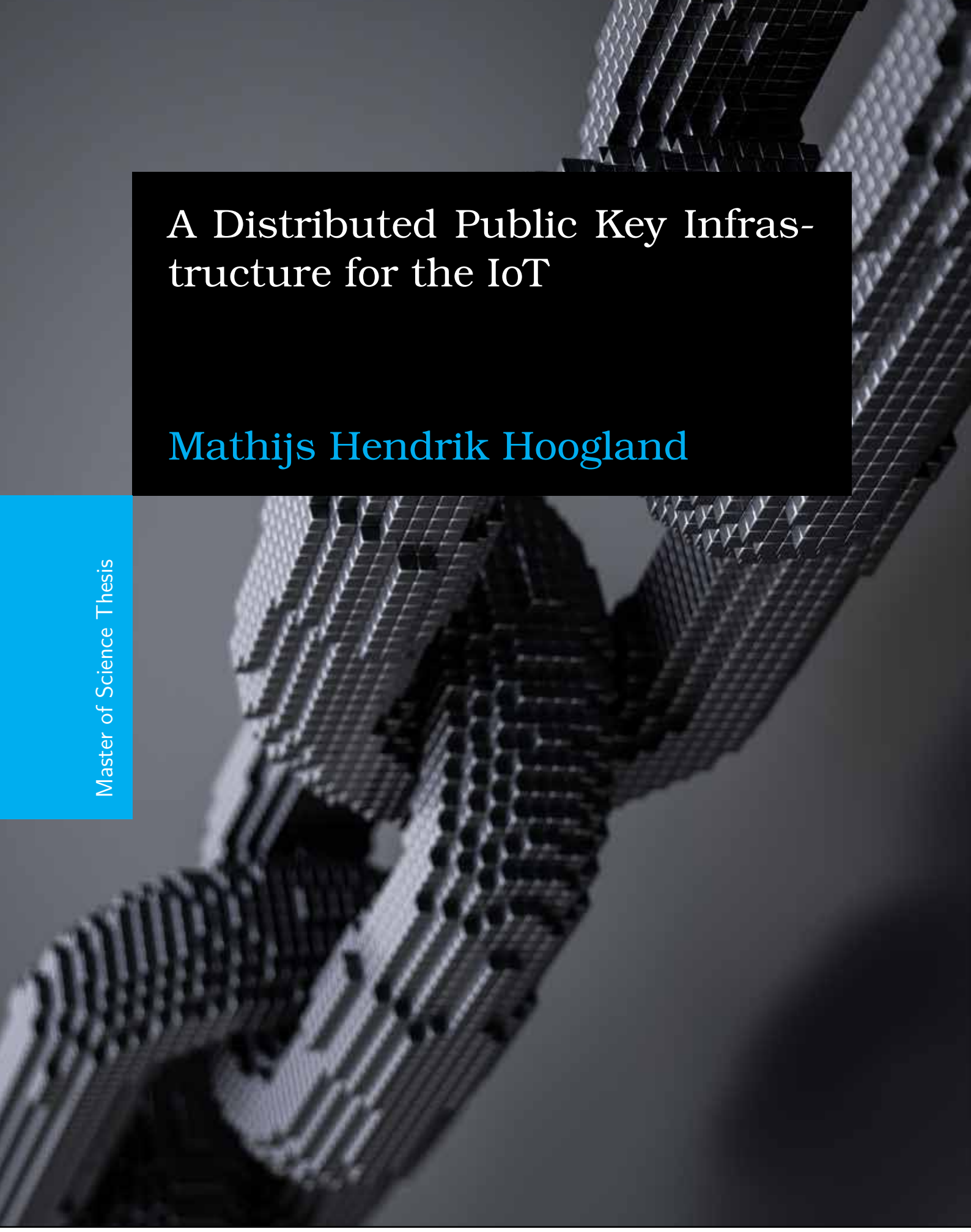

A Distributed Public Key Infrastructure for the IoT

Mathijs Hendrik Hoogland

Master of Science Thesis

TU Delft Delft University of Technology

Faculty of EEMCS

A Distributed Public Key Infrastructure for the IoT

Master of Science Thesis

For the degree of Master of Science in Computer Science with the track Data Science and Technology with Cyber Security specialization at Delft University of Technology

Mathijs Hendrik Hoogland
4237676

Thesis committee
Dr.ir. J.C.A. van der Lubbe
Dr.ir. Z. Erkin
Dr.ir. F.A. Kuipers
ir. S. Hijgenaar

Supervisors
Dr.ir. Z. Erkin
ir. S. Hijgenaar

June 30, 2018

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) · Delft University of Technology

Preface

This graduate thesis researches the possibilities of decentralizing public key infrastructures. The research was carried out at the Cyber Security department of Delft University of Technology. In concludes the master's degree in Computer Science (track Data Science and Technology with Cyber Security specialization).

I want to thank Zeki Erkin, who was my supervisor, for his supervision and advice during this research. Your meetings always helped me in putting a dot on the horizon. I should also mention Oguzhan Ersoy and Sjors Hijgenaar whom both helped me during the project. Although I am an independent worker, and the number of meetings between us was limited, I appreciate the fact that both of you were always there whenever I needed some guidance.

Delft, University of Technology
June 30, 2018

Mathijs Hendrik Hoogland

Abstract

The role of Internet of Things (IoT) devices is becoming larger in our day-to-day life. More and more, devices are used to help with simple tasks in our lives. Little computers are installed in a large variety of consumer products and are often connected to the internet. The large increase of these "resource-constrained" devices that are connected to the internet results in security challenges. Communication between IoT devices often occurs through asymmetric key encryption where keys are distributed through a Public Key Infrastructure (PKI). Traditional PKIs have a number of downsides: they have a single point of failure, provide a difficult revocation scheme and lack transparency.

The emergence of Blockchain technology has provided a new direction for the research on PKIs, solving certain downsides of traditional PKI systems. In this thesis, we present DEcentralized Key INfrastructure (DECKIN). Our solution addresses scalability, key management for recovery and revocation and verification of the identity that is being registered in a fully decentralized context. We do this by building an infrastructure on top of blockchain technology. The system uses cryptographic accumulators to enable resource-constrained devices to look-up if certain identity-key values are correctly registered within the system without requiring these devices to traverse the entire blockchain. Our system also contains an identity validation protocol. This protocol is executed by each device that wants to register its identity in our system and requires the node to successfully interact with other nodes in the network for the duration of a block creation. When a node successfully conducts this interaction, his identity will be confirmed and the registration will be completed. Participants can now lookup his key in the system. Lastly, we use Physical Unclonable Function (PUF) technology to handle key-management challenges in our system. PUF technology uses hardware specific features to create a digital fingerprint for each device. This technique enables devices to recover their secret key without requiring them to store anything locally.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Internet and devices that are connected to the Internet are playing an ever-growing role in our day-to-day life [1]. Not only are we connected to the Internet throughout our phone and computer, the trend these days is to connect all sorts of devices to the Internet to make our life easier. From a sensor that enables us to configure the heating of our house when driving back home, devices that are placed inside our garden to measure the amount of rain to microwaves that enable us to look up cooking instructions on YouTube while working in our kitchen. This category of unconventional devices that are connected to the Internet is known as the IoT [2]. According to Gartner [3], the number of devices connected to the Internet will be 20 billion in 2020.

With more and more devices being connected to the Internet, it can be expected that many of our day-to-day tasks will be aided by small connected computers, and, eventually, probably be executed without any human intervention. Imagine refrigerators ordering food online on behalf of their owner whenever they notice that there is a shortage on milk, pacemakers that get their newest operating system automatically uploaded or door locks that automatically lock the doors of a house whenever a home automation system notices that the owner left the building. All these digital innovations can make our life more comfortable, but they also open up many risks. The IoT consists of and will consist even more, of devices that generate, process, and exchange vast amounts of privacy-sensitive information, revealing a lot of information that could also be interesting for adversaries. This is why these devices are an appealing target for various cyber attacks [4]. The large number of devices with relatively high computational power also makes them an attractive target for attackers seeking to compromise these devices and use them to create large-scale botnets [5]. To make things even worse, it is also shown that, compared to conventional computing systems, IoT systems are at higher security risk because of common vulnerabilities that conventional computers often do not have [6]. Conducted research has shown that IoT devices are among others more insecure than conventional computers [7] because:

1. Lack of experience– IoT device manufacturers are not in the business of cybersecurity.

Up until recently, they often were not even into the computer business. This is why many manufacturers do not have the knowledge to secure their devices sufficiently.

2. Margins– Manufacturers are financially motivated in their actions. Because of this, they often do not prioritize security. Why invest in a security division to support already sold devices, when the return on investment is extremely low?

3. Keeping in line with compliances and regulations– The healthcare industry is a good example of an industry where regulations can hinder IoT security. Healthcare regulations, for example, prescribe the use of certain communication protocols. It takes years, however, for these regulations to change. Even if some of these prescribed protocols are provably insecure.

