As the left-hand side is a public key that only Alice can sign for with secret key $\mu = 2(m_2 - m_1) - m$, to prove that the commitments are committed to $\mathcal{M}$ and d, Alice can pick

a number of inputs $M_{in,1}, ..., M_{in,n}$ and capability ID candidates $\mathcal{M}_1, ..., \mathcal{M}_n$ and create a ring signature on :

$$\begin{aligned}\{&2(M_{out} - M_{in,1}) - \mathcal{M}_1 - M_{cap},\\ &2(M_{out} - M_{in,2}) - \mathcal{M}_2 - M_{cap},\\ &...,\\ &2(M_{out} - M_{in,n}) - \mathcal{M}_n - M_{cap},\\ &2(M_{out} - M_{in}) - \mathcal{M} - M_{cap}\}\end{aligned}$$

Proof We show that (4.11) is sufficient to guarantee that a malicious user cannot create illegal M_{cap} and depth. Suppose that Mike chooses $x, y \neq 1$ and computes the commitments as follows

$$\begin{aligned}\mathcal{M}' &= x\mathcal{M} + mG\\ M_{in} &= d\mathcal{M}' + m_1 G\\ M_{out} &= (d+y)\mathcal{M}' + m_2 G\end{aligned}$$

Hence

$$2(M_{out} - M_{in}) - \mathcal{M}' - \mathcal{M} = (2xy - x - 1)\mathcal{M} + \mu G \tag{4.12}$$

In order for Mike to sign with μ, $\mathcal{M}$ must be canceled out, implying $2xy - x - 1 = 0$. It is clear that $x = y = 1$ is the only solution, which means that Mike cannot cheat and create illegal commitments.

Capability commitment After having the obfuscated M_{cap} for CAP, an input commitment to the capability with expiry time t is

$$C_{in} = xG + tM_{cap}$$

where x is another random scalar. Now to delegate CAP with a new expiry time t_1, two output commitments can be created as follows:

$$C_{out} = x_1 G + t_1 M_{cap} \tag{4.13}$$

$$\overline{C_{out}} = x_2 G + (t - t_1) M_{cap} \tag{4.14}$$

By the ring signature on $C_{in} - (C_{out} + \overline{C_{out}})$ and range proofs on t_1 and $t - t_1$, the network can verify that the new expiry time t_1 must not be later than the original one t.

The reason M_{cap} is used instead of $\mathcal{M}$ is it serves both as a blinded version of $\mathcal{M}$ and a public point similar to the point H in equation (4.1) so that the network can use to verify the ring signatures and range proofs. It should be noted that here C_{out} is the only valid delegated capability, the complement $\overline{C_{out}}$ is just a dump output that serves for the verification purpose.

Unlike money that cannot be double spent, capabilities can be transferred multiple times to different users. However as ring signature is linkable by the key image, it is possible to detect a repeated transfer of a capability. To avoid that, besides a normal capability output, a delegation can also include an echo output that sends back the capability to the sender but under a new address and invalidates the old one. The new echo address is generated in exactly the same way with the normal recipient's address, for example, Alice's echo address will be

$$P_A = \mathcal{H}(s^{(d)} A) G$$

4.3.4 Access request at device

In the first use of the capability received from Alice, Bob presents the transaction to the device and signs it with his primary keypair (B, b). The device then verifies the signature, decrypts the primary address B and checks if the sub address P_B is generated from the correct seed s. If everything is correct, it sends a $\mathsf{tx}_{\mathsf{confirm}}$ to CapChain by signing with the domain's sub address P_V. After the transaction is confirmed, Bob can use his sub address

instead of his primary one for the signature and the device no longer needs to check the generation of the sub address. To prove the ownership of the capability received from Alice, Bob needs to

1. Prove that his capability ID and expiration time are valid as these values are hidden from the transaction.

2. Prove that he is the recipient of the transaction.

Algorithm 2 Request verification at device

```
1: procedure VERIFY(txid)
2:     Verify commitment of capability
3:     if !is_confirmed(txid) then
4:         Verify commitment of depth d
5:         Verify generation of user's sub address P_X = H(s^(d) X)G
6:         Post tx_confirm
7:     end if
8:     Decrypt user's primary address using s^(d)
9:     Verify user's signature by primary address
10: end procedure
```

To prove the capability information, from equation (4.6) and (4.13) we have

$$C = C_{out} - t_1\mathcal{M} = (x_1 + t_1 m)G \tag{4.15}$$

which is a public key with $c = x_1 + t_1 m$ as private key. Therefore Bob can provide a signature with (C, c) as the public-private key pair. The device then can perform the same calculation as 4.15 to obtain the public key C and verify the signature.

To prove the identity, Bob needs to show that his sub address is generated by the correct depth d. Similar to the capability information, he also knows the private key of the public key $M_{out} - d\mathcal{M}$, hence can provide another signature. After verifying the depth, the device can decrypt Bob's primary address and check if Bob's sub address $P_B = \mathcal{H}(s^{(d)}B)G$. It is noticeable that any insider having the same capability but at a smaller depth than Bob can deduce his sub address as well as the private key. Therefore although this does not affect

the delegation process as nobody except Bob can produce the ring signatures without the knowledge of the commitment blinding factors, Bob should use his primary address instead of the sub address as the proof of his identity.

Since the purpose of sub address is to facilitate the revocation process and the user's identity is proved using primary address instead, we only have the device check the generation of sub address at the first time a transaction is presented to reduce the computation overhead. The procedure is summarized in Algorithm 2.

4.4 Experiments and discussion

Our testbed includes an Arduino MKR1000 with a 32-bit ARM Cortex-M0+ MCU with 32 KB of SRAM and 256 KB of flash and a Raspberry Pi Zero W with a 1 GHz single-core CPU and 512 MB RAM. Both the Arduino and the Pi simulate smart devices, but the Pi is also a CapChain node that acts as a local proxy for the less powerful Arduino.

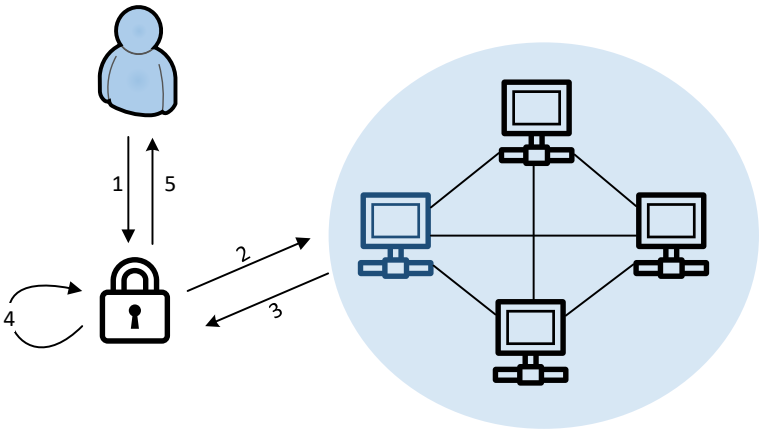


Figure 4.5: Testbed with a low-power Arduino as smart device and a trusted daemon running on blockchain

4.4.1 Blockchain performance

We build CapChain based on Monero's source codes. A test network is deployed on our MSU HPCC, which consists of 20 mining nodes and 20 wallet nodes that send transactions every 5 seconds. The Pi is a regular node that communicates with HPCC network through SSH tunneling. For convenience, we leverage the mining process to publish 4000 capabilities before starting the transfer. With the current Monero's implementation of proof of work, a new block is generated every 3 minutes on average, which means users should wait for at least 3 minutes before receiving a capability. For reliability, users may wait for more new blocks to make sure their transactions are on the main chain. However, such latency only occurs at the delegation phase as once the capability is received, it can be used at the IoT device without any more transaction involved, except $\mathsf{tx}_{\mathsf{confirm}}$ sent by the device to confirm that it has seen the capability.

Compared to the current size of a basic Monero transaction which is 13 KB, our transaction takes up to 30 KB due to additional signatures and range proofs. However, since an access right is often valid for a period of time, a delegation would happen less frequently than a money transaction, hence the growth of the blockchain should also be much slower. A Raspberry Pi with 1 GB spare storage (excluding the OS) can store more than 30000 delegations, which is scalable to larger organizations beyond that of home environments.

4.4.2 Performance at IoT devices

On the Arduino we used the ArduinoLibs [3], which supports Curve25519 [20] that is used in Monero's signature scheme. The performance is shown in Table 5.3.

We found that the point multiplication operation contributes the most to the computation time. Since both the generation of sub address and signature verification requires 2 multiplications, they have similar speed. The verification of capability commitment takes the longest time due to 1 additional multiplication as in equation (4.15). Although the ver-

[3] https://rweather.github.io/arduinolibs/crypto.html

Table 4.1: Processing time of request

Action	#multiplications	Processing time on Arduino MKR	Estimated time on Arduino Due
(1) Verifying sub address	2	1436 ms	315 ms
(2) Verifying depth commitment	3	1656 ms	363 ms
(3) Verifying user's signature	2	1394 ms	306 ms
(4) Verifying capability commitment	3	2125 ms	466 ms
(5) Other computations (parsing messages, decoding, etc.) and communication		111 ms	24 ms
Total round trip time at first use with confirmation		**6777 ms**	**1616 ms**
Total round trip time without confirmation (3)+(4)+(5)		**3685 ms**	**796 ms**

ification of depth should have a similar processing time, we found that with small values like depth which is often less than 10, doing a loop of point additions is much faster than multiplication, hence the actual processing time is reduced.

Although the total processing time per request seems not realistic due to the slow signature verification process, we believe the performance can be improved with a slightly more powerful hardware or a different choice of library and curve. According to ArduinoLibs benchmark, an Ed25519 signature verification [48] takes 306 ms on Arduino Due with ARM Cortex-M3 MCU, which is about 4.5 times faster than the Arduino MKR. Like CryptoNote, the Ed25519 signature scheme also consists of 2 point multiplications in the verification, thus it should takes a similar amount of time for the CryptoNote scheme to perform on the same hardware. For curve comparison, we performed an offline test on the same device with secp256k1 curve provided by microECC library [5]. The test resulted in 496 ms per verification. On a different note, previous projects that implement ECC on resource-constrained devices also show a more reasonable performance. For example, Skarmeta et al. [80] has an optimized version of ECDSA which takes 215 ms to validate a signature on a JN5139 mote with a 32-bit RISC low power processor. An analysis by Mssinger et al. [64] also reports a verification time of 476 ms with a fast implementation of secp256k1 on a TI's CC2538 mote. The libraries also contribute an important part to the speed. In the same analysis, a library that supports Curve25519 is also tested but it results in 7 seconds. Unfortunately, we

could not find any other Curve25519 library for Arduino, therefore it is uncertain to justify whether ArduinoLibs can be optimized. However given that all of the mentioned platforms are targeted for constrained devices, we believe that it is feasible to implement our solution in IoT applications with reasonable processing time. Based on the reported benchmark from ArduinoLibs, we calculate a rough estimation of processing time on Arduino Due, assuming it is proportional to the processing time on the Arduino MKR. It is shown in table 5.3 that the total processing time without confirmation stage could be less than a second.

The trusted local proxy does not need to be a powerful node that stores the entire blockchain but only transactions that belong to its domain. Compared to the current size of a basic Monero transaction which is 13 KB, our transaction takes up to 30 KB due to additional signatures and range proofs. However, since an access right is often valid for a period of time, a delegation would happen less frequently than a money transaction, hence the growth of the blockchain should also be much slower. A Raspberry Pi with 1 GB spare storage (excluding the OS) can store more than 30000 delegations, which is scalable to larger organizations beyond that of home environments.

4.5 Conclusion

We present CapChain - an access control framework for sharing and delegation based on blockchain technology. CapChain is not only reliable thanks to the blockchain architecture but also preserves user privacy by hiding sensitive information about the access delegation from the public. Our experiments show the applicability and scalability of CapChain in IoT environments.

Continuing focus on privacy, in the next chapter we extend the use of shared tokens to provide automated tasks based on certain conditions. It is presented by PPCA, a privacy preserving service for IoT conditional actions.

Chapter 5

PPCA: Privacy-Preserving Conditional Actions for IoT Environments Using Smart Contracts

5.1 Introduction

With the development of smart devices, automation plays an important role in both consumer and industrial IoT environments. In most scenarios, automated IoT devices are controlled according to conditions within the environment. For example, in home automation, a common scenario is to have a light turn on if occupancy is detected in a room by a thermostat. In terms of access control, the scenario involves 2 types of permissions: 1) to access data of the thermostat; 2) to control the light. A trivial way to implement this automated task is to give a third party both permissions so that it can listen to events from the thermostat and command the light accordingly. However, giving control permission without restriction could allow the third party to be over-privileged, as it now can actually turn on/off the light any time regardless of the occupancy status. IFTTT (If-This-Then-That [6]) is a popular platform that serves as such a third party. It acts a bridge between different services and allows users to specify a set of conditional statements (called applets) to trigger their services. As an trusted party that stores a large number of API keys, IFTTT could be an attractive target to attackers who want to exploit the over-privileged permissions.

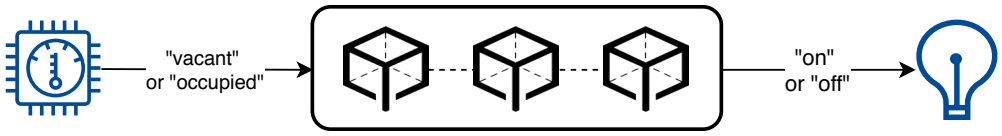


Figure 5.1: A thermostat frequently reports occupancy status to the blockchain to control the light bulb.

To prevent such attacks, it is necessary to restrict the third party so that it is only allowed to act given certain conditions. We call such type of actions *conditional actions*. As a distributed ledger, we find blockchain an ideal decentralized platform for conditional actions, where the blockchain can serve as a public witness to verify the given conditions, which can be defined in forms of conditional statements with the help of smart contracts. For example, the above home scenario can be implemented on a blockchain as described in Figure 5.1, which shows a thermostat that regularly uploads the occupancy status to a smart contract on the blockchain. If the status says "occupied", the smart contract sends a message to the light, telling it to switch on.

Although the above implementation seems straightforward, it has several problems with privacy:

- If a condition is translated directly to a smart contract, it can reveal users' sensitive information. For example, the occupancy status could tell if the home is vacant or not. Even if the data may be "sealed", events triggered by the conditions can still be observed, which can help adversaries to study users' activities.

- In most cases, the conditions rely on certain data providers. The data can be in the form of texts such as occupancy messages, but also numerical values such as weather data or locations. Due to the heterogeneity of the IoT, it is difficult to define a smart contract that involve various type of data for the conditions without losing privacy.

- Posting data to a blockchain regularly is not cost-effective, especially for payment-required platforms such as Ethereum. In addition, having events triggered by the

blockchain can cause delay as new blocks need time to be confirmed. This is not suitable for real-time services.

In this chapter, we design PPCA - a privacy-preserving service for conditional actions using smart contracts. Our model comes from the insight that most conditions can be described in forms of either texts or numerical values. Therefore, using homomorphic encryption, we can transform the data into commitments that are verifiable by the blockchain network but do not expose the real data.

5.1.1 Chapter organization

This chapter is organized as follows. Section 5.2 presents the design of PPCA. Next, the transformation of conditions is presented in section 5.3. The implementation of PPCA using smart contracts and real devices is provided in section 5.4, followed by the evaluation in section 5.5. The security and privacy analysis of PPCA is discussed in section 5.6. Finally, section 5.7 concludes the chapter.

5.2 System design

5.2.1 Definitions

Condition components

A conditional statement consists of the following components:

- Data provider: provides data for the conditions. The thermostat in Figure 5.1 is an example of a data provider.
- Data reported by the data provider.
- Trigger: is the data that satisfies certain conditions.
- Action: specifies what happens when the conditions are satisfied.

Figure 5.2 describes two conditional statements and their corresponding components. For example, for a conditional statement "*if my thermostat reports occupancy status as occupied, turn on the light*", the data provider is the thermostat, the reported data is the occupancy status, "occupied" is the trigger and the action is to turn on the light.

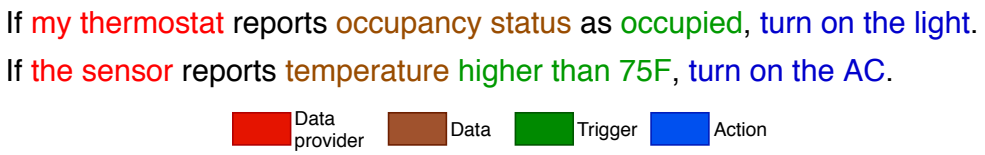


Figure 5.2: Condition components

In order for an action to be executed, besides the data provider, we also specify two other parties:

- Target service: is the endpoint of the action. The target service should be able to verify the conditional permissions and grant access accordingly. The light bulb in Figure 5.1 is an example of a target service.
- Watcher: regularly receives data from the data provider and watches for when the data satisfies the condition, then makes a request to trigger the target service accordingly. The watcher can be a home PC or a server in a larger-scale system.

5.2.2 Design goals

We address the following design goals:

- Strict privacy on blockchains: no private data should be exposed on the blockchain.
- Weak privacy with data provider and target services: data providers only provide data and do not need to be aware of the conditions that trigger the target services. Likewise, the target services also may process the access request without knowing about the triggers.
- Strict privilege: watchers can request the target services only if the conditions are satisfied.

- Permissions should be fast to confirm if the target is a resource-constraint device.

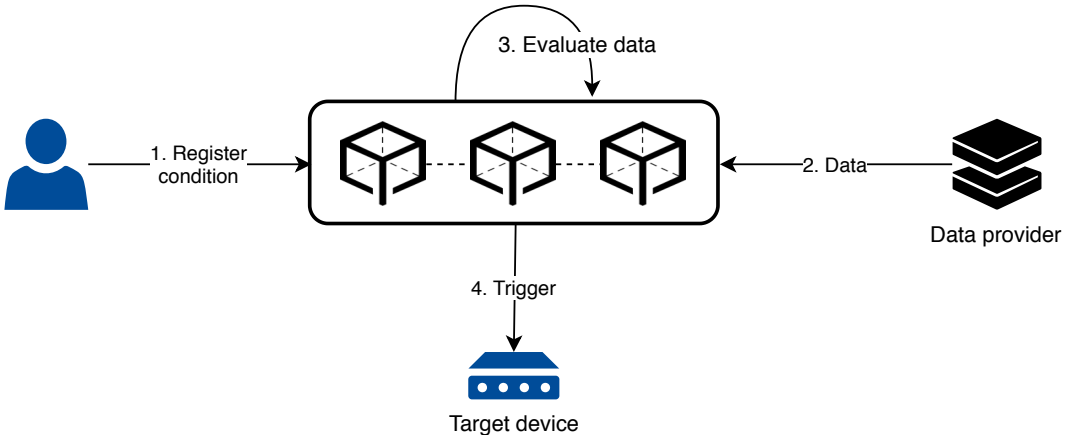


Figure 5.3: Non-privacy model with blockchain as a watcher.

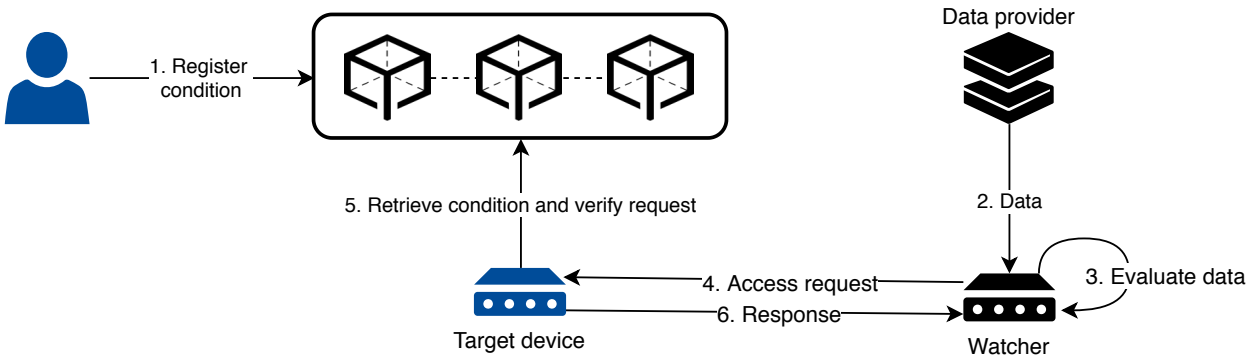


Figure 5.4: Proposed model with watcher separated from the blockchain.

5.2.3 A non-privacy model

We start with a straight-forward but non-privacy model as described in Figure 5.3. In this model, users first register their conditional statements on to the blockchain. The blockchain also serves as a watcher that regularly evaluates data from data providers to trigger the target service.

This model has several benefits. First, with the blockchain serving as a central endpoint, the data provider and target resource can be free from most heavy operations. Second, the target resource does not need to be aware of any condition. If the smart contract is well-designed, the blockchain will be a secure watcher that cannot have false execution and

over-privileged access. The blockchain is also resilient to denial-of-service attack. If the condition is satisfied, the action must be performed without any interruption. However, since the data is sent to the blockchain, as previously discussed in Section 5.1, there may be significant delay when using the blockchain to trigger events. Additionally, it is a challenge to design a smart contract that not only can evaluate a variety of conditions but also does not expose user privacy.

5.2.4 The proposed model

Since having data posted directly to the blockchain is not ideal for privacy, we propose to have the watcher as a separate entity. In this model, only condition definitions are registered on the blockchain. The communication between the data provider and watcher can be carried out off-chain. When the received data satisfies the conditions, the watcher presents a proof to the target resource for a request. The target resource then retrieves the condition definitions from the blockchain and verifies the request.

With most of the important data being transferred off-chain, the only sensitive data left on the blockchain is the condition definitions. A trivial approach to protect data privacy is to have the data encrypted. Following this approach, the user can first encrypt the entire condition before posting it to the blockchain, then the target service will need to decrypt it. There are several drawbacks with this approach:

- The condition cannot be modified. If the user wants to adjust a parameter, he/she would need to revoke the old condition and create a new one.

- It is not scalable. Since the condition is encrypted, only the target resource can decrypt it. If the user wants to use the condition to another target, he/she would need to either encrypt it with the new target's key or share the key to the new target. In the latter case, a revocation scheme is required to manage all of the targets.

- A translator is needed to translate various conditions to a format that the target can

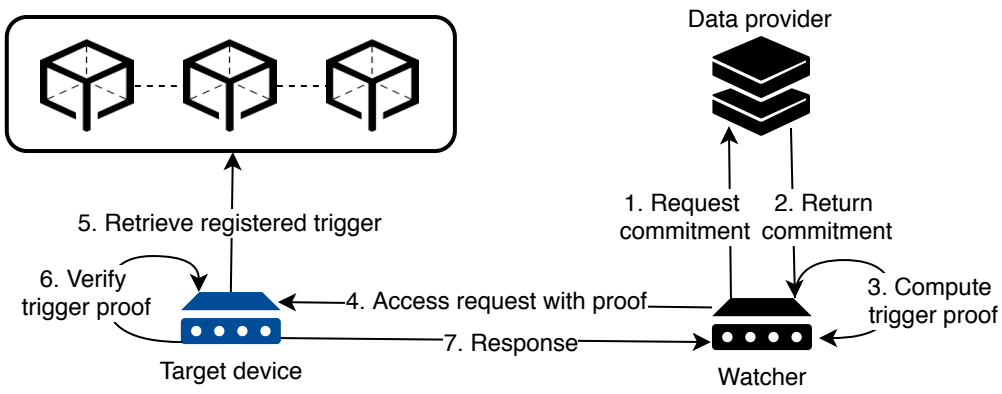


Figure 5.5: Workflow to verify a conditional action.

use to verify requests.

Although there are various types of data, we notice that most conditions can be described by $data == x$, $data \leq x$ or $data \geq x$, where $data$ can be represented in either texts or numerical values. Therefore, the triggers can be categorized into two types:

- Exact values: usually in text forms. For example, the light status is "on" or "off", the occupancy status is "vacant" or "occupied", etc.
- Range values: usually in numerical forms. For example, the temperature is higher than 75F, the current location is within 10 feet from home, etc.

By generalizing the triggers to these 2 types, using Elliptic Curve Cryptography (ECC) and Pedersen commitments, we can published the triggers (specifically, $data$ and x) in the form of commitments to the blockchain instead of plain data. An advantage of using the Pedersen commitment scheme is that it has a homomorphic property that allows the construction of a proof that the committed value is in a certain range. Therefore, the target service can still verify the trigger using the proof without knowing what the real trigger value is.

The workflow is as follows:

1. The user uploads to the blockchain a commitment to the trigger value and the data provider that will provide the value.

2. The user grants the watcher a permission to a target device, specifying the trigger required to use the permission. The permission is also stored on the blockchain.

3. The watcher keeps receiving data from the data provider. If the data activates the trigger (i.e. the trigger condition satisfies), it asked the data provider for the commitment to the trigger data.