4. Innovation– Since the IoT domain is new, manufacturers are continually trying to keep up with competition by developing new devices to address the growing interest. Building security from scratch, however, takes a lot of time and in the eyes of manufacturers only slows them down. This is why many manufacturers often do not prioritize security.

A recent example demonstrating a substantial IoT vulnerability being exploited took place in October 2016. Service provider Dyn was targeted by an IoT-botnet called Mirai [8]. This attack took down hundreds of websites, including Reddit, Twitter, Netflix and GitHub, for several hours [9]. Mirai works primarily by spreading its malware to devices like DVRs, webcams, and routers that run on BusyBox. The administrative credentials of other IoT devices are then deduced by brute forcing, relying on a small dictionary of potential username-password pairs [8]. The lack of standardization between manufacturers results in a large number of different protocols and no standard in securing and updating IoT devices. The Mirai incident exemplified how, by exploiting one common IoT vulnerability, the whole world could be impacted. Summarizing, the current security architecture of the IoT has a number of shortcomings and there is room for improvement. One of the proposed enhancements is minimizing the current heavy reliance of IoT devices on a centralized cloud, removing security risks that arise through this reliance.

Blockchain technology Blockchain technology is often suggested as a solution to help improve the drawbacks of centralized architectures by decentralizing them [10]. Blockchain is first seen with the introduction of Bitcoin [11] in 2008. Bitcoin is a digital currency and, initially, research was focused on optimizing digital currencies and improving underlying architectures. Blockchain introduced the possibility of distributed consensus. The blockchain is a P2P ledger that is tamper-proof and contains only authentic information. This ledger is not controlled by any centralized entity. It also enables permanent and verifiable transfer of ownership for digital resources in a decentralized system, we will discuss this technology in more detail in Section 2-2. The properties that were introduced with Blockchain technology in the Bitcoin system can also add value to other domains. Decentralization, for example, can help limit the reliance of IoT devices on the centralized cloud. Multiple angles can be taken when trying to achieve this decentralization. Research has already been done on the potential of decentralizing infrastructures to help combat challenges that arise with the increasing number of IoT devices. Examples of issues that researchers hope to solve by using Blockchain technology by minimizing the reliance on the centralized cloud are [12]:

1. Increasing costs and capacity constraints– Because of the expected increase of connected IoT devices, the network capacity will need to increase. By removing the need for a centralized entity, devices can communicate securely, exchange values with each other, and execute actions automatically through smart contracts. This removes the need to increase network capacity.

2. Improve the architecture– Each IoT device acts as a vulnerability to Distributed denial-of-service (DDoS) attacks, hacking data theft, and remote hijacking. By identifying and verifying the validity of a device's identity, and cryptographically signing and verifying transactions, it can be assured that only a message's originator could have sent it.

3. Prevent cloud server downtime and unavailability of services– By removing the single point of failure, records are stored on many computers and therefore unavailability will be minimized.

4. Remove the susceptibility to manipulation– Information is likely to be manipulated an put to appropriate uses. Blockchain can prevent the possibility of a single malicious node modifying values; the system would reject these changes.

Migrating existing infrastructures to the blockchain, on the other hand, is not as evident as it initially might seem. Since blockchain uses a publicly verifiable ledger, without a third party managing its nodes, the technology also introduces challenges. All data in a blockchain ledger is visible for all nodes in the network, resulting in privacy issues [13]. Pseudonyms are often used to combat these privacy issues. This solution, however, can be considered as just a Band-aid. Pseudonyms are still linkable and, since stored data on the ledger is available forever, identification of a pseudonym and a real-world identity in the future can also result in privacy issues. It is also difficult to assign these pseudonyms without disclosing confidential information for adversaries [13]. Having no trusted third party to control the nodes in a network also results in complications. If access, provided to nodes for certain services in a blockchain, needs to be revoked in a decentralized context, complicated schemes need to be set up that are less trivial compared to revocation schemes in a centralized context. Current solutions suggest using trusted neighbor groups to vote for possible key revocations [14]. Often, a balance needs to be found between privacy and ease of revocation. A last complication that should be taken into account when considering to use blockchain technology for the IoT security is its computational cost. Blockchains are known to be computationally expensive and involve high bandwidth overhead and delays [15]. This can, taking into account that IoT devices often do not have computational power in abundance, make certain proposals unsuitable.

1-1 Public Key Infrastructure landscape

This thesis focuses on designing a decentralized PKI to improve IoT security. PKIs have become the starting point for modern security mechanisms on the Internet [16]. A PKI provides an architecture for entities connected to the Internet to issue and revoke digital certificates. These certificates can be seen as a kind of electronic passport mapping a user's digital signature to its public key [16]. When browsing the web, users unconsciously interact with a PKI architecture on a day-to-day basis. It is used in our Internet infrastructure to

map domain names to IP addresses and is visually displayed as the green secure bar in our Internet browser. Traditionally there are two main approaches to setting up PKIs: Certificate Authority (CA) and Web of Trust (WoT).

The most common approach to PKI is CA-based (or specifically, the X.509 standard) [14]. In this standard, a CA acts as a trusted party, who issues a signed certificate verifying an entity's ownership of a certain public key on request. Image 1-1 shows all the steps in a X.509 standard. In this standard, users need to apply for certificates with their public key at a Registration Authority (RA). If the RA agrees, he confirms the user's identity and passes this information forth to the CA. The CA, in turn, issues a certificate. The Validation Authority (VA) receives information about all issued certificates. A user can now digitally sign a contract with his new certificate, other parties can verify that the user is who he claims to be by looking up if the signature corresponds to the information that the VA has. This standard has a couple of downsides:

1. The largest problem of the CA-based PKI is that there is a single point of failure. If adversaries are able to compromise the CA, they are able to issue certificates for whoever they want.

2. This system lacks transparency: users are not able to see what certificates are issued by the CA. Because of this, a site owner has no way to detect what certificates are issued for his domain. This makes possible breaches hard to detect (although some initiatives like Certificate Transparency (CT) try to solve this problem, more on this in Section 3-3-1).

3. It is unpractical to untrust certificates in the current system. If it is detected that a CA has given out rogue certificates, the only way to ensure certificate authenticity is by blocking all certificates that have been given out by this breached CA. It is likely however that this would also result in non-malicious certificates being rejected, affecting authentic users.

A recent example demonstrating these vulnerabilities was the DigiNotar hack in 2011. During this hack, DigiNotar, a former Dutch CA that issued certificates for websites all over the world, was hacked. Adversaries issued around 500 rogue certificates for famous social network services like Gmail, Twitter, Facebook and for the domains from the secret services CIA, MI6 and Mossad. Through these false certificates, adversaries were theoretically able to execute man-in-the-middle attacks for all these domains [17]. After the detection of this breach, large browsers like Google Chrome and Mozilla Firefox decided to untrust all certificates issued by DigiNotar. Since multiple Dutch governmental websites like DigiD used DigiNotar certificates, users were not able to securely connect with these services [17].

An alternative to this hierarchical trust model is called the WoT. The WoT uses a decentralized approach. The concept was first introduced in 1992 by Philip Zimmermann as a description in the manual for PGP 2.0: "As time goes on, you will accumulate keys from other people that you may want to designate as trusted introducers. Everyone else will each choose their own trusted introducers. And everyone will gradually accumulate and distribute with their key a collection of certified signatures from other people, with the expectation

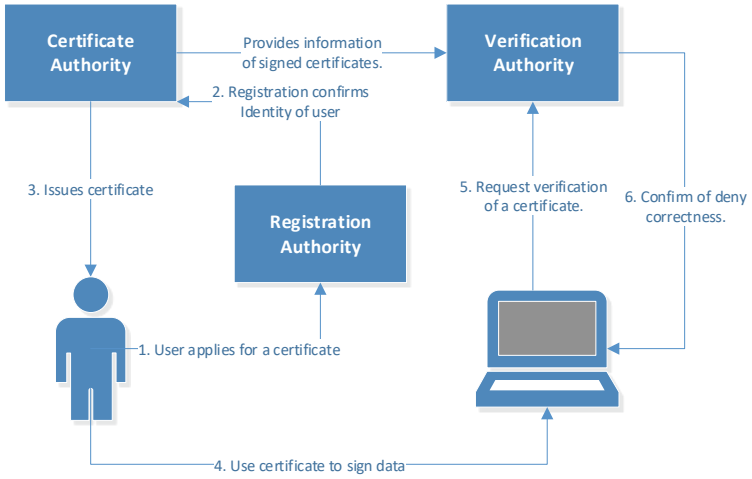

Figure 1-1: All the steps in the X.509 standard.

that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys". There are no central authorities in the WoT, instead each user of the system can sign each others public key. Because of this decentralized system, the impact on the network is limited if a signer is compromised. The WoT PKI approach is not used by most people on a day-to-day basis, in contrary to its CA counterpart. PGP, GnuPG, and other OpenPGP-compatible systems are examples of implementations of WoT in real life. The WoT however also has some disadvantages.

1. There is a barrier for nodes to participate in this scheme, it takes time and a lot of interaction to accumulate votes in this scheme.

2. It is trivial for nodes to generate new users and sign their own keys.

The decentral architecture of the WoT makes it an interesting approach when considering decentralizing services of the current cloud-based architecture of the IoT.