4. The watcher generates a proof that the trigger condition satisfies and makes a request to the target device.

5. Upon receiving the watcher's request, the target device queries the blockchain for the permission and trigger commitment, then verifies the proof.

5.2.5 Threat model and assumptions

We assume that the data published to the blockchain cannot be tampered. Therefore, nobody can modify triggers or permissions once they are registered on the blockchain, except the user who defines the conditional action and may be able to revoke it through a revoke transaction.

The user is assumed to have both access to the data provider and target service. Hence, the watcher can be granted permission to access the data provider by the user. The watcher can be considered a representative of the user, as it can access both the data provider and target service, except that the action on the target service is conditional. The watcher may be compromised and attempt to make an action even when the condition is not satisfied. The watcher may also refuse to perform when the condition is satisfied. However, in case of compromise, we consider over-privilege more important than denial-of-service.

On the other hand, since conditional permissions depends on data providers, we assume that the data provider is honest, i.e. the provided data is always true. It is possible that the data provider may be compromised to provide false data, but this type of attack is not in the scope of this dissertation.

We consider the watcher and data provider as two separate physical entities. One can argue that if the data provider is honest, the watcher can be embedded into the data provider, which then can be allowed to command the target service directly. For example, the thermostat can be allowed to trigger the light. However, this is not possible if the action involves multiple conditions with multiple data providers. Additionally, in many cases, the data provider and target service are not managed by the same user (for example, using an external weather service to turn on lights at sunset). Therefore, allowing it to make a request would result in over-privilege access.

5.3 Condition transformation

5.3.1 Preliminaries

Like Chapter 4, our computations are also based on Elliptic Curve Cryptography (ECC). An ECC public key P with private key p is a point on an elliptic curve E such that $P = p \times G$, where G is a base point on E. We assume that both user, data provider and watcher have a public key that is registered on the blockchain.

5.3.2 Transformation of range values

Range proofs using Pedersen commitments

For range values, our goal is to build a proof that $a < b$ without revealing both a and b. Particularly, the upper bound b is specified by the user and represents the condition for a request while value a is provided by the data source. The watcher can make a request to a target device if it can present a proof that $a < b$.

To implement this kind of range conditions, we adopt the range proofs using Pedersen commitments in Confidential Transactions [68]. Instead of publishing the raw values of b, the user computes a commitment to b by

$$C(b) = bH + x_b G \tag{5.1}$$

where x_b is a secret key that can be shared with the watcher. Similarly, the data source also creates a commitment to a by:

$$C(a) = bH + x_a G \tag{5.2}$$

To prove that $a < b$, a prover can compute a commitment to $b - a$ by

$$C(b-a) = (b-a)H + yG \tag{5.3}$$

Since $C = C(b-a) + C(a) - C(b) = (y + x_a - x_b)G$ which is a public key, a signature by C can be used to prove the inequality if the signer knows y, x_a, x_b. Although no party knows all of the 3 secrets (the watcher only knows y and x_b while the data provider only knows x_a), it is possible for them to provide a co-signature. We apply MuSig [59], a multi-signature scheme that is designed for Bitcoin, to compute the co-signature.

It is noteworthy that the commitment $C(a)$ can be computed by any party not just the data source. Therefore, besides the inequality proof, the data source also needs to provide another signature to prove that it is the source of $C(a)$. Having too many signatures causes more computation overhead at not only the data provider but also the verifier. Hence, we incorporate the identity proof in the range proof so that the proving process will only result in one final signature. We provide more details in the next sections.

MuSig recaps

Let $\mathbb{X} = \{X_1, X_2, \ldots, X_n\}$ be the public keys of the co-signers. Every signer computes the same X, L as follows:

$$\begin{aligned} L &= H(X_1, X_2, \ldots, X_n) \\ X &= \sum H(L, X_i) X_i \end{aligned} \tag{5.4}$$

Each signer chooses a random value r_i, and shares $R_i = r_i G$ with the other signers, then they each compute their part of the co-signature for message m

$$s_i = r_i + H(X, R, m)H(L, X_i)x_i$$

where $R = \sum R_i$. The final signature is (R, s) where $s = \sum s_i$. The verifier verifies the signature by checking if $sG = R + H(X, R, m)X$.

Condition registration for numerical values

In this step the owner registers the data source's public-private key pair (D, d) and two commitments for the lower bound and upper bound (v_L, v_U):

$$\begin{aligned} C_L &= v_L D + x_L G \\ C_U &= v_U D + x_U G \end{aligned} \tag{5.5}$$

where x_L, x_U are blinding factors that are shared to the watcher. The watcher also knows about v_L and v_U.

Data provider and watcher co-operate

When the watcher notices a trigger value v, it asks the data provider to co-sign a proof that $v_L < v < v_U$. First, to construct the proof, the watcher and data provider exchange a one-time random key, $X_W = x_W G$ and $X_D = x_D G$, respectively. With each party now holding 2 keys (one main key and one sub-key), the shared factors L, X from (5.4) can be computed by

$$\begin{aligned} L &= H(W, X_W, D, X_D) \\ X &= H(L, W)W + H(L, X_W)X_W + H(L, D)D + H(L, X_D)X_D \end{aligned} \tag{5.6}$$

Hence, the data provider can compute a commitment to v by

$$C_D = vD + H(L, D)D + H(L, X_D)X_D \tag{5.7}$$

and its MuSig part $\{R_D, s_D\}$ on a message m will be

$$\begin{aligned} R_D &= r_D G \\ s_D &= r_D + H(X, R, m)H(L, X_D)x_D + H(X, R, m)H(L, D)d \end{aligned} \tag{5.8}$$

On the watcher's side, to prove $v_L < v < v_U$, it prepares two commitments to $v - v_l$ and $v_U - v$ as follows:

$$\overline{C_L} = (v - v_L)D - H(L, X_W)X_W - H(L, W)W - x_L G \tag{5.9}$$

Similarly, the commitment for the upper bound is:

$$\overline{C_U} = (v_U - v)D + H(L, X_W)X_W + H(L, W)W - x_H G \tag{5.10}$$

The watcher can also generate the second part of the MuSig:

$$\begin{aligned} R_W &= r_W G \\ s_W &= r_W + H(X, R, m)H(L, X_W)x_W + H(X, R, m)H(L, W)w \end{aligned} \tag{5.11}$$

After exchanging all needed information, the watcher gathers 2 MuSig parts and computes the final signature $\{R, s\}$ where $R = R_W + R_D, s = s_W + s_D$. The final proof to be presented to the target device is $\pi = \{C_D, \overline{C_L}, \overline{C_U}, X_D, X_W, [R, s]\}$

Data provider (D, d)		**Watcher** (W, w)		**Target service**
Save X_W, R_W	$\xleftarrow{X_W, R_W}$	Generate $(X_W, x_W), (R_W, r_W)$		
Generate $(X_D, x_D), (R_D, r_D)$	$\xrightarrow{X_D, R_D}$	Save X_D, R_D		
Compute L, X by Eq. (5.6)		Compute L, X by Eq. (5.6)		
Compute C_D by Eq. (5.7)		Compute $\overline{C_L}, \overline{C_U}$ by Eq. (5.9),(5.10)		
Save $id_{CA}, \overline{C_L}, \overline{C_U}$	$\xleftarrow{id_{CA}, \overline{C_L}, \overline{C_U}}$	Generate $(X_W, x_W), (R_W, r_W)$		
$m = \{id_{CA}, C_D, \overline{C_L}, \overline{C_U}, X_W, X_D\}$				
Compute s_D				
	$\xrightarrow{C_D, R_D, s_D}$	Save C_D, R_D, s_D		
		$m = \{id_{CA}, C_D, \overline{C_L}, \overline{C_U}, X_W, X_D\}$		
		Compute s_W		
		$R = R_W + R_D$		
		$s = s_w + s_D$		
		$m' = \{m, v - v_l, v - v_u\}$	$\xrightarrow{R, s, m'}$	Retrieve C_L, C_H, W, D from the blockchain using id_{CA}
				Compute L, X by Eq. (5.6)
				Verify $v - v_l, v_u - v$
				Verify Eq. (5.13)
				Verify Eq. (5.14)

Figure 5.6: Steps to construct a proof for a conditional action (CA) with range value and verification at the target service. The CA is referred by the ID id_{CA} on the blockchain.

Target device verifies the request

From equation (5.5), (5.7), (5.9), (5.10), it can be seen that

$$\begin{aligned} &C_D - (C_L + \overline{C_L}) = C_U - (\overline{C_U} + C_D) \\ =&H(L, D)D + H(L, X_D)X_D + H(L, W)W + H(L, X_W)X_W \\ =&X \end{aligned} \tag{5.12}$$

Therefore, given the aggregated MuSig on X, the target device can verify both the range proof and the identities of the watcher and datasource. Particularly, given the proof π, the target device performs the following operations to verify the request:

- Verify the range proof by checking if

$$\begin{aligned} &C_D - (C_L + \overline{C_L}) = C_U - (\overline{C_U} + C_D) \\ =&H(L, D)D + H(L, X_D)X_D + H(L, W)W + H(L, X_W)X_W \end{aligned} \tag{5.13}$$

- Verify the MuSig:

$$sG = R + H(X, R, m)X \tag{5.14}$$

- Checking (5.13)-(5.14) alone is not enough to guarantee that the condition is satisfied. Since the elliptic curve is a cyclic group, $v - v_l$ and $v_u - v$ may overflow, which can still make (5.13) correct even though v is invalid. Since the difference $v - v_l$ and $v_u - v$ do not expose v, v_l or v_h, they can be revealed to the device for overflow checking. To retrieve more privacy, a range proof using ring signature can be computed instead of presenting the plain values. However, range proofs are more complicated and would take more gas/time to build.

Figure 5.6 describes the details of the protocol to construct the range proof and verify the conditional action at the target service.

Security guarantees

1. *The watcher cannot produce fake data and generate a proof by itself since it requires the data provider as the co-signer.*

2. *The watcher cannot generate a valid proof with data that does not satisfy the conditions.*

   Let v' be a value the does not satisfy the lower bound v_L, i.e. $v' < v_L$. The commitment to v' will be

   $$C'_D = v'D + H(L, D)D + H(L, X_D)X_D \tag{5.15}$$

   To satisfy (), the commitment $\overline{C'_L}$ generated by the watcher must be committed to $v' - v_L$. However, since $v' < v_L$, $v' - v_L$ is negative and will not pass the overflow checking. Similarly, if v' exceeds the upper bound v_U, the commitment will also be invalid.

5.3.3 Transformation of exact values

The transformation of exact values is simpler than range values. The user register an exact value v to the blockchain by mapping v to a point M as follows:

$$P = h(v, s)G \tag{5.16}$$

where s is a shared secret between the user and data provider. Since the data provider knows the private key of P, it can generate a signature with M as the public key. Therefore, a proof for trigger v can be a co-signature between the watcher with public key W and data provider with public key P.

Watcher (W, w)		**Data provider** (P, p)
Generate (R_W, r_W)		
$m = \{id_{CA}\}$	$\xrightarrow{R_W, m}$	Save R_W, m
Compute L, X using Eq. 5.4		Generate (R_P, r_P)
		$R = R_P + R_W$
		Compute L, X using Eq. 5.4
Save R_P, s_P	$\xleftarrow{R_P, s_P}$	Compute s_P
$R = R_W + R_P$		
Compute s_P		
$s = s_W + s_P$		

Figure 5.7: Steps to construct a proof for a conditional action (CA) with exact value.

5.4 Implementation

5.4.1 Curve choice

We choose alt_bn128 as the elliptic curve for our ECC operations. Since alt_bn128 is supported by Ethereum in precompiled contracts, using it can reduce the cost of our contract. Although other curves may be chosen since technically the smart contract is only to store commitments and the ECC operations are performed locally at the front-end, using alt_bn128 enables the future use of blockchain as a watcher.

5.4.2 The front-end

We implement the front-end interface for users in Node.js and in Python for the watcher and data provider. With alt_bn128 supported, all the ECC operations can be also written in Solidity (the smart contract language of Ethereum) and run in Ethereum Virtual Machines (EVM). However, we opt into Python since it runs faster than using EVM. The Python library for alt_bn128 is from [7].

5.4.3 The smart contract

The smart contract is implemented on Ethereum for registration and management of conditional actions.

Identities

Technically, each user needs an Ethereum address as an identity to make transactions. Although the Ethereum address is already a public key[1], it uses the secp256k1 curve instead of alt_bn128. Therefore, we manage a mapping from an Ethereum address to an alt_bn128 public key. Before using the system, each entity (user, data provider or watcher) uses their master address to register the public key on the smart contract.

Public keys A list of registered alt_bn128 public keys as discussed above.

User permissions As discussed earlier, we assume the user is already permitted to use the target resource and how the permission is granted is not in the scope of the chapter. Therefore, we mimic the resource ownership by a table that records permissions held by a user. The permission is only an abstract field that represented by a 32-byte token. Depending on the target endpoint, the actual permission can be an API key, or a capability token, etc.

Commitment Represents a trigger value that can be either to a text message or a numerical values. A commitment is identified by a tuple of:

- `commitment_value`: a Pedersen commitment to the value.
- `provider_address`: the Ethereum address of the data provider.
- `commitment_hash`: the 256-bit hash value of the commitment, used as a key to look up commitments.

[1]To be precise, it is the hash of the public key.

Trigger Represents the condition to activate a permission. It is identified by a tuple of

- `commitment_hash`
- `trigger_info`: a JSON string that contains details of the trigger, including type of trigger (i.e. lower bound, upper bound or exact value), scaling factor (since commitments are only for integers, floating-point values needs to be scaled before generating commitments). In addition, as discussed in section 3.2, the blinding factors are also needs to be shared to watchers. Therefore, `trigger_info` also includes these values in encrypted form that only the watchers can decrypt. To save transaction cost, the blinding factors can be shared via off-chain communication. However we choose on-chain storage for convenience so that watchers only need to maintain one communication endpoint to the blockchain.

Conditional action (CA) A CA is identified by

- `holder_address`: the Ethereum address of the permission's holder. The holder can be a watcher, or a normal user.
- `resource_address`: the Ethereum address of the target service that will verify the permission.
- `action_token`: represents the action that can be performed at the target resource. This is the token from the user permissions table. Only the owner of the token can issue the conditional action.
- `trigger[]`: an array of triggers needed to activate the token. Access is granted only if all of the triggers are satisfied. The trigger array represents the AND logic. To have the OR condition, a separate permission with the same `{holder, resource, token}` but different set of triggers can be created, so that the watcher only needs to present 1 permission that satisfies.

Functions

The functions that actually spend gas in out smart contract are:

- `registerPubkey()`
- `registerTrigger()`
- `registerConditionalAction()`

The cost of these functions is shown in Table 5.1.

Table 5.1: Ethereum gas cost of register functions.

Function	Gas	Estimated USD [2]
registerPubkey()	69410	$0.0331
registerTrigger()	121797	$0.0581
registerConditionalAction()	313098	$0.1494

In addition, there are also static helper functions that can be executed locally and does not cost gas for proof/signature generation and verification.

5.5 Evaluation

As a proof-of-concept, we set up a private Ethereum network with 2 Dell laptops with Intel Core i5 processor and a Raspberry Pi Zero W with a single-core CPU. With these 3 machines, we perform 3 different tests where the data provider and target resource run on different platforms. While the watcher always run on the laptop, the data provider and target resource can both run on the Raspberry Pi. Since the watcher acts as a broker that can serve multiple conditions, it is not practical to have it run on a weak device. On the other hand, in many scenarios, the data provider and target resource are IoT resource-constraint devices.

For each test, we evaluate the three following conditional actions (CA):

[2]The estimation is based on https://ethgasstation.info, as of May 2019.

Table 5.2: Node configuration.

Role	**Test 1**	**Test 2**	**Test 3**
Watcher	Dell	Dell	Dell
Data provider	Dell	Dell	R. Pi
Target resource	Dell	R. Pi	R. Pi

CA1 ***(Conditional action with exact trigger value)*** "If the room is occupied, turn on the light". In this CA, the data provider regularly sends "vacant" or "occupied".

CA2 ***(Conditional action with range trigger value)*** "If the room temperature is higher than 75F, turn on the AC". In this CA, the data provider regularly sends a random number in the range of (60, 80).

CA3 ***(Conditional action with multiple trigger value)*** "If my current location is within 10 ft from home, turn on the heat". Since the location coordinate consists of 2 values, there are totally 4 triggers in this CA (each longitude/latitude has 2 triggers, lower bound and upper bound).

Table 5.3 shows the total time to perform each action, counting from when a trigger is detected until the request is verified. The processing time is broken down to two phases: 1) time to construct a proof between watcher and data provider and 2) time to verify a request at the target resource.

Table 5.3: Processing time.

		Test 1	**Test 2**	**Test 3**
CA1	Construct proof	0.049	0.053	0.60
	Verify request	0.161	1.556	1.693
	Total	*0.210*	*1.609*	*2.293*
CA2	Construct proof	0.191	0.269	2.862
	Verify request	0.322	3.878	3.731
	Total	*0.513*	*4.147*	*6.593*
CA3	Construct proof	0.180	0.378	4.419
	Verify request	0.917	5.165	5.377
	Total	*1.097*	*5.543*	*9.796*

As shown in the table, if the CAs are performed using regular PCs, the processing time is only within a second. CA1 with exact value has the best time, since it does not require any commitment. The bottleneck is at the Raspberry Pi, which takes 0.5 second to perform a multiplication. Hence, the more complicated the CA is, the longer it takes to verify a request. Although a range proof is more costly and may take more than 4 seconds, actions that involve range values usually are not time-critical and users often are unable to detect exactly when a trigger happens. For example, for CA2, since users usually are not aware of a change in temperature quickly, a few seconds of delay should not affect users' experiences.

It is noteworthy that our experiments are not optimized. Although ECC operations on Python is faster than Javascript, C/C++ may be a better choice for constrained platforms. Unfortunately, it is not supported by Ethereum API. In addition, our model of Raspberry Pi only has a single-core armv6 CPU and 512 MB of RAM, which is the lowest configuration among other current models. A benchmark in [15] shows that a Raspberry Pi with a quad-core CPU can process thousands of signatures per second on the Ed25519 curve. On the other hand, our implementation can only process around 5 alt_bn 128 signatures per second even on a regular PCs. Therefore, we believe with an optimized implementation, the processing time should be significantly smaller.

5.6 Security and privacy analysis

Watcher's strict privilege PPCA can guarantee strict privilege on watchers if the data provider is honest. As shown in section 3, the watcher cannot trigger the target service when the conditions are not satisfied. The co-signature guarantees that the watcher cannot make a request without the data provider.

Blockchain privacy In Ethereum, the identities of users, watchers, data providers and target services are only protected by pseudonyms (i.e. Ethereum addresses). This means that if someone happens to know which service an address belongs to, they may be able to guess what type of conditional actions the user are using. However, since only commitments

of values are stored on the blockchain, nobody can know about the real triggers. There have been proposals to use ring signatures or zero-knowledge proofs to bring anonymity to Ethereum [60, 25], but they are focused on payment applications instead of IoT.

Data provider and target service privacy Unless the addresses are known, the data provider would be unable to know what service its data can trigger. If the data is always sent to the watcher in the form of commitments even when it is a non-trigger, the data provider would not know when the trigger happens. However, if data is sent in normal form and only is committed when the watcher requests, the trigger may be revealed to the provider. How the data is transmitted should be decided by the user, depending on how strict of privacy they want.

Likewise, the target service would also not know about the data provider if it does not know the address. In the verification step, the service needs $v - v_l$ and $v_u - v$ to check for overflow. This may reveal how "far" the trigger value v is from the bounds, but the real value is never disclosed. Although a range proof with ring signatures for overflow checking can be computed [58, 68] to obtain more privacy, we chose not to implement it due to the high cost, especially on constrained devices. More recently, Bünz et al. propose Bulletproof [26], a much shorter and more efficient range proof. Although its performance on constrained devices has not been confirmed, Bulletproof could be a potential improvement for privacy.

5.7 Conclusion

In this chapter, we propose PPCA, a decentralized service to create conditional automated tasks in IoT environments. Using smart contracts and homomorphic cryptography, PPCA can guarantee user privacy and strict privilege that the third party who watches for trigger events can perform actions only when the events satisfy the conditions. The prototype performance shows the feasibility of PPCA in IoT environments.

In chapter 4 and 5, we discussed how access control data is created and transferred via blockchains in a private manner. In the next chapter, we address the dependence of IoT

devices on gateways as discussed in section 2.3.1 and propose a solution for them to retrieve access control data from the blockchain in a reliable way.

Chapter 6

A Lightweight Block Validation Method for Resource-Constrained IoT Devices in Blockchain-Based Applications

6.1 Introduction

In chapter 4 and 5, we used blockchains as a secure environment to store access capabilities. In our proofs of concept, we assume the end resource-constrained device can connect to a gateway to obtain access control information on the blockchain. As discussed in section 2.3.1, the use of gateway nodes removes the decentralization benefit of blockchain since if the gateways are compromised, they can feed the IoT devices a false version of the blockchain and give attackers illegal access. Therefore, we propose a validation mechanism that allows IoT devices to validate block headers without relying solely on a single trusted node. Our idea is to have random nodes in the network, which we call *witnesses*, to help the IoT devices confirm incoming block headers. While we still use the capabilities of more powerful nodes as the edge/gateway nodes to relay the main data, the random selection of witnesses would make it difficult for attackers to compromise the edge nodes. It is noteworthy that our mechanism is not to replace, but work on top of blockchain consensus protocols to support

low-power devices.

6.1.1 Chapter organization

The rest of the chapter is organized as follows. Section 6.2 is an overview of our protocol. Section 6.3 discusses the details of each step of the protocol. We provide a probabilistic security analysis in section 6.4 and performance evaluation in section 6.5. Finally, the chapter is concluded by section 6.6.

6.2 Protocol overview

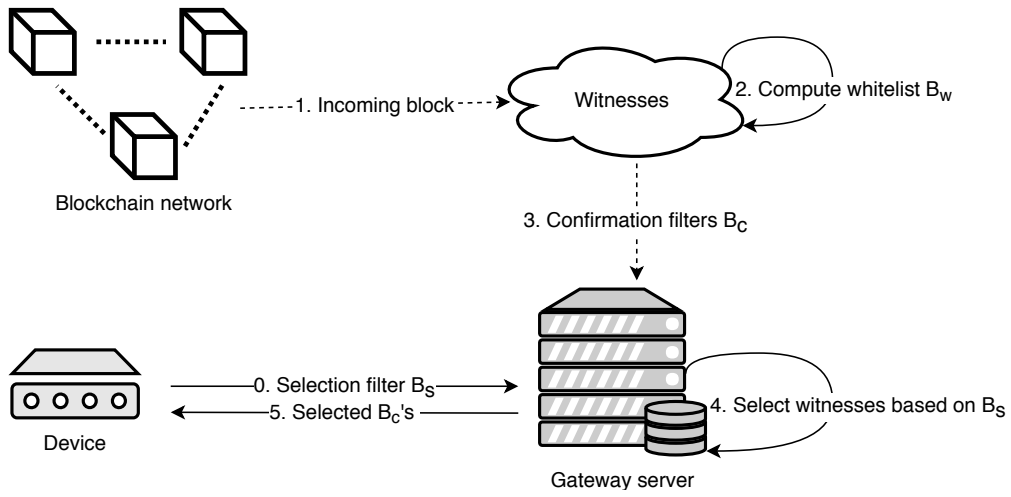


Figure 6.1: Each validation round starts at step 0 when a device sends a random condition represented by selection filter $\mathcal{B}_s$ to its server to select witnesses for the next incoming block.

6.2.1 System model

We consider 3 parties in our protocol:

- *Device* D: a resource-constrained device that may not directly interact with the blockchain network.
- *Gateway server* S: to transfer any necessary messages from the blockchain network to D.

- *Witnesses*: regular peers on the blockchain network that regularly broadcast a lightweight *"confirmation"* message that confirms one or multiple blocks.

The device D will accept a new block header from S if it receives confirmations for the same block from a set of witnesses that is selected randomly to prevent corruption. Our protocol employs a probabilistic approach based on Bloom filters [21], which have been known to be space and time efficient for membership testing. Therefore, it can be lightweight enough for resource-constrained devices. A Bloom filter $\mathcal{B}_X$ is characterized by 3 parameters: the size m_X, the number of bits used per item k_X and the fill rate f_X, which is the ratio between the number of bits that are set to 1 and m_X. Specifically, the following 3 filters are used in our protocol:

- *Confirmation filter* ($\mathcal{B}_c$): this is the main content of the confirmation messages sent by witnesses. $\mathcal{B}_c$ stores individual block confirmations between a witness and each of the devices it is going to serve. Each individual confirmation is inserted to $\mathcal{B}_c$ using k_c bits.

- *Selection filter* ($\mathcal{B}_s$): a random filter generated by each device that represents the condition to select witnesses. A witness is selected if its ID has k_s bits included in $\mathcal{B}_s$.