PKIs can help combat one of the largest challenges that arises when considering IoT security: *authentication*. Since IoT devices usually communicate without the interference of users and therefore without the use of traditional methods for authentication like passwords, biometrics and tokens, it is difficult for devices to identify themselves. This can be resolved by giving devices certificates and registering the public key for known devices. This, however, is especially for the IoT less straightforward than one might think. Certificates and public keys need to be regularly updated in order to ensure security [18]. Managing this through conventional CA-based PKI systems for millions of devices becomes extremely complex and not well scalable. By developing a distributed PKI solution on the blockchain we aim to lower the barrier for industry to adapt PKIs and hope to make the Internet a safer place.

1-2 Research objective

The rise of connected devices results in multiple security challenges. The goal of this research is to develop and analyze a distributed PKI to improve security and ease the management of the large number of identities. The underlying research question of this thesis is therefor as follows:

Research question: *"How can a scalable PKI be designed with practical key recovery, key revocation and identity verification functionality without a single point of failure to improve IoT security?"*

This research question raises the following sub-questions:

1. *"How can an IoT device identifier and key pair be verified without a central party?"*
2. *"How can practical key recovery be implemented without a central party?"*
3. *"How can key revocation be implemented?"*
4. *"How can this system be designed in a well scaling manner to facilitate PKI functionality for millions of devices?"*

1-3 Our contribution: DECKIN

In this thesis, we present DECKIN, a Decentralized Public Key Infrastructure that is built on top of the blockchain. DECKIN is a protocol that is designed to be able to cope with a large number of nodes with low computational complexity. With the introduction of our Identity Validation Algorithm we enable DECKIN to verify if nodes have access to the identity that they want to register. We also incorporated Physical Unclonable Function (PUF) technology to our system to help simplify complex key management challenges. Another characteristic of DECKIN is the usage of cryptographic accumulators, with these, nodes are able to lookup the validity of key-value pairs without having to traverse the entire blockchain.

1-4 Thesis outline

The structure of this research is composed in the following way. The first part focuses on existing solutions and the background knowledge that is used for the design of our solution. Chapter 2 discusses cryptographic preliminaries that are used in this research. Chapter 3 gives an overview of existing distributed PKI solutions and tries to map their shortcomings. We conclude with a discussion on the research methodology that we use throughout the upcomming chapters and what assumptions we make for the design of DECKIN in Chapter 4. The second part of this research presents our contribution and evaluates its performance. Chapter 5 introduces DECKIN. The protocol that we designed demonstrating techniques that are introduced to solve our research questions. In this chapter we also discuss the

theoretical performance of DECKIN, focusing on the computational and storage complexity of our system. Chapter 6 evaluates our implementation of elements of DECKIN, where we try to verify if our theoretical performance analysis is also met in practice. Finally, in Chapter 7 we provide a discussion, conclusion, and describe future work.

Chapter 2

Preliminaries

PKIs and blockchains are applications that heavily rely on cryptographic building blocks. This chapter discusses the necessary cryptographic knowledge that is used throughout this thesis. Section 2-1 focuses on the cryptographic primitives, Section 2-2 is dedicated to blockchain technology. We end our chapter with the introduction of PUF technology in Section 2-3.

2-1 Cryptographic primitives

2-1-1 Cryptographic Hash Functions

Cryptographic Hash Functions map the input of a message of arbitrary length to a short fixed length output string. The output is called the hash or the digest. Cryptographic Hash Functions are formally defined as follows:

$$H : 0,1^* \rightarrow 0,1^{|H|} \tag{2-1}$$

By definition, hash functions poses the following three properties [19]:

1. Preimage resistance– It should be hard to find a message with a given hash value.
2. Second Preimage resistance– It should be hard to find two messages with the same hash value.
3. Collision resistance– Given one message it should be hard to find another message with the same hash values.

Practical applications of hash functions are authentication, digital signatures and message integrity checks. The most widely deployed hash functions are MD-5, RIPEMD-160, SHA-1 and SHA-2 [20]. Our own protocol DECKIN uses cryptographic hash functions extensively when verifying if a user posesses the identity that he wants to register. Similarly, the Bitcoin protocol uses SHA-2 for its proof of work algorithm.

2-1-2 Asymmetric encryption

In public key cryptography, also known as asymmetric cryptography, *public* and *private* keys are used to respectively encrypt and decrypt data. Each user i generates its own keypair pk_i and sk_i. If Alice (A) now wants to send a message m to Bob (B), Alice should first retrieve Bob's *public* key pk_B. Alice now encrypts her message with the encryption function E with as input pk_B, resulting in the ciphertext c.

$$E_{pk_B}(m) = c \tag{2-2}$$

The ciphertext can now safely be sent to Bob, without having to fear that people are able to read its contents. This message can only be decrypted by Bob (assuming nobody stole his *secret* key sk_B) with the decryption function D as follows:

$$D_{sk_B}(c) = m \tag{2-3}$$

One of the disadvantages of asymmetric encryption is that the encryption and decryption process is relatively slow compared to symmetric encryption. For asymmetric encryption to be secure, users need to be certain that a public key is authentic and that it belongs to the person or entity who claims he is, and that the key has not been tampered with or replaced by a malicious third party.

2-1-3 Symmetric encryption

Symmetric encryption uses a single key for encryption and decryption of information. If Alice and Bob want to send data to each other with symmetric encryption, they first need to agree on a shared key k. Alice now encrypts her message m with the encryption function E to obtain the ciphertext c.

$$E_k(m) = c \tag{2-4}$$

The ciphertext can now be sent to Bob, that is able to decrypt it with the same key k and the decryption function D.

$$D_k(c) = m \tag{2-5}$$

Symmetric encryption is used a lot in modern day computing. Advanced Encryption Standard (AES) is an example of a widely adopted symmetric encryption scheme and is chosen by NIST as the standard for symmetric encryption algorithms [21]. Our system (DECKIN) focuses on the distribution of keys, and does not make a distinction between asymmetric or symmetric keys.

2-1-4 Merkle Trees

Merkle trees, also known as binary hash trees or Merkle hash trees, were introduced in 1987 in the context of digital signatures [22]. A (static) merkle tree is a tree where every leaf node is labeled with that hash of a data block and every non-leaf node is labeled with the cryptographic hash of the labels of its child nodes. Thus, the hash of each parent node comprises a compact representation of its two child nodes. Given a hash for the root of the Merkle tree, what is known as the *Merkle root hash*, it can be proved that a specific leaf node

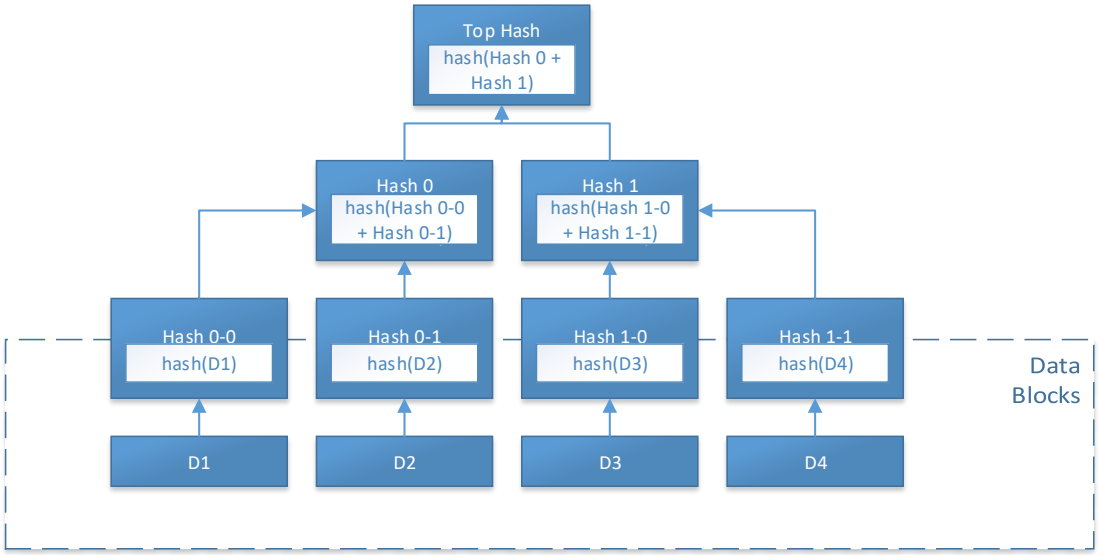

Figure 2-1: A Merkle Tree used to make a single hash value for a large number of Data Blocks.

is present in the tree by verifying only $\mathcal{O}(\log n)$ hashes [23]. Figure 2-1 shows an example of a static Merkle tree.