- *Whitelist filter* ($\mathcal{B}_w$): a pre-selection filter to narrow down the set of devices that a witness can serve. A device is available to a witness and thus can be added to the witness's confirmation filter if its ID has k_w bits in $\mathcal{B}_w$.

It is noteworthy that among the 3 filters, only $\mathcal{B}_c$ actually represents a set of items (i.e. individual confirmations) while $\mathcal{B}_s$ and $\mathcal{B}_w$ are just arrays of bits that describe the condition for selection. Additionally, we use a fixed fill rate of set bits to avoid the use of filters with all/most of 1s. Therefore in case of $\mathcal{B}_c$, if there are not enough bits after inserting all necessary individual confirmations, the witnesses can add more random bits to reach the fill rate.

Figure 6.1 describes the main steps of our protocol to verify new blocks at a device. Each verification round starts at step 0 when device D informs its server S the selection filter $\mathcal{B}_s$ for the upcoming block. When the block arrives, each witness computes the whitelist $\mathcal{B}_w$ to determine the set of devices available to them and sends out confirmation filters $\mathcal{B}_c$ accordingly. S then selects only confirmations from witnesses that satisfy the condition $\mathcal{B}_s$ and relays them to D together with the new block header. The block is accepted by D if it is confirmed by η witnesses. If D does not have enough space to store the header after verifying it using the selected confirmations, D can "stamp" the header and sends it back to S for storage. The stamped header can be a message authentication code (e.g. HMAC) computed using the device's own secret to prevent the server from altering the header.

6.2.2 Threat model

As discussed earlier, a node needs at least the block headers in order to verify a transaction. It also needs to be sure that the headers are valid. While consensus protocols (e.g. proof of work) can provide agreement on the current state of the blockchain, a resource-constrained device with limited bandwidth may not be able to handle all of the peer-to-peer message exchange and communication during the consensus phase. As a result, a gateway between the devices and blockchain network cannot be avoided completely. We consider an extreme scenario where D has very limited resource, hence it needs to frequently receive not only new block headers but also confirmation messages from the gateway S. It should be emphasized that the gateway is not obligatory if the device has reasonable storage and networking capabilities.

We assume that S can be compromised and create false blocks and confirmations to D. Our assumption includes the possibility of a man-in-the-middle attack, which can also make D receive false information.

Table 6.1: Notations

m_X	Size of Bloom filter X.
f_X	Fill rate of Bloom filter $\mathcal{B}_X$, i.e. the fraction of bit 1s in the filter.
k_X	Number of bits used per item in Bloom filter $\mathcal{B}_X$.
N_W	Total number of witnesses in the network.
N_D	Total number of devices in the network.
n_c	Capacity of confirmation filter $\mathcal{B}_c$
η	Required number of selected witnesses.

6.3 Protocol details

6.3.1 Witness advertisement

Any node that wants to become a witness can advertise itself by posting its ID to the blockchain. To encourage nodes to be witnesses, the devices can pay a small reward to every selected witness. On the other hand, witnesses also need to make a deposit along with the advertisement. The deposit amount will be the initial weight of each witness to prevent a Sybil attack and can be returned when the witness decides to retire. It is assumed that the devices also publish their IDs during some bootstrap stage. We use public keys as the IDs, thus throughout the chapter ID/public key may be used interchangeably.

It is important that when a device is newly launched, it has no knowledge of any witness. Therefore, we suppose at this bootstrap stage, the device would require a trusted server or some designated witnesses to download previous block headers in order to synchronize with the network. As long as the bootstrap is monitored, the device can receive new blocks reliably with the support of new witnesses.

6.3.2 Witness selection

Simple selection

The selection of witnesses takes place off-chain and only involves the device D and server S. First D selects a random seed to generate a condition for choosing witnesses. S then picks up a list of witnesses that match the condition and sends it back to D. For each selected witness, S also attaches the block that includes the witness's advertisement transaction as a proof that the witness is valid.

The random condition is represented by a Bloom filter $\mathcal{B}_s$ with a fill rate f_s, thus a witness is selected if its ID matches k_s bits of the filter. Therefore, the probability of being selected of each witness is

$$p_s = f_s^{k_s} \tag{6.1}$$

Let N_W be the total number of witnesses in the network, n_W be the number of witnesses selected by $\mathcal{B}_s$ and η be the number of witnesses that the devices requires to accept a new block. Since each witness is selected with the same probability p_s, n_W follows a Binomial distribution $B(N_W, p_s)$. Hence, the probability that less than η witnesses are selected is

$$Pr(n_W \leq \eta) = \binom{N_W}{\eta} \times {p_s}^{\eta} \times (1 - p_s)^{N_W - \eta} \tag{6.2}$$

Since there should be at least η selected, with a given value of N_W, η and f_s, D can adjust k_s to make the probability negligible. Figure 6.2 plots the CDF as a function of k_s with $\eta = 16$ and N_W from 1000 to 10000. Based on the figure, one can choose $k_s < 6$ to maintain a negligible probability as the network grows.

To prevent S from selecting witnesses in its favor, in each round of selection, D generates a new filter $\mathcal{B}_s$ and a random number r using a secret seed. A witness with ID $\mathcal{W}$ is identified by a codeword $\mathcal{H}(r, \mathcal{W})$ where $\mathcal{H}$ is a pseudo-random function. There are various ways to generate a random filter. A possible method is described by procedure GenSelectionFilter()

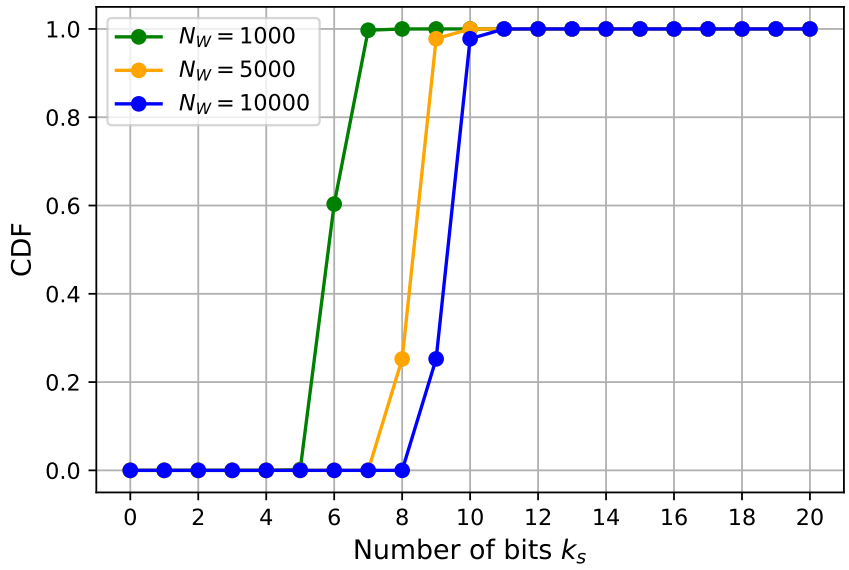


Figure 6.2: Probability that there are less than $\eta = 16$ witnesses selected as a function of k_s.

in Algorithm 1, which repeatedly inserts the hashes computed by a secret s held by the device and a random number to the filter until it reaches the fill rate f_s .

Witness selection based on weights

By the above selection, witnesses are selected with equal probability $p_s = {f_s}^{k_s}$, hence it is vulnerable to a Sybil attack when an adversary can create as many witnesses as they want and increase their probability of being chosen. Let $W = \sum_{i=1}^{N} w_i$ be the cumulative weight of all witnesses where w_i is the weight of witness i, a weight-based selection can be implemented by treating each unit of w_i as a sub-witness, hence there will be W sub-witnesses in total and equation (2) can be changed to

$$Pr(n_W \leq \eta) = \binom{W}{\eta} \times {p_s}^{\eta} \times (1 - p_s)^{W-\eta} \tag{6.3}$$

Procedure SelectByWeight() in Algorithm 3 describes the weight-based selection, where a witness $\mathcal{W}$ with w units of weight is simulated as w sub-witnesses, i.e. it has w chances to be selected.

It is noteworthy that for such weight-based selection, a device also needs to update

Algorithm 3 GenSelectionFilter() is to generate a random $\mathcal{B}_s$ with given m_s, f_s and a secret s. SelectByWeight() is to select a witness with ID $\mathcal{W}$ and weight w, k_s is the number of bits needed to match with Bloom filter $\mathcal{B}_s$.

procedure GenSelectionFilter(s, m_s, f_s)
    $\mathcal{B}_s \leftarrow empty$
    $i \leftarrow 0$
    $r \leftarrow rand()$
    $k = \lfloor \frac{hashlen}{log_2 m_s} \rfloor$
    while $\mathcal{B}_s.CountSetBits() < m_s \times f_s$ **do**
        $l \leftarrow m_s \times f_s - \mathcal{B}_s.CountSetBits()$
        if $r \geq k_s$ **then**
            Insert $\mathcal{H}(i, r, s)$ to $\mathcal{B}_s$ using k bits
        else
            Insert $\mathcal{H}(i, r, s)$ to $\mathcal{B}_s$ using l bits
        end if
        $i \leftarrow i + 1$
    end while
    return $\mathcal{B}_s$
end procedure
procedure SelectByWeight($\mathcal{W}, w, k_s, \mathcal{B}_s$)
    $i \leftarrow 0$
    $r \leftarrow rand()$
    while $i < w$ **do**
        $hash \leftarrow \mathcal{H}(r, \mathcal{W}, i)$
        if $hash$ has k_s bits match with $\mathcal{B}_s$ **then**
            return i
        else
            $i \leftarrow i + 1$
        end if
    end while
    return -1
end procedure

the witnesses' weights correctly. As addressed in section III-A, the witnesses deposit an amount as the initial weight and are rewarded for each time they are selected. All of these events are recorded on the blockchain as transactions. Therefore, a witness can include in its confirmation message a flag that indicates the number of reward transactions existing in the new block. When validating the block, D can receive these transactions from its server, validate them using Merkle proofs [61] as in Bitcoin's SPV method [65] and then update the weights accordingly. Like the stamped block headers, if D does not have enough space to maintain the weights locally, it can also stamp and store them on the server instead.

6.3.3 Confirmation messages

A witness confirms a new block header by broadcasting a confirmation message. Since we assume that the server S will deliver any message from the blockchain to their devices, the messages need to be authenticated to make sure that they come from valid witnesses. Although it would be ideal to use signed messages, verifying signatures with public key cryptography could result in high computation overhead at the resource-constrained devices, especially when the rate of incoming blocks becomes high. On the other hand, symmetric key approaches are more lightweight, however they are not suitable for message broadcasting/multicasting as it would require a lot of bandwidth for witnesses to send one-to-one confirmations to all of their devices. Therefore, to save the bandwidth as well as reduce the computation overhead, we have the witnesses to include all of the individual confirmations to a Bloom filter and broadcast it as a single confirmation message.

Recall that both witnesses and devices have their public keys on the blockchain, hence each pair of witness and device can compute a Diffie-Hellman (DH) shared key s by themselves. Let $\mathbf{s} = \{s_i\}$ be the set of DH keys for all of devices witness $\mathcal{W}$ would like to serve. For each block header B_j, $\mathcal{W}$ computes a codeword $c_{ij} = \mathcal{H}(s_i, B_j)$ where $\mathcal{H}$ is a pseudo-random function and insert those values to confirmation filter $\mathcal{B}_c$, which will be broadcast then together with the ID of $\mathcal{W}$ as a confirmation message.

Algorithm 4 To generate a confirmation message $\mathcal{B}_c$ with parameters for a new block B using a set of shared keys $\mathbf{s}$

procedure GENCONFIRMATION($k_c, B, \mathbf{s}$)
    $\mathcal{B}_c \leftarrow \emptyset$
    for each $s_i \in \mathbf{s}$ **do**
        $c \leftarrow \mathcal{H}(s_i, B)$
        Insert c to $\mathcal{B}_c$ using k_c bits
    end for
    return $\mathcal{B}_c$
end procedure

To validate a new block, a device D checks if the block header is included in every $\mathcal{B}_c$ sent by its selected witnesses using the same shared DH keys. Due to the limited resource for both memory storage and computation, it is not practical to have the device store keys for all of the witnesses nor recompute the shared keys for every validation. Therefore, the keys can be encrypted and then stored on the server S instead once it is computed by D.

With a fixed fill rate f_c, the false positive rate, i.e. the probability that a block header can be falsely recognized in a single filter is ${f_c}^{k_c}$. As a block header needs confirmations from η witnesses, the probability that a device may accept a false block is exponentially reduced to

$$p = {f_c}^{k_c \times \eta} \tag{6.4}$$

Since the number of devices that can be included in $\mathcal{B}_c$ is limited by k_c, we can select more witnesses to maintain a reasonable false positive rate while still keeping k_c low to increase the capacity of $\mathcal{B}_c$. In the lower part of Figure 6.3, we plot n (denoted by ■) with respect to $p = 2^{-128}$, which we suppose should be the minimum threshold to maintain reasonable security (since this is also the probability of finding a 128-bit key). On the other hand, with pre-defined (m_c, k_c, f_c), according to [81], the capacity of $\mathcal{B}_c$ can be approximated by

$$n_c = -\frac{m_c}{k_c} ln\,(1 - f_c) \tag{6.5}$$

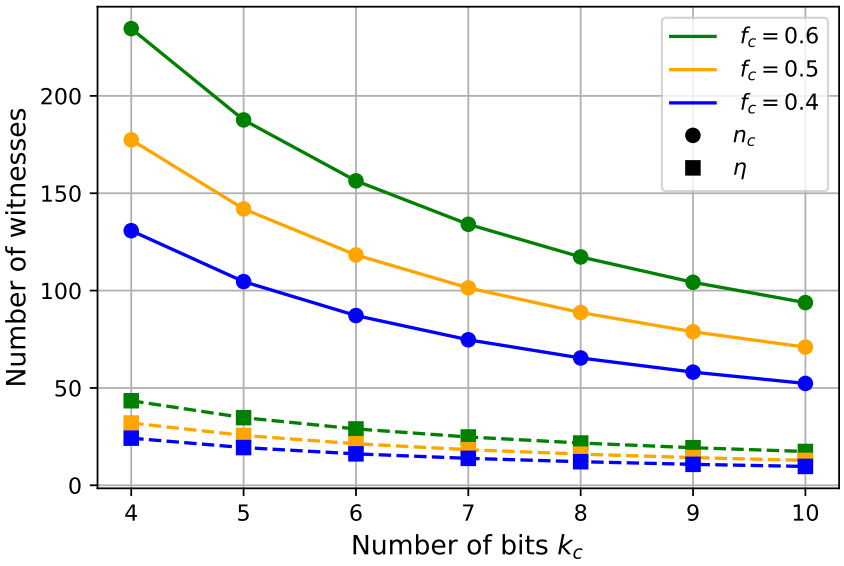


Figure 6.3: This figure plots the size of witness set (η) with respect to $m_c = 1024$ and the number of devices that can fit to a $\mathcal{B}_c$ (n_c) with respect to false positive rate $p = 2^{-128}$.

In the same figure, we also plot n_c (denoted by $\bullet$) for a 1024-bit filter (i.e. $m_c = 1024$). As shown in the figure, a fill rate of 50% and 7-9 bits used seem to provide a good balance between n_c and η, since we would not need too many witnesses but also want to include to $\mathcal{B}_c$ as many devices as possible.

6.3.4 Whitelist filter

Since n_c would be much smaller than the total number of devices in the whole network, witnesses may try to maximize their profit by sending as many confirmations as possible. To prevent the network from being flooded with too many messages, we use a second filter, denoted by $\mathcal{B}_w$, as a "whitelist" that is announced at the beginning of each round. Unlike $\mathcal{B}_s$ which is created by the devices, when a new block is received, each witness computes its own whitelist based on its ID and the new block hash. A device can be added to the witness's confirmation filter $\mathcal{B}_c$ if its ID has k_w bits in $\mathcal{B}_w$. The generation of $\mathcal{B}_w$ is described in Algorithm 5.

Assuming every device has the same probability p_w to be included in the whitelist, the

Algorithm 5 To compute the whitelist filter $\mathcal{B}_w$ with parameters (m_w, f_w, k_w) for witness $\mathcal{W}$ and new block B

```
1: procedure GenWhitelist(m_w, f_w, k_w, B, W)
2:     B_w ← ∅
3:     i ← 0
4:     while B_w.CountSetBits() < m_w × f_w do
5:         r ← m_b × f_w − B_w.CountSetBits()
6:         if r ≥ k_w then
7:             Insert H(i, B, W) to B_w using k_b bits
8:         else
9:             Insert H(i, B, W) to B_w using r bits
10:        end if
11:        i ← i + 1
12:    end while
13:    return B_w
14: end procedure
```

expected number of devices that are available to a witness is $p_w \times N_D$, where N_D is the total number of devices. Hence, with a certain n_c, p_w can be chosen such that

$$p_w = \frac{n_c}{N_D} \tag{6.6}$$

Since the whitelist is also a Bloom filter, it is characterized by the tuple (m_w, f_w, k_w). A witness can only pass if it has k_w bits in $\mathcal{B}_w$, the whitelist probability is

$$p_w = {f_w}^{k_w} \tag{6.7}$$

Putting (4) - (7) together, the condition to choose the parameters of the whitelist filter will be

$${f_w}^{k_w} = \frac{n_c}{N_D} = -\frac{m_c}{N_D \times k_c} ln\,(1 - f_c) \tag{6.8}$$

It is important that with the whitelist, the expected number of witnesses that can serve a certain device is reduced to $\frac{n_c}{N_D} \times N_W$. Hence, if the number of devices is much larger than the number of witnesses, i.e. $\frac{N_D}{N_W} > n_c$, most of the devices will have no witness available to go to the selection stage. However, since we are using weighted witnesses, by dividing the

weight unit, we can increase the number of sub-witnesses and narrow down the ratio. In practice, we believe that the gap between the two numbers would not be too big, as nodes are always incentivized to become witnesses. Additionally, devices with the same owner may also share a group ID on the blockchain instead of individual ones.

6.3.5 Network dynamics

Not enough confirmations

Since our method is probabilistic, there is still a small probability that less than η witnesses satisfy the condition. Another possibility that there are not enough confirmations is when forks happen, i.e. more than one valid block are produced nearly at the same time, which can result in multiple versions of the blockchain among nodes. To deal with such cases, instead of confirming 1 block at a time, the witnesses can compute their codewords for the confirmation filter $\mathcal{B}_c$ using the n most recent blocks in their main chain [1]. Hence, if a device does not get enough confirmations for a new block, it can delay the validation until the next round, when it can receive more confirmations.

Changes in N_W and N_D

In order to determine N_W and N_D, we require both witnesses and devices to register their IDs on the blockchain. A witness leaving can be detected by a retirement transaction that allows the witness to withdraw its deposit. Hence, like regular miners, the witnesses are incentivized to be online most of the time.

Unlike the witnesses, IoT devices can be intermittently disconnected. In such case, the gateway can save all of the confirmations from the witnesses that pass the first whitelist $\mathcal{B}_w$, so that when the device is back, it can resume the validation process by randomly selecting a subset of saved confirmations.

[1] As a result, the codeword c in Algorithm 2 can be computed as $c = \mathcal{H}(s_i, B_{i-n}|...B_{i-1}|B_i)$ where $\{B_{i-n}, \ldots, B_i\}$ are the n most recent blocks.

6.3.6 Example of parameter selection

Suppose the network consists of 10 000 witnesses with equal weight and 10 000 devices, i.e. $N_W = N_D = 10\,000$. First for $\mathcal{B}_c$, we choose $m_c = 1024, f_c = 0.5$ and $k_c = 8$. Hence, by Eq. (4), to achieve the false positive rate $p = 2^{-128}$, we would need confirmations from $\eta = 16$ witnesses. By Eq. (5), the estimated capacity of $\mathcal{B}_c$ is $n_c = 88$.

To have at maximum 88 out of 10 000 devices pass the whitelist $\mathcal{B}_w$, by Eq. (8) we choose $f_w = 0.5$ and $k_w = 7$. With such parameters, the expected number of witnesses remains for each device after the whitelist stage is $0.5^7 \times N_W = 78$. To have at least $\eta = 16$ witnesses selected out of 78 witnesses, a device can choose $k_s = 2$ and $f_s = 0.5$.

6.3.7 Operations on IoT devices

To verify a confirmation filter $\mathcal{B}_c$ from a witness $\mathcal{W}$, a device needs to perform the following steps:

1. Check if it matches $\mathcal{W}$'s whitelist.

2. Retrieve the shared key with $\mathcal{W}$ from server S, compute the corresponding codeword for the header and check if it is included in $\mathcal{B}_c$.

Additionally, if the block contains advertisements of new witnesses, the device also needs to verify these advertisements using Merkle proof and calculate new shared keys. We describe the details of all these steps in Algorithm 4.

6.4 Security analysis

As addressed in Section 6.2.2, we assumes that the gateway server S can deliver false information to device D. Since the condition to select witnesses is generated by D itself, in order to make D accept a false block, S must have the selected witnesses include the false block in their confirmation filters $\mathcal{B}_c$. Recall that a shared key between D and the

Algorithm 6 Verification of a new block header B using a set of witnesses and their corresponding confirmations $\{B_{c_i}, \mathcal{W}_i\}$.

procedure VERIFYBLOCK($B, \{\mathcal{B}_{c_i}, \mathcal{W}_i\}$)
    for each $\mathcal{B}_{c_i}, \mathcal{W}_i$ **do**
        $\mathcal{B}_{w_i} \leftarrow GenWhitelist(B, m_w, f_w, k_w, \mathcal{W}_i)$
        if $D \notin \mathcal{B}_{w_i}$ **then**
            return false
        else
            Retrieve s_i from S
            $c_i = \mathcal{H}(s_i, B)$
            if $c_i \notin \mathcal{B}_{c_i}$ **then**
                return false
            end if
        end if
    end for
    if B has tx_{adv} **then**
        for each tx_{adv_i} **do**
            $UpdateWitness(tx_{adv_i})$
        end for
    end if
end procedure

procedure UPDATEWITNESS(B, tx_{adv})
    $\pi \leftarrow RetrieveMerkleProof(B, tx_{adv})$
    if $VerifyTx(tx_{adv}, \pi) == true$ **then**
        $\mathcal{W} \leftarrow RetrieveWitnessID(tx_{adv})$
        $s = DiffieHellman(\mathcal{W}, D)$
        Send $Encrypt(s)$ to S
    end if
end procedure

witnesses is used to generate $\mathcal{B}_c$, a successful attack depends on whether S can compromise the witnesses or not.

If S cannot compromise the selected witnesses, it is unable to know the shared keys. Therefore, it may have a false block accepted in 2 ways:

1. S alters all of the selected $\mathcal{B}_c$'s and finds a false block header that matches all of the fake filters. Since S does not know the secret shared keys, it is unable to know what bits are set for the block header, therefore the probability that S could find η $\mathcal{B}_c$'s that includes the header is $\binom{m_c}{k_c}^{-\eta}$. For a 1024-bit filter with $k_c = 8$ and $\eta = 16$, the probability is negligible.

2. S keeps the true selected $\mathcal{B}_c$'s but is able to find a false positive block header that matches all of the $\mathcal{B}_c$'s. We have shown in section 6.3.3 that by properly choosing parameters for $\mathcal{B}_c$ as well as the required number of witnesses η, the probability that S can successfully modify all of the selected $\mathcal{B}_c$ can be as low as 2^{-128}.

If all of the selected witnesses are compromised by S [2], they can send confirmations for the false block. In order to be selected, a witness $\mathcal{W}$ has to pass both the whitelist $\mathcal{B}_w$ and selection filter $\mathcal{B}_s$. For simplicity, we assume that all witnesses have equal weight. Thus, they have the same probability p to pass both filters. Since $\mathcal{B}_s$'s parameters are chosen based on $\mathcal{B}_w$, p can be calculated as follows:

$$\begin{aligned} p &= Pr(\text{pass } \mathcal{B}_s | \text{pass } \mathcal{B}_w) \times Pr(\text{pass } \mathcal{B}_w) \\ &= {f_s}^{k_s} \times {f_w}^{k_w} \end{aligned}$$

Assuming S can hold X out of N_W witnesses, the probability that exactly η compromised witnesses can be selected is

$$p_{compromised} = \binom{X}{\eta} \times p^{\eta}(1-p)^{X-\eta} \tag{6.9}$$

In table 6.2 we calculate $p_{compromised}$ for the example in section 6.3.6, i.e. $\eta = 16, N_W = 10000$

[2] Here we focus on the edge node S, but the analysis can be applied to any adversary.

and $p = 0.5^2 \times 0.5^7$. As shown in the table, unless S can acquire more than 50% witnesses, the compromised probability is quite small.