Each parent node N_i containing data blocks L_j and L_{j+1} can be calculated as follows:

$$N_i = H(L_j || L_{j+1}). \tag{2-6}$$

Verifying whether or not certain data blocks belong to a Merkle Tree can be done efficiently. Another advantage of Merkle Hashes is that it allows you to make a signature of an object containing multiple objects that has the size of a single string. Because of these advantages, Merkle hashes are used in blockchains to register containing all transactions that belong to the block. To obtain Merkle trees which can efficiently be updated, a tree can be constructed that not only associates the leaves for data blocks, but also adds the hash of a data block to an interior node. For each interior node, the hash is then computed by concatenating three hashes (the two children and the extra data point). Now, new nodes can easily be built on top of the existing tree. Our system relies heavily on merkle trees, the cryptographic accumulator (explained in Section 2-1-6) that DECKIN uses, is based on merkle trees.

2-1-5 Digital Signature Schemes

Digital Signature schemes are schemes that help demonstrating the authenticity of data. A digital signature scheme consists out of 3 operations, key generation, signing and verifying.

$$keygen(1^k) \rightarrow (sk, pk) \tag{2-7}$$

Key generation generates a secret key together with a corresponding public key given a security parameter k.

$$sign_{sk}(m) \rightarrow \sigma \tag{2-8}$$

The signing produces a digital signature σ on the message m using the users secret key sk.

$$ver(pk, \sigma, m) \rightarrow b \in 0, 1 \tag{2-9}$$

Verification determines whether or not σ is a valid signature on m under the secret key corresponding to the public key pk. Digital signatures should be *existentially unforgeable* [24]. This means that, without knowledge of the secret key sk corresponding to the public key pk, no probabilistic polynomial-time adversary A should be able to produce a signature-message pair (σ, μ) such that ver (pk, σ, μ)=1 with probability non-negligible in the security parameter k [25]. Our protocol uses Elliptic Curve Digital Signature Algorithm (ECDSA), this is a signature scheme that uses Elliptic Curve Cryptography (ECC). ECDSA is also used by Bitcoin and Etheureum. One of the large benefits of the usage of ECC over the conventional public key cryptography systems (like RSA), is that the key-size is significantly lower for ECC-based systems achieving a similar level of security.

2-1-6 Cryptographic accumulators

Cryptographic accumulators were introduced in 1994 as a decentralized alternative to digital signatures [26]. Cryptographic accumulators play an important role in our protocol. Cryptographic accumulators are a compact representation of a set of elements that support membership queries. In 2002 a more challenging notion of dynamic accumulators was introduced [27], allowing to dynamically add and delete elements into the original set. Upon addition of an element x into an accumulator A, a witness w is generated. This witness can be used to prove that a value x is added to the accumulator A. A basic cryptographic accumulator consists of the following four polynomial-time algorithms:

$\mathbf{AccGen}(\mathbf{1^k}) \rightarrow \mathbf{a}$ This function initializes the empty value of an accumulator.

$\mathbf{AccAdd}(\mathbf{a}, \mathbf{y}) \rightarrow (\mathbf{a'}, \mathbf{w})$ This function, that takes the current state of the accumulator a and a to-be-added value y, returns a new accumulator value a' and a witness w.

$\mathbf{AccWitAdd}(\mathbf{w}, \mathbf{y}) \rightarrow \mathbf{w'}$ This function takes a witness value w and a newly added value y, and returns an updated witness value.

$\mathbf{AccVer}(\mathbf{a}, \mathbf{y}, \mathbf{w}) \rightarrow \mathbf{0}, \mathbf{1}$ This function take an accumulator a, a value y and a witness w and verifies if y has been added to the accumulator.

Relevant properties for cryptographic accumulators are *compactness*, *universality*, *strength* and *public checkability*. An accumulator is called *compact* if its size is constant. It is called *universal* if the accumulator also supports non-membership proofs in addition to membership proofs [28]. *Strong* accumulators are accumulators which do not assume that the accumulator manager is trusted. This means that accumulators cannot use trapdoor information in the creation or maintenance of the accumulator [29]. *Publicly checkable* indicates if each operation on an accumulator can be publicly verified.

Table 2-1: Comparison of different accumulator types

Accumulator	Compact	Universal	Strong	Publicly Checkable
RSA	yes	yes	no	yes
Bilinear Map	yes	no	no	yes
Merkle	no	yes	yes	yes

In Table 2-1 a comparison is made between the RSA accumulator construction [26], the Bilinear Map construction [30] and the Merkle Tree construction [29]. In this paragraph, we are going to discuss the Merkle tree accumulator more detailed because it is the only accumulator that possesses the strength property. Although this accumulator is not compact, a property that would be desirable considering the large number of devices, the accumulator size is $O(\log(n^2))$ where n is the number of elements that are added to the accumulator. Assuming the accumulator would have to store 1 billion registrations, the logarithm would be 36. The size of a Merkle tree accumulator is the digest of a hash function, assuming SHA-512 this would be 512 bits. This would make the size of our accumulator with 1 billion nodes the same size as 16384 or roughly 16KB. This size is so small that we do not consider it a problem that the Merkle tree accumulator is not compact. In the Merkle Tree construction, a set element is a leaf of the tree. Its witness is the sequence of the element's ancestors' siblings. The Merkle root is the accumulator value. Nodes can simply verify if an element is part of the set by verifying if the Merkle root equals the value obtained when calculating the Merkle root value with the hash of your element and the values obtained from your witness. A downside of this accumulator construction is that, if a value is added to the accumulator, all witnesses need to be updated. The accumulator and witness are out of sync. This means that a witness is not updated with the latest additions of an accumulator, resulting in a witness being useless since the traditional Merkle tree accumulator requires accumulator and witnesses to be in sync. In [31], a strong asynchronous accumulator is introduced based on the Merkle tree accumulator, that requires a *low update frequency* and *old-accumulator compatibility*. This means that a witness only needs to be updated a small number of times, and that a witness can be used with outdated accumulator values. We are going to discuss the workings of this specific accumulator in the following paragraphs.

Construction Let n be the number of elements in our accumulator, and h a collision resistant hash function. Our accumulator maintains a list of $D = \lceil \log(n+1) \rceil$ Merkle tree roots $r_{D-1}, ..r_0$ (as opposed to just one Merkle tree root). The leaves of these Merkle trees are the accumulated elements. r_d is defined as the root of a complete Merkle tree with 2^d leaves. Initializing an empty array is sufficient to construct our accumulator.

Addition Elements are added by merging Merkle trees and by creating deeper ones. Assume the nth element x is being added to an accumulator consisting of $[r_{D-1}, ..., r_0]$. If $r_0 = \emptyset$, we set $r_0 = h(x)$. If, however, $r_0 \neq \emptyset$, a depth-one Merkle tree root is created $z = h(r_0, h(x))$ and set $r_0 = \emptyset$. We now try to place z at r_1. If r_1 also is not empty we create a depth-two Merkle tree root. We continue with this until a root is found where z can be stored. Algorithm 1 shows the formal definition of this operation. As you can see, the output is a_{t+1}, what is the

new accumulator value, and $w_{t_1}^x$, what is the witness value. Image 2-2 show examples of an accumulator before and after adding the value x_{t+1}.

Membership verification A witness for element x, w_x, is the authenticating path for x in the Merkle tree that contains x. In the example of x_{t-2} in Image 2-2, the witness for x_{t-2} would be $w^{x_{t-2}} = (h(x_{t-1}), right)$. Membership verification for any element x is done by using the witness w^x and the element x in question to recompute the Merkle tree root and check that it indeed matches the accumulator root r_d, where d is the length of w^x.

Algorithm 1 The addition operation of the extended Merkle tree accumulator, as defined in [31]

$a_{t+1} = a_t$ ▷ New accumulators start out as a copy of the old one.
$w_{t+1} = []$ ▷ The witness starts out as an empty list.
$d = 0$ ▷ The depth of the witness starts out as 0.
$z = h(x)$
while $a_{t+1}[d] \neq \emptyset$ **do**
    if the length of $a_{t+1} < d + 2$ **then**
        append $\emptyset$ to a_{t+1}
    end if
    $z = h(a_{t+1}[d], left)$ to w_{t+1}^x
    $a_{t+1}[d] = \emptyset$
    $d = d + 1$
end while
$a_{t+1}[d] = z$
return a_{t+1}, w_{t+1}^x

2-1-7 Distributed Hash Tables

A Distributed Hash Table (DHT) is a distributed system that provides a service comparable to a hash table, spread over a distributed network. Responsibility for maintaining the mapping from keys to values is divided among all the nodes in the distributed network. DHTs are often used in combination with blockchain solutions because of the lack of a central party. By deploying DHTs, nodes are able to store data without the need of a central party.

2-2 Blockchain technology

A blockchain is an append-only public ledger that is replicated among all the connected nodes in a peer-to-peer network. Blockchain technology was originally designed to store financial transactions for the Bitcoin cryptocurrency. Presented in 2009 by a pseudonym named Satoshi Nakamoto [11], it was the first protocol that solved the *double spending problem*. The double spending problem is the problem of preventing users in your system to spend a single digital token more than once. Where conventional systems always have a trusted central authority that keeps track of what has been spent, Blockchain needs to implement a system where users

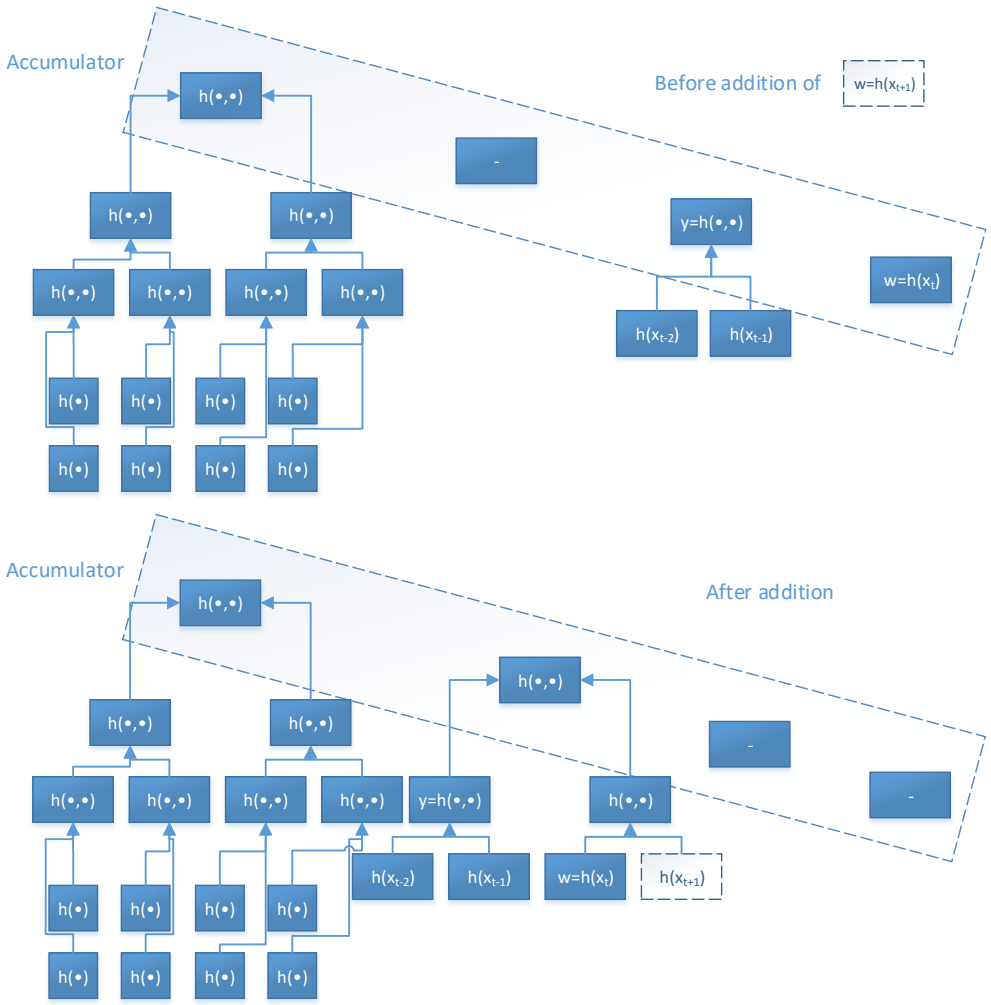

Figure 2-2: An example of an addition of $h(x_{t+1})$ to our accumulator, the upper diagram shows the begin state, the lower figure shows the end state after addition succeeded.

agree on the validity of transactions where the present state of the system is determined based on cryptographically verifiable schemes, called consensus models.

2-2-1 Consensus models

One of the revolutionary novelties that was introduced with the Bitcoin cryptocurrency was its consensus model named Proof of Work (PoW), also known as *Nakamoto consensus*. Consensus models are the distributed protocols that decide which node is allowed to add a new block to the chain. The PoW algorithm has been reused by many succeeding blockchains including Ethereum and Lightcoin [32]. In PoW, nodes called miners try to solve a complex computational puzzle. The miner that finishes this puzzle first is the node that is allowed to add a new block to the blockchain. The puzzle is constructed in such a way that it is difficult to solve, but easy to verify. This is called a *trapdoor function* in cryptography. The

Bitcoin proof of work algorithm consists out of finding the SHA-2 hash value that meets the requirement shown in Equation 2-10.

$$\text{SHA2(SHA-2(block header))} < 2^{(256-k)}, \tag{2-10}$$

Where k is a difficulty factor that is determined by all nodes in the network. The Bitcoin algorithm is defined in such a way that on average a block is mined every 10 minutes. Miners are given an incentive to mine because they are awarded with transaction costs and a block reward, which is paid to the address found in the first transaction in the block, called the *coinbase transaction*.

A problem with the PoW algorithm is its environmental impact. A study released in 2018 showed that the energy usage of the Bitcoin network was currently about 2.55 GW of electricity per year and is moving towards consuming 7.67 GW in the future [33]. For reference, Ireland consumes 3.1 GW and Austria 8.2 GW of energy per year. The work predicts that that the Bitcoin network will consume 0.5% of the worlds energy usage at the end of 2018 [33]. One of the consensus models that tries to solve this is the Proof of Stake (PoS) model. In this model, a block leader is pseudo randomly selected where the amount of stake of network users affects the outcome. The more stake a user has, the higher the possibility that he is chosen. The idea is that a node who has a large number of tokens would have an incentive to be benevolent, since malicious behavior would undermine his own wealth. PoS-based blockchain solutions are considered better scalable than PoW-based solutions since the required computational power to solve challenges does not increase with the number of users in the network [34]. There are far too many alternative consensus algorithms to discuss, the PoW and PoS algorithms however are extensively researched because of their applications in Bitcoin and Ethereum [32].

2-2-2 Transactions

Image 2-3 shows an example of a Bitcoin transaction. A transaction contains a header and a list of transactions. Each block header contains a Merkle root, making sure that the connected transactions are recorded, a timestamp registering when the block was initialized, a reference to the previous block in the form of a block header hash, and a nonce and difficulty integer that are used for the consensus model. In the case of Bitcoin, miners try to adjust values like the nonce and the timestamp up until the point where their block hash is accepted as a valid new block hash.

2-2-3 Attacks

Although our research does not focus on the implementation of a concrete blockchain protocol but on a protocol that is built on top of a blockchain, we do introduce attacks that are known since we consider this relevant knowledge when developing applications on this new technology. We argue that knowing what type of attacks exist helps in designing a safer protocol.

51%-attack The Blockchain mechanism is designed with the assumption that a majority of the nodes in the network are honest. If an attacker would have a majority of the computing

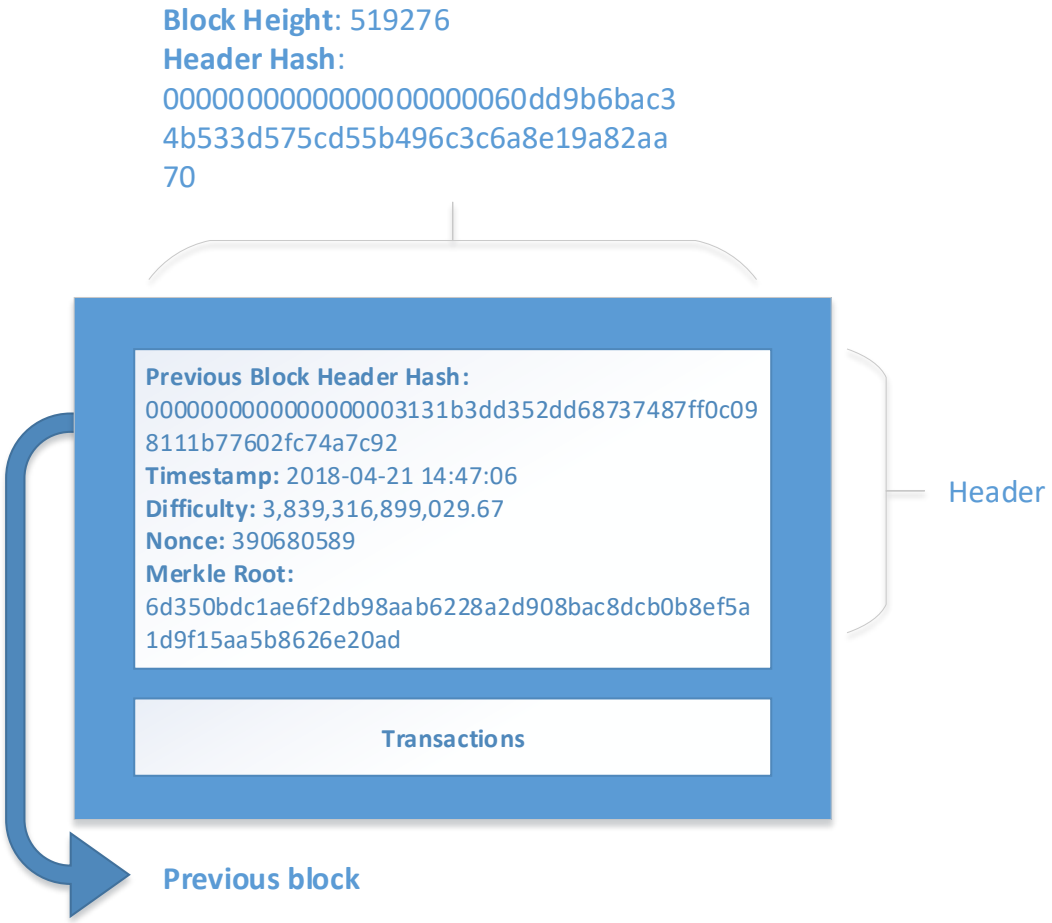

Figure 2-3: Example of all information stored in Bitcoin blocks.

power available in the system, the network is vulnerable for the so called "51% attack". Since the Bitcoin network is designed as a fully decentralized network, the likelihood of this happening is low as long as a large amount of nodes participates. It has however been shown that market-based centralization of mining power by a few large mining pools increases the risk of a 51% attack[35]. Other researchers [36] argue that, in order to truly be able to consider a protocol safe, adversaries should not be able to acquire 1/3 of the total computing power. Although there have not yet been known confirmed 51% attacks on the Bitcoin network, the Bitcoin Gold protocol, a fork of the original Bitcoin protocol, has suffered a 51% on May 2018 [37].