6.5 Evaluation

As our method is based on random selection, it is important to evaluate how the confirmations are propagated through the network and if a node can receive a sufficient number of confirmations on time in order to carry out the selection. Therefore, we use a simulator to simulate a Bitcoin network in different conditions, i.e. the rate of block generation and block size. In addition, to show the feasibility of our mechanism, we also conduct an experiment using a low-power device.

Table 6.2: Probability that the set of selected witnesses is from an adversary given $n_w = 16, N_W = 10000, p = 0.5^9$

% of compromised witnesses	$p_{compromised}$
10%	2.7746×10^{-10}
20%	2.7345×10^{-6}
30%	0.0002594
40%	0.0037
50%	0.01872
60%	0.049205
70%	0.082288
80%	0.098874
90%	0.092296

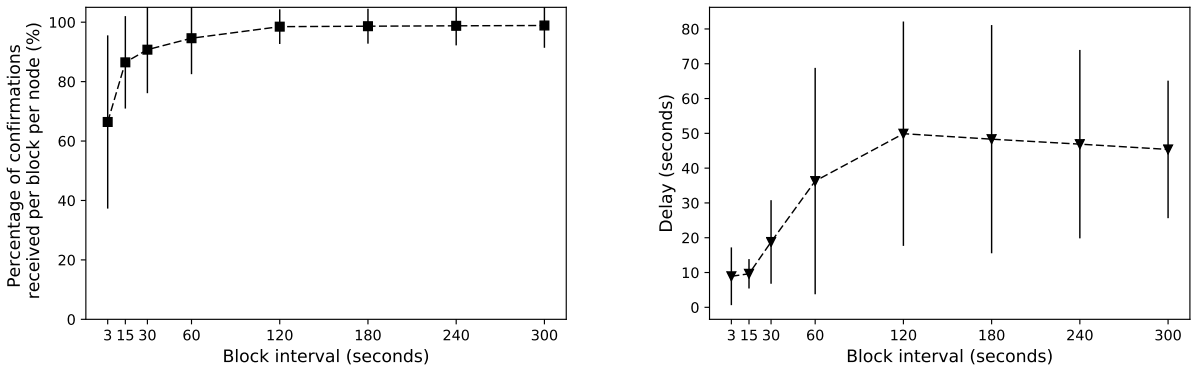
6.5.1 Simulation to evaluate confirmation throughput

We use a customized version of the Bitcoin Simulator [41] on ns-3 to evaluate the throughput of the confirmation messages. We suppose that the witnesses can form an overlay network between themselves for faster communication. Using an overlay network on top of the existing blockchain is not new. For instance, Bitcoin's Lightning network [74] is an overlay network that supports instant payments and improves transaction throughput. Therefore, we simulate a network that consists of 100 witnesses that are also Bitcoin nodes.

Each confirmation message is 192 bytes, including a 1024-bit confirmation filter $\mathcal{B}_c$ and the witness's ID and signature, which is 32 bytes each. Each witness sends out a confirmation after validating a new block. The confirmations are propagated by gossiping, i.e. nodes relay the messages if they have not heard of them before. For block generation, there are additionally 16 miners to produce new blocks. However, the miners are not witnesses, since they are only to trigger the selection process. Each simulation runs for 200 consecutive blocks.

We assume that the 100 witnesses send confirmations to the same set of devices and each of them is also a server to deliver the confirmations to some devices. Therefore, we measure the total number of confirmations each witness receives from the other witnesses for each new block. To account for delay, we only consider confirmations that are received before a new block arrives. Figure 6.4a plots the average percentage of confirmations received per block with the block interval varied from 3 seconds to 5 minutes. The block size is fixed to 500 KB for every interval. As shown in the figure, the higher the block rate, the less confirmations arrive on time. In addition, the throughput also has more volatility, as described by the error bars, when blocks arrive fast. This is expected as a long block interval would allow nodes to have more time to receive the confirmations. Moreover, since we set a fixed block size, it would take nodes a similar amount of time to receive a complete block and validate it before they can send out confirmations regardless of block intervals. In Figure 6.4b, we plot the delay to receive all the "on-time" confirmations. As the interval is longer, the delay becomes stable at around 1 minute. Since the block size is the same for every interval, based on Figure 5a, it can be inferred that this is also the delay for nodes to receive more than 90% of confirmations.

To evaluate the impact of block size, in our second experiment, the block size is set to follow the real distribution of Bitcoin's network, which is estimated by data from a Bitcoin explorer website [41]. Following such distribution, the block size decreases as the block rate becomes faster. The mean block size for each interval is provided in Table 6.3. For instance,



(a) Percentage of confirmations

(b) Confirmation delay

Figure 6.4: Average percentage of confirmations per block and confirmation delay with block size = 500 KB.

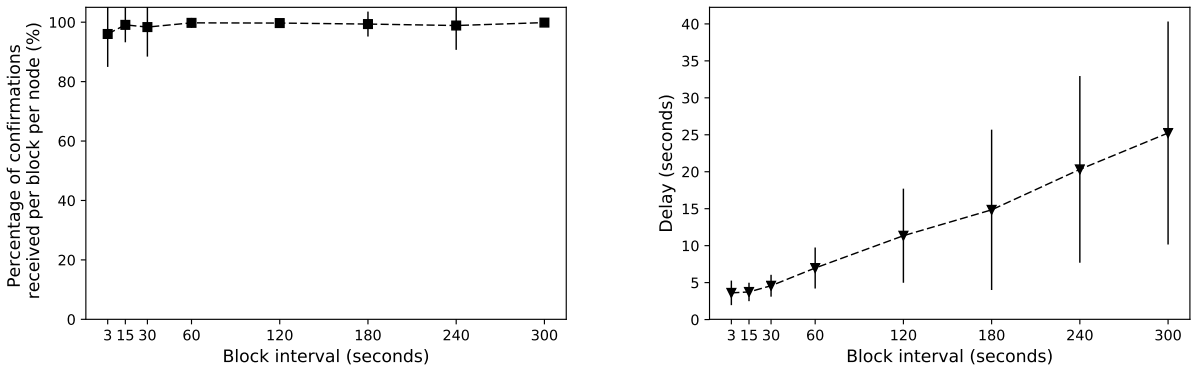
the mean block size when the block interval is 3 seconds is only 2.4 KB. Therefore, blocks can be propagated and validated faster. As a result, as shown in Figure 6.5, even when the block interval is short, nodes can still receive nearly all of the confirmations on time. The delay is always within the block intervals as well.

Table 6.3: Block size according to Bitcoin block distribution

Mean block interval	Mean block size (KB)
3 seconds	2.4
15 seconds	12.6
30 seconds	23.8
1 minute	50.2
2 minutes	105.9
3 minutes	149.6
4 minutes	213.8
5 minutes	264.0

6.5.2 Performance on IoT devices

We conduct an experiment to evaluate the performance on a low-power device, which is an Arduino MKR1000 with 32KB RAM. It is connected via Wi-fi to a Dell laptop that acts as the server. Since we assume that the selected confirmations and new block headers are relayed to the device via its server, no blockchain network activities except block rate may



(a) Percentage of confirmations

(b) Confirmation delay

Figure 6.5: Average percentage of confirmations per block and confirmation delay when block size varies depending on block rate.

affect the performance of the device. Therefore, we have the server simulate the witnesses and subscribe to the Bitcoin explorer website blockchain.info via websocket to receive new block headers instead of connecting to a real blockchain network. As shown in table IV, since it takes only 19 ms to verify one confirmation filter, then if 16 witnesses are required, the total time to verify a new block header is 304 ms. This is significantly smaller compared to the use of signatures, which could take 1.4 second for a single verification.

The most time-consuming operation in our mechanism is to compute a shared key between the device and a witness. Therefore, if the new block consists more than one new witness advertisement, the witness update process can take up to several seconds. However, the computation of shared keys can be delayed until the witnesses are actually selected to confirm new blocks.

Although the simulation shows that most confirmations can be received on time even when the block interval is short, we suggest that for blockchains with high block rates and devices that are accessed frequently, it is better to perform validation for every n blocks instead of one block at a time, as discussed in section 6.3.5, to reduce the workload on the devices.

Table 6.4: Performance on Arduino MKR1000

Action	Sub action	Time
Update witness	Verify Merkle proof	14 ms.
	Compute one shared key	460 ms
Verify block	Verify one $\mathcal{B}_c$	19 ms
Verify signature		1451 ms

6.6 Conclusion

We propose a lightweight method for resource-constrained IoT devices to validate new blocks with the help of witnesses from the blockchain network. Using Bloom filters, the IoT devices can perform validation without complex computation. It is shown by both our simulation and experiment on a low-power device that our method is achievable.

Chapter 7

Summary and Future Work

7.1 Summary

The limited storage and computational power of some IoT devices have presented unique challenges to the access control problem. Particularly, an access control framework for IoT should be lightweight, flexible and supports fine-grained policies. In addition, a decentralized architecture may be more suitable for the distributed nature of IoT environment than the traditional centralized approach.

In Chapter 3 we present a lightweight authorization protocol that provides flexible device sharing using Bloom filters. We leverage the non-reversibility of Bloom filters so that users can generate new permissions by adding valid access rights to their current keys. We focus on key generation and validation and assume a secure environment for key exchange. The method is designed for closed and small-scale environments like smart homes where there is trust between users and their identities are easy to manage by a central authority.

As discussed in Chapter 1, a secure exchange typically involves a trusted third-party, which does not only present a single point of failure but also induce privacy concerns. Therefore, we seek for a decentralized solution by exploring the feasibility of applying blockchain to design a decentralized access control framework named CapChain for IoT environments in Chapter 4. In our system, a capability (or access token) is treated as a digital asset and can be transferred to users via transactions on CapChain. We solve the privacy problem

by applying multiple techniques from existing cryptocurrency systems to hide the details of transactions. Next, in Chapter 5, we explore the capability sharing to create conditional tasks for automation, such that the capability can only be used when one or several conditions are satisfied. Unlike other existing platforms for automated tasks, we focus on user privacy in the sense that condition information can be hidden from other blockchain users.

With the main focus on privacy in Chapter 4 and 5, we assume an existing blockchain architecture with some consensus protocol (e.g. proof of work). With limited resources, performing consensus is not an easy task for IoT devices. As a result, to update new blocks, the devices need to rely on some other more powerful nodes that may not be trustworthy. Therefore, in Chapter 6, we propose a lightweight block validation method for resource-constrained devices with the help of a random set of other nodes. Through both simulation and experiments, we show that the method is applicable to IoT applications.

7.2 Future Work

Since blockchain is a fairly new technology, there are several aspects for enhancement in IoT domains:

Consensus and incentives As discussed in Chapter 2, proof of work (PoW) currently produces latency in transaction confirmation. For CapChain, this means that a capability token may not be used immediately without some waiting time after the transaction is issued. This is especially critical for revocation which should be effective instantly. Moreover, PoW requires hardware power which is almost impossible for IoT devices. In addition, most of consensus protocols rely on a certain economic benefit that incentivizes users to participate and behave honestly. Since our objective is not to build a cryptocurrency, a bounty mechanism like [17] in which IoT data is provided as a reward seems to be a good incentive. For example, device owners can publish a special capability for data acquisition and attach it to their transactions as a reward to miners.

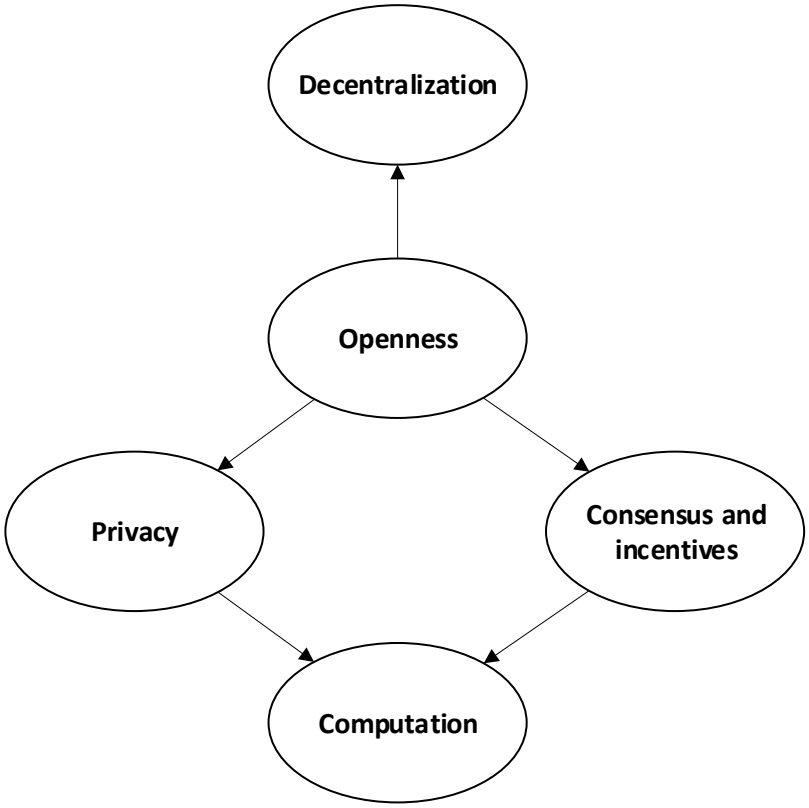


Figure 7.1: Considerations for an IoT blockchain. The network openness (i.e. public or private mode) decides the level of decentralization and privacy as well as the type of consensus protocol and incentives. On the other hand, computational overhead also depends on the privacy level and the consensus protocol.