Authentication attacks The private key is the major authentication element in the Bitcoin network. The most well-known issue with this was the hack of the Bitcoin exchange Mt. Gox in 2014 [38]. Hackers stole private keys that were stored online, resulting in a loss of 744408 bitcoins [38] stolen from users that stored their key at the Mt. Gox exchange. This attack

has motivated some studies in strengthening authentication in Bitcoin, since some researchers argue that the usage of certificates is not sufficient [39].

2-2-4 Blockchain scalability

The underlying technology of Bitcoin, blockchain, has been relatively new. In a few years Bitcoin has changed from a small network run by a few enthusiasts, to a network that counts thousands of users and nodes. The underlying technology has not changed a lot and has also exposed problems [40]. One of the largest challenges that Bitcoin needs to combat is scalability: currently, the number of transactions the Bitcoin protocol can handle per second is 7. There are a number of teams working on a variety of solutions to solve this problem, in this section we will be taking a closer look at some of the promising scaling solutions for Ethereum. Ethereum has an active developer community and is considered as one of the largest cryptocurrencies.

Sharding In traditional blockchain systems, each node needs to process (download, compute, store, read) every transaction in a blockchain, sharding changes this. Sharding is an on-chain scalability solution. The goal of sharding is to partition all network computational resources into shards, so that a node only needs to process the transactions that belong to a specific shard. Each shards is handled separately by nodes that can decide what shard to use [41].

Raiden In contrary to Sharding, Raiden is a so-called "off-chain" scalability solution. Simply explained, Raiden enables collections of nodes to establish payment channels between each other to facilitate transactions, without them directly transacting with the Ethereum blockchain and clogging up capacity [42]. To prevent double spending, each node must hold a balance greater than or equal to the amount in these transactions that is publicly stored on the blockchain.

Plasma Plasma is another off-chain solution that was announced by the creator of Ethereum, Vitalik Buterin, in August 2017 [43]. Plasma uses a series of smart contracts to create hierarchical trees of sidechains. These sidechains can be seen as child blockchains that live within a parent blockchain, periodically relaying information back to the root chain (the original Ethereum blockchain). Just as with Raiden, Plasma computations are done off chain and therefore transactions are cheaper and faster when comparing them with transactions that are executed on chain.

2-3 Physical unclonable functions (PUFs)

PUFs are a hardware cryptographic primitive, first introduced in 2001 [44]. PUF technology uses specific hardware features that is sometimes called a silicon fingerprint, analogs to the unique and unclonable human fingerprint. The idea behind PUF technology is that during the production of electronic products, slight differences are always present. If these differences are unique for a device, they can be used as input for a key generator. This results in a

unique key that can only be regenerated by extracting information from a physical device, making it safe compared to keys that are based on programmatically stored keys. These can be extracted through connected devices through for example the Internet. There are many different types of characteristics that can be used for PUF generation, this can vary from the angle that light breaks when going through a piece of glass to the amount of resistance that is measurable on a PCB. Over the years, a broad taxonomy for PUFs has been developed [45]:

1. *Non-electronic PUFs* – These are PUF applications that use non-electronic properties as for example the random fibre structure of the paper or of the scattering characteristic in an optical medium.
2. *Electronic PUFs* – These are PUF applications that make use of electronic properties. Capacitance is an example.
    (a) *Silicon PUFs* – A subclass of the Electronic PUFs. These are PUF applications that uses the electronic properties of integrated circuits.

There are two different aspects that need to be considered in PUF technology, a physical part and an operational part. The physical part refers to the physical attributes that are being used to extract data. The operational part makes sure that a sufficient set of inputs (from now on called *challenges*) is available that results to a sufficient number of outputs (from now on called *responses*). A combination of a challenge and a response is called a challenge-response-pair (CRP). Distinctions are made between PUFs based on the number of CRP they have [46]: *strong* PUFs support a large number of CRPs, *weak* PUFs support a limited number of CRPs. Because of this, *strong* PUFs can be used for authentication, *weak* PUFs not because of there low entropy value. PUFs can also be categorized based on their physical design. The two major categories are intrinsic and extrinsic. Intrinsic PUFs use an attribute that is available naturally during the manufacturing process. Extrinsic PUFs need extra hardware added to the PUF component in order to make it usable. An example would be an optical PUF where you need a measurement device to measure the angle of light bulbs.

In order to be qualified as a PUF, a device should at least fulfill the following characteristics [47]:

1. Reliable – A response to the same challenge should be able to be reproduced over time and over a various of conditions.
2. Unpredictable – A response to a challenge on a PUF device should be unrelated to a response to another challenge from the same device or the same challenge from different devices.
3. Unclonable – Challenge-response pairs mapping of a device should be unique and cannot be duplicated.
4. Physically Unbreakable – Any physical attempts to maliciously modify the device will result in malfunction or permanent damage.

Image 2-4 is a schematic example of the uniqueness property.

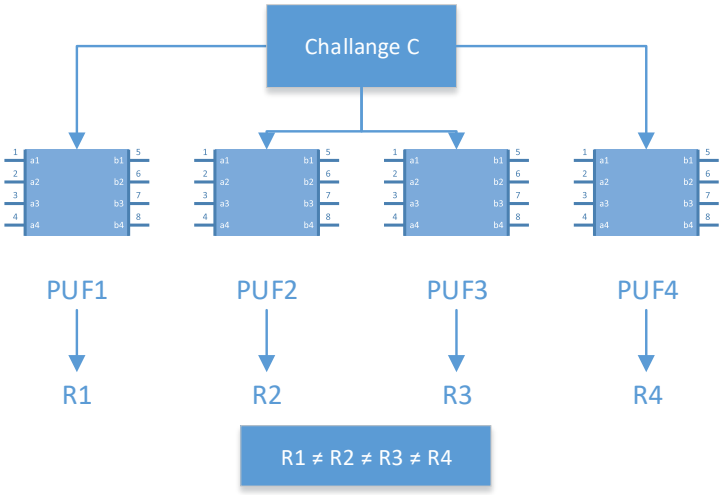

Figure 2-4: An example of the uniqueness property. Each device generates a unique response for a each challenge C.

2-3-1 Static Random-Access Memory PUFs

Static Random-Access Memory (SRAM) PUFs are a specific type of PUF category. This PUF technology is commonly used in industry since it relies on standard digital integrated circuit components, components that most electronic devices contain. This is also why we choose for this technique for our protocol. Image 2-5 shows a basic six-transistor CMOS SRAM cell. Transistors T_1 and T_2, and transistors T_3 and T_4 both form inverters. Transistors T_5 and T_6 are used to access the transistor values. These inverters are designed to be well-balanced in a symmetrical way. Due to small and random process variations during the manufacturing of these cells, a cell acquires a preferred state when turned on. This can be either 0 or 1, referred to as the electronic fingerprint of this cell. When this electronic fingerprint is collected from enough memory cells, a unique identifier can be extracted for each device containing SRAM cells.

Reliability

In order for SRAM PUFs to be usable, it is important that the start-up values of circuits are uniquely determined, unpredictable and similar each time the circuit is turned on [48]. Due to variances in for example voltage that is used on the circuit, temperatures in which circuits are used and simply the aging of circuits, this is not always the case. It often happens that when a PUF key is extracted for a second time, some noise is added, making the keys not identical. In order to improve reliability, the usage of error-correction codes has been suggested [49]. An error-correction code, or helper data, is a codeword that is generated during the initial PUF key extraction. It needs to be stored in memory that is accessible by the PUF algorithm, but since it is designed to not leak any information about the key, this can be stored off-chip. When a PUF key is extracted for a second time, the error-correction

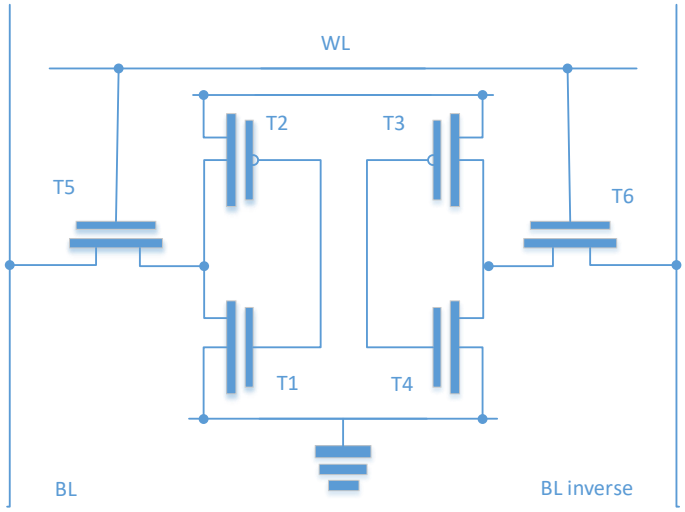

Figure 2-5: A six-transistor CMOS SRAM cell. T_1 and T_2 form a CMOS inverter, and T_3 and T_4 form a CMOS inverter. T_5 and T_6 are used the access the cell.

code is used to remove noise and therefore improve reliability. Image 2-6 illustrates both the enrollment and the reconstruction phases.

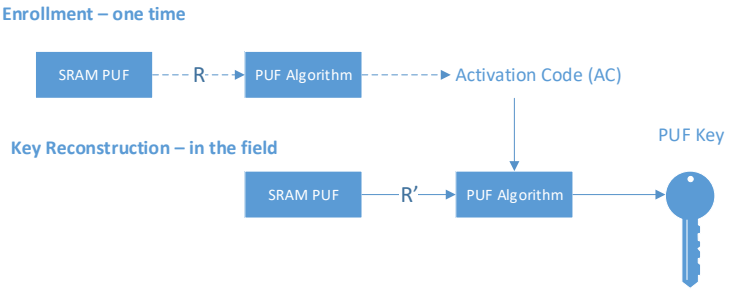

Figure 2-6: The enrollment and reconstruction phase for the generation of PUF keys.

Chapter 3

State-of-the-art

The recent rise of blockchain technology has resulted in an increase of research on distributed identity infrastructures [50]. In the same time, recent cybersecurity incidents have increased research in looking how to improve PKIs. PKIs are currently widely used to setup SSL/TLS connections for Internet websites. To our best knowledge, there does not exist a specification for a completely decentralized PKI with functional validation, key recovery and key revocation infrastructure that is optimized for devices with little storage space and low computational power. Multiple solutions have been proposed, however, that implement one or more of these features. This chapter addresses existing work on distributed PKI solutions, trying to map all relevant research that has been done previously. We discuss the latest progress and research done on existing blockchain-based identity solutions in Section 3-1. We then try to compare these protocols and show how they relate to each other in Section 3-2. We then look at alternative, non-blockchain based initiatives that are being developed to address the shortcomings of existing PKI solutions in Section 3-3. We end with discussing the different considerations that need to be done when deciding what blockchain framework should be used when building a distributed PKI solution in Section 3-4.

3-1 Existing blockchain-based solutions

Traditionally, PKI and Domain Name System (DNS) are two separated services in the Internet infrastructure. Where DNS is used to map domain names to IP addresses, a PKI is used to publish keys mapped to identities. In centralized setups a PKI plays an important and unique role in verification and validation, ensuring that a key is issued to legitimate users. In research trying to develop protocols that are distributed, the role of PKI and DNS infrastructures are often combined. This is why the following subsections often present protocols that are developed for a variety of applications. Since the applications of all the discussed protocols is pretty varying we focus on the following six attributes to make the protocols comparable:

1. Validation– Under validation we consider verifying if a user or node has access to the domain or identity that it is trying to register.

Table 3-1: New transactions used in the Namecoin blockchain. Note that after a `new` command a domain is still not owned. After `firstupdate` you own the domain during the next 36000 blocks (approximately six months), and `update` gives your another 36000 blocks of ownership.

Command	Registration fee	Transaction fee	Summary
`new`	0.01 NMC	0.005 NMC	Pre-order a domain name.
`firstupdate`	0.00 NMC	0.005 NMC	Finalize registration. The name becomes public.
`update`	0.00 NMC	0.005 NMC	Renew, update, or transfer a name.

2. Authentication– Under Authentication we consider identifying the user or node that is being registered.

3. Scalability– Under scalability we consider if the presented work can be successfully executed by nodes with low storage and low computing power on extremely large (more than a billion nodes) networks.

4. Key recovery– We verify if users are able to recover keys in the presented protocol.

5. Key revocation– We verify if users are able to revoke keys, for example in case of an update of a keypair or if keys are stolen.

6. Privacy preserving– We look if privacy-preserving techniques are implemented for the presented schemes.

A comparison of all the presented work is shown in Table 3-2 and discussed in Section 3-2.

3-1-1 Namecoin

Namecoin [51] was one of the first designed alterations of Bitcoin after its introduction. It is a cryptocurrency that is designed to act as a decentralized DNS for *.bit* addresses. The Bitcoin cryptocurrency is extended with three new transaction types: `name_new`, `name_firstupdate` and `name_update`. Table 3-1 gives an overview of these new transaction types. Registering a domain name costs 0.02 units of Namecoin, for the other operations, only a transaction fee of 0.005NMC is applied. As of 13 January 2018, this would be roughly 0,05 dollar*. Although Namecoin is initially designed as a decentralized DNS registry, it includes the functionality for registering certificates that can be used to authenticate the associated identity. Namecoin was one of the first blockchain based PKI solutions presented. It does not present all to many features, but is used as a basis by many following protocols. Verification, authentication, key-recovery and key-revocation mechanisms are not implemented in Namecoin. Namecoin would also not be able to handle more than a billion nodes with low storage and computing power, since each participating node needs to traverse the entire blockchain to lookup records.

*http://coinmarketcap.nl

3-1-2 Certcoin

In 2014, Certcoin [25] was presented. Certcoin is a decentralized and publicly available authentication scheme that is built on top of Namecoin. Registration, update and revocation of certificates is supported. In this scheme, each user needs to generate an *online* and an *offline* keypair (containing a public and a secret key). The user then posts its public keys on the blockchain, combined with signatures proving he is in the possession of the secret key (verifiable by each node). The *online* secret key is used by users in the system to authenticate messages to and from the user, the *offline* secret key needs to be stored securely offline. The *offline* secret key functions as a backup and is used to (1) revoke old keys and (2) sign new keys in case of a key compromise.

The Certcoin protocol is one of the few distributed PKI solutions that works on supporting extremely large networks (in this particular case it even supports more than a billion nodes) and is therefore considered scalable according to our presented definition in Section 3-1. By introducing cryptographic accumulators (see Section 2-1-6), this scheme enables users to verify identity and key pairs efficiently without traversing through the entire blockchain. For each action performed (key registration, key update or key revocation), the accumulator a is updated. The new accumulator value a' and the newly generated witness w are then added to the newly mined block that will store this new transaction. Nodes can now easily verify if an identifier-key tuple is valid on the blockchain by looking at the latest accumulator value a on the blockchain and verifying it in combination with a witness value w. Note that this witness value needs to be provided to the verifier. This can, for example, be the entity that is storing its key on the PKI. With this addition, nodes do not have to traverse the entire blockchain to see if a key value pair is valid. This ensures that also devices with low storage space are able to lookup key value pairs. The Certcoin paper also introduces the usage of a DHT (see Section 2-1-7). In this paper, this is used to give users the possibility to lookup what key-value pair belongs to an identity, without the necessity to traverse the entire blockchain. The authors propose to use the Kademlia DHT [52]. All nodes in the system are incentivized to participate in this DHT scheme since public keys from node that are not participating are removed from the blockchain. The DHT contains (pk, w) entries (the public keys combined with the accumulator witness w), queryable by (id).

The revocation process in Certcoin differs depending on which key is accessed or stolen. If the *online* key is lost, a user can simply execute a `revoke` or `update` transaction signed with its *offline* secret key. If the *online* key is stolen, a user can also execute a `revoke` or `update` transaction signed with his *offline* or *online* secret key. The risk, however, exists that an adversary has already revoked or updated the stolen key before the user notices that its key is stolen. This is why any statement signed by both secret keys (*online* and *offline*) outweighs any statement signed by only one. If both keys however are lost, no revocation process is possible in Certcoin. Certcoin also implements a key recovery technique that is similar to the technique used by the Bitcoin wallet management platform Armory [53]: Certcoin requires users to secretly share their secret key with at least three trusted "friends", with a threshold of at least two for reconstruction. A user can now recover his secret key through an interactive cryptographic protocol with his trusted peers.

Although Certcoin does not verify and authenticate newly registered entries, it is one of the few works that actively tries to modify its scheme in such a manner that it is able to cope

with more than a billion nodes with limited resources. Key-recovery is implemented in the scheme through a n-threshold secret sharing scheme.

3-1-3 Privacy-Aware Blockchain-Based PKI

PB-PKI [14] builds upon the Certcoin paper with a strong emphasis on privacy. Just as in the Certcoin paper, a user generates an *online* and an *offline* keypair. As opposed to Certcoin, however, users in PB-PKI only link their first *online* key to their identity, the *offline* keypair is only indirectly linked to this identity. Updates, containing new public keys, do not contain an identity. The new online public key at each update is computed as a function of the previous online public key and the offline secret key (see Figure 3-1). Since this offline secret key is only known by the user, a hidden link is created between updates and the initially posted identity. Therefore, once an identity *id* is established, key updates are anonymous. A user cannot lookup what key belongs to an identity (assuming this key has had its initial update). It can only be verified that an identity is registered to the blockchain, and if a key belongs to one of the participating nodes in the network. PB-PKI implements *user-controlled disclosure*.

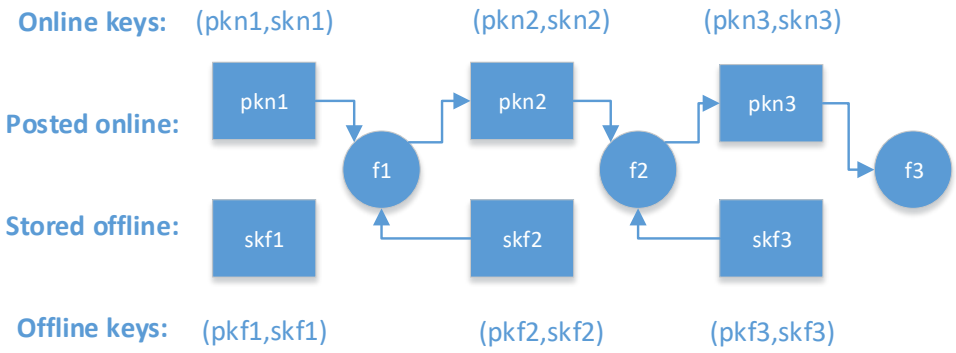

Figure 3-1: The hidden linkage of skf_n to pkn_n makes PB-PKI anonymous.

This means that the owner of a registered identity decides if the mapping from a public key to his identity is shown. This technique is therefore not usable in the use-case of a PKI used for the authentication of IoT devices. In PB-PKI users need to register all the old *offline* keypairs that they have generated in the past since they need to show all these identities to prove the linkage of their identity to a public key. Assuming that keys are renewed on a frequent basis, this means that identities will have to store a large number of keys as time passes by. The authors of PB-PKI reason that links between identities and their public keys should be available when required by, for example, law-enforcement entities. To implement this functionality they decided that, for each key update in their scheme, each entity should share his offline secret key between a majority of the network using a secret-sharing scheme. Although this implements the ability of tracing misbehaving entities, if required through collusion of a majority of the network, it adds an extra layer of complexity and makes scaling towards a large (more than a billion nodes) network not feasible without modifications to the proposed scheme. Revocation is implemented in multiple ways. Besides the two (*online* and *offline*) keypairs generated at initial key generation, each user also generates a *master offline*

key pair during the initial registration. This master key pair is stored offline and is only used for revocation. If keys are lost, operations signed with the master key pair can always overrule operations signed with the other keypairs. A user can lose his *online* keypair, his *offline* keypair and his *offline* master keypair. The paper describes each scenario extensively, but it should be noted that in the worst case (where all keys are stolen) there is no mechanism for a user to recover the ownership of his public key. As mentioned, PB-PKI does not scale well when considering a network of more than a billion nodes with low computational power. Each node needs to keep track of each key change and store his previous offline keys to prove linkage of his identity to a public key. Each update, however, also needs to be registered by a majority of the network in order to lookup a link between identities and their public keys if needed by law enforcing entities. PB-PKI also does not demonstrate verification and authentication techniques. It is one of the few protocols however that takes privacy into account.

3-1-4 Blockstack

Introduced in 2016, Blockstack [54] is a more recent design of a system presenting a distributed PKI built on top of a blockchain. In Blockstack, a portable (also known as blockchain agnostic) architecture is implemented. This means it is designed to be able to read and write data to any blockchain and the logic for operating the domain name system is decoupled from the logic of the underlying blockchain [55]. The Blockstack architecture consists out of 4 layers. A graphical representation of these four layers can be found in Image 3-2. Layer 1 and layer 2 are called the *control plane*, since its task is to control the integrity of all claims made in the Blockstack network. The first layer of the Blockstack architecture, layer 1, consists out of a physical blockchain. This layer serves two purposes: it provides the storage medium for operations and it provides consensus on the order in which the operations were written [54]. The second layer in Blockstack is called the *virtualchain*. This layer defines new operations without requiring changes to the underlying blockchain. Blockchain nodes do see the raw transactions, but the logic to process Blockstack operations only exists at the virtualchain level [55]. The third and fourth layer in Blockstack are respectively called the *routing layer* and the *storage layer*, these layers are called the *data plane*, since the task of these layers is to map logic from the *control plane* to data. The *routing layer* routs requests. This layer uses *zone files* for storing routing information, these *zone files* are identical to DNS zone files in their format. Blockstack uses a DHT-based network to store *zone files*. The *storage layer* hosts the actual data values of name-value pairs. By storing data values outside of the blockchain, you are able to store arbitrarily sized objects. A variety of storage backends can be used. The data hash, or a user signature, is stored in the control plane. Since users can easily check the data integrity by comparing a hash of the data with the stored hash value in the control plane, users are able to trust the data even though it might be stored on an insecure location. This data plane can use distributed storage protocols as well as non-distributed storage protocols (such as Amazon S3 and Microsoft Azure).

It should be noted that Blockstack is not just a PKI solution, but an infrastructure for decentralized applications. It aims to address the centralization at the application-layer of the Internet. It has released an open-source browser that is able to communicate with the existing Blockstack network, allowing you to fetch identities and store information in its network. We focus on the DNS and PKI functionality, since that is what this chapter tries

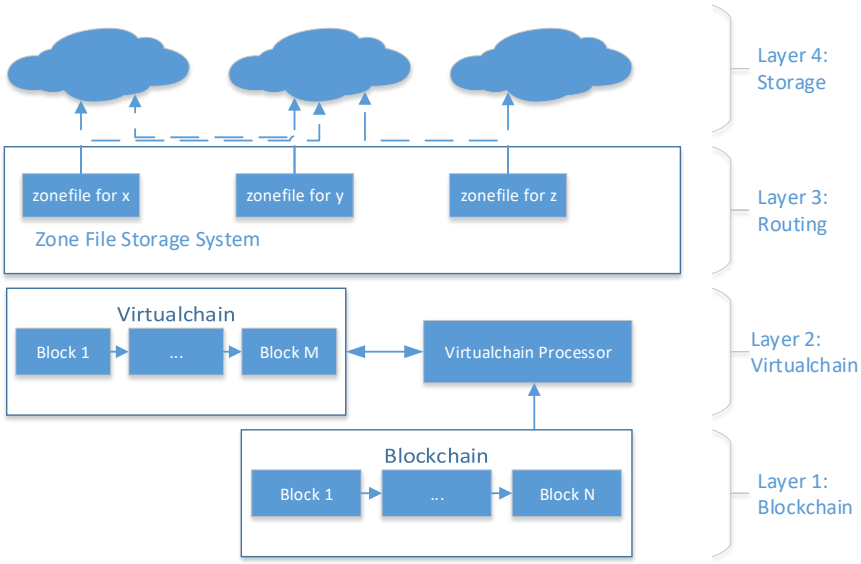

Figure 3-2: Overview of Blockstack's architecture. Blockchain records give mappings. Hashes are looked up in routing layer to discover routes to data. Data, signed by name owner's public-key, is stored in cloud storage.

to lay out. For the Blockstack protocol this functionality is implemented in the Blockchain Name System (BNS). The BNS is comparable with previously discussed schemes. Users are able to register, update, transfer and revoke names. The BNS registers domain names with IP addresses and public keys. The difference with previously discussed systems is that names are organized into *namespaces*. Namespaces are the functional equivalent of top-level domains in DNS, defining the cost and the renewal rates of names. These namespaces are registered on a *root blockchain*, layer 1 of the Blockstack protocol, and links to a zone file being stored on an external source. This external source for zone files is untrusted. However, since the hash of this zone file is present in the blockchain record, any tampering attempt can be easily detected by checking the hash. Pricing functions, that are defined in the virtualchain, algorithmically define how expensive it is to create a namespace or to register names in a namespace. The *virtualchain* is a virtual blockchain providing a fork*-consistency model [56]. A fork*-consistency model is an improved version of traditional Byzantine Fault Tolerant systems where the system can cope with more inconsistency, being more secure than a system with fork consistency [57]. A node of virtualchain can be run by each participant in the blockchain network. The underlying blockchain application, currently Bitcoin, occasionally stores data for this virtualchain. This data is marked through a field designated for additional data, called *OP_RETURN*. By constantly storing hashes on the original Blockchain, that can be considered trusted, virtualchain operations can be verified. An example of a *OP_RETURN* code that could be used to register a domain name is `id;hoogland.id`, this registers the domain name to the public key linked to the transaction. This system requires nodes to constantly scan new transactions in the Bitcoin network and look for relevant metadata. Blockstack is new approach in the search of a safe and distributed