Although our lightweight validation method for IoT devices in Chapter 6 works as a mitigation on top of the main consensus protocol, it would be interesting to find a more suitable alternative consensus protocol for IoT that is not only faster but also less computationally expensive.

Blockchain architecture Permissioned blockchain refers to a type of blockchain which only allows certain peers. On the other hand, permissionless or public blockchain does not have any restriction on who can participate. In [45] Hardjono et al. introduce a definition of *semi-permissioned* blockchain, where only authorized nodes can update the blockchain but anyone can read and validate transactions. In the IoT context, we find this type of

in-between openness more rational as the identities of IoT devices should be validated by some authorized parties such as manufacturers. An interesting research problem would be to design a blockchain architecture that can support the entire life cycle of IoT devices, from manufacturing, deployment to functioning and retirement.

It should be noted that the above aspects are closely related to each other and introduce trade-off to privacy, computation and decentralization, as illustrated in Figure 7.1. If we decrease the openness of the network by making the blockchain private, BFT-style consensus protocols can be used without any incentive but the system also becomes more centralized. On the other hand, public blockchains are fully decentralized but hard to manage and require economic incentives. Public blockchains also mean that more compute-intensive cryptography techniques are needed to preserve privacy while still allowing nodes to validate transactions. To reduce computation, evaluation of sensitive information could be off-loaded to local domain, which, in turn, may require central authorities. Therefore, there is probably no "one size fits all" solution but a more flexible design that can give users options to choose the level of privacy, computation or decentralization depending on their needs.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] https://cloudblogs.microsoft.com/industry-blog/manufacturing/2017/03/28/accelerate-value-deliver-transformative-experiences-with-microsoft-connected-consumer-devices.

[2] https://www.businessinsider.com/internet-of-things-definition.

[3] https://slock.it.

[4] https://chain.com/docs/1.2/protocol/papers/whitepaper. [Online; accessed 22-Sep-2017].

[5] https://github.com/kmackay/micro-ecc.

[6] https://ifttt.com/discover.

[7] https://github.com/solidblu1992/RingCTToken.

[8] Blackcoin. https://blackcoin.co. [accessed 19-Nov-2017].

[9] Corda. https://docs.corda.net/_static/corda-technical-whitepaper.pdf. [accessed 19-Nov-2017].

[10] Ethereum wiki. https://github.com/ethereum/wiki/wiki. [accessed 19-Nov-2017].

[11] Nxtcoin. https://nxtplatform.org. [accessed 19-Nov-2017].

[12] Peercoin. https://peercoin.net. [accessed 19-Nov-2017].

[13] Vera user guides. http://support.getvera.com/. Accessed: 2017-09-22.

[14] Samsung's SmartThings smart home hub has been down since yesterday for some users. https://www.theverge.com/circuitbreaker/2018/3/13/17115624/samsung-smartthings-outage-over-14-hours-update, 2018. [Online; accessed 19-Aug-2018].

[15] Michael P Andersen, John Kolb, Kaifei Chen, Gabriel Fierro, David E Culler, and Raluca Ada Popa. Wave: A decentralized authorization system for iot via blockchain smart contracts. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2017-234*, 2017.

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

[17] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. Medrec: Using blockchain for medical data access and permission management. In *Open and Big Data (OBD), International Conference on*, pages 25–30. IEEE, 2016.

[18] Lujo Bauer, Scott Garriss, Jonathan M McCune, Michael K Reiter, Jason Rouse, and Peter Rutenbar. *Device-enabled authorization in the Grey system.* Springer, 2005.

[19] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *International Conference on Financial Cryptography and Data Security*, pages 142–157. Springer, 2016.

[20] Daniel J Bernstein. Curve25519: new diffie-hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer, 2006.

[21] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[22] Mic Bowman, Andrea Miele, Michael Steiner, and Bruno Vavala. Private data objects: an overview. *arXiv preprint arXiv:1807.05686*, 2018.

[23] Ernie Brickell and Jiangtao Li. Enhanced privacy id: A direct anonymous attestation scheme with enhanced revocation capabilities. *IEEE Transactions on Dependable and Secure Computing*, 9(3):345–360, 2012.

[24] Paul Brody and Veena Pureswaran. Device democracy: Saving the future of the internet of things. *IBM, September*, 2014.

[25] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world.

[26] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[27] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[28] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution. *CoRR*, abs/1804.05141, 2018.

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

[30] Anton Churyumov. Byteball : A Decentralized System for Storage and Transfer of Value. https://byteball.org/Byteball.pdf. [accessed 19-Nov-2017].

[31] Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Sandeep Tamrakar, and Christian Wachsmann. *SmartTokens: Delegable access control with NFC-enabled smartphones.* Springer, 2012.

[32] Ali Dorri, Salil S Kanhere, and Raja Jurdak. 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.

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

[34] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[35] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security Analysis of Emerging Smart Home Applications. In *Proceedings of the 37th IEEE Symposium on Security and Privacy*, May 2016.

[36] Earlence Fernandes, Amir Rahmati, Jaeyeon Jung, and Atul Prakash. Decoupled-ifttt: Constraining privilege in trigger-action platforms for the internet of things. *arXiv preprint arXiv:1707.00405*, 2017.

[37] Simon N Foley and Guillermo Navarro-Arribas. A bloom filter based model for decentralized authorization. *International Journal of Intelligent Systems*, 28(6):565–582, 2013.

[38] N. Fotiou, T. Kotsonis, G. F. Marias, and G. C. Polyzos. Access control for the internet of things. In *2016 International Workshop on Secure Internet of Things (SIoT)*, pages 29–38, Sept 2016.

[39] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. A decentralized public key infrastructure with identity retention. *IACR Cryptology ePrint Archive*, 2014:803, 2014.

[40] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In *Public Key Cryptography*, volume 4450, pages 181–200. Springer, 2007.

[41] Arthur Gervais, Ghassan Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 23nd ACM SIGSAC Conference on Computer and Communication Security (CCS)*. ACM, 2016.

[42] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *In Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[43] Eu-Jin Goh et al. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

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

[45] Thomas Hardjono and Ned Smith. Cloud-based commissioning of constrained devices using permissioned blockchains. In *Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security*, pages 29–36. ACM, 2016.

[46] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, pages 461–472, New York, NY, USA, 2016. ACM.

[47] D. Hussein, E. Bertin, and V. Frey. A community-driven access control approach in distributed iot environments. *IEEE Communications Magazine*, 55(3):146–153, March 2017.

[48] Simon Josefsson and Ilari Liusvaara. Edwards-curve digital signature algorithm (eddsa). *RFC 8032*, 2017.

[49] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1353–1370, 2018.

[50] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10401 LNCS:357–388, 2017.

[51] Tiffany Hyun-Jin Kim, Lujo Bauer, James Newsome, Adrian Perrig, and Jesse Walker. Access right assignment mechanisms for secure home networks. *Communications and Networks, Journal of*, 13(2):175–186, 2011.

[52] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 839–858. IEEE, 2016.

[53] Jae Kwon. Tendermint: Consensus without mining. https://tendermint.com/static/docs/tendermint.pdf. [accessed 19-Nov-2017].

[54] Kaspersky Lab. Somebody's watching! When cameras are more than just 'smart'. https://ics-cert.kaspersky.com/reports/2018/03/12/somebodys-watching-when-cameras-are-more-than-just-smart/, 2018. [Online; accessed 19-Aug-2018].

[55] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[56] Joonghwan Lee, Jae Woo Seo, Hoon Ko, and Hyoungshick Kim. Tard: Temporary access rights delegation for guest network devices. *Journal of Computer and System Sciences*, 86:59–69, 2017.

[57] Greg Maxwell. Confidential transactions. https://people.xiph.org/~greg/confidential_values.txt, 2015. [Online; accessed 22-Sep-2017].

[58] Greg Maxwell. Confidential transactions (2015). *URL: https://github. com/ElementsProject/elementsproject. github. io/blob/master/confidential_values. md*, 2015.

[59] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, pages 1–26, 2018.

[60] Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless tumbling for transaction privacy. *Proceedings on Privacy Enhancing Technologies*, 2018(2):105–121, 2018.

[61] Ralph C Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, pages 122–122. IEEE, 1980.

[62] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016.

[63] Saraju P Mohanty, Uma Choppali, and Elias Kougianos. Everything you wanted to know about smart cities: The internet of things is the backbone. *IEEE Consumer Electronics Magazine*, 5(3):60–70, 2016.

[64] M. Mössinger, B. Petschkuhn, J. Bauer, R. C. Staudemeyer, M. Wójcik, and H. C. Pöhls. Towards quantifying the cost of a secure iot: Overhead and energy consumption of ecc signatures on an arm-based device. In *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6, June 2016.

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

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

[67] Shen Noether. Ring signature confidential transactions for monero. *IACR Cryptology ePrint Archive*, 2015:1098, 2015.

[68] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016.

[69] S. Notra, M. Siddiqi, H. Habibi Gharakheili, V. Sivaraman, and R. Boreli. An experimental study of security and privacy risks with emerging household appliances. In *Communications and Network Security (CNS), 2014 IEEE Conference on*, pages 79–84, Oct 2014.

[70] 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.

[71] Torben P Pedersen et al. Non-interactive and information-theoretic secure verifiable secret sharing. In *Crypto*, volume 91, pages 129–140. Springer, 1991.

[72] O. J. A. Pinno, A. R. A. Gregio, and L. C. E. De Bona. Controlchain: Blockchain as a central enabler for access control authorizations in the iot. In *IEEE GLOBECOM 2017*, 2017.

[73] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets.

[74] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *See https://lightning. network/lightning-network-paper. pdf*, 2016.

[75] Serguei Popov. The Tangle. https://iota.org/IOTA_Whitepaper.pdf, 2017. [accessed 19-Nov-2017].

[76] Ana Reyna, Cristian Martín, Jaime Chen, Enrique Soler, and Manuel Díaz. On blockchain and its integration with iot. challenges and opportunities. *Future Generation Computer Systems*, 2018.

[77] 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.

[78] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.

[79] L Seitz et al. Authorization for the internet of things using oauth 2.0. *work in progress), December*, 2015.

[80] 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.

[81] S Joshua Swamidass and Pierre Baldi. Mathematical correction for fingerprint similarity measures to improve chemical retrieval. *Journal of chemical information and modeling*, 47(3), 2007.

[82] Blase Ur, Jaeyeon Jung, and Stuart Schechter. The current state of access control for smart devices in homes. In *Workshop on Home Usable Privacy and Security (HUPS)*, 2013.

[83] Nicolas van Saberhagen. Cryptonote v 2.0. https://cryptonote.org/whitepaper.pdf, 2013. [Online; accessed 22-Sep-2017].

[84] Pureswaran Veena, Sanjay Panikkar, Sumabala Nair, and Paul Brody. Empowering the edge-practical insights on a decentralized internet of things. *Empowering the Edge-Practical Insights on a Decentralized Internet of Things. IBM Institute for Business Value*, 17, 2015.

[85] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International Workshop on Open Problems in Network Security*, pages 112–125. Springer, 2015.

[86] Lifei Wei, Haojin Zhu, Zhenfu Cao, Xiaolei Dong, Weiwei Jia, Yunlu Chen, and Athanasios V Vasilakos. Security and privacy for storage and computation in cloud computing. *Information Sciences*, 258:371–386, 2014.

[87] Eric Zeng, Shrirang Mare, and Franziska Roesner. End user security & privacy concerns with smart homes. In *Symposium on Usable Privacy and Security (SOUPS)*, 2017.

[88] J. Zhou, Z. Cao, X. Dong, and A. V. Vasilakos. Security and privacy for cloud-based iot: Challenges. *IEEE Communications Magazine*, 2017.

[89] Feng Zhu, Matt W Mutka, and Lionel M Ni. The master key: A private authentication approach for pervasive computing environments. In *null*, pages 212–221. IEEE, 2006.

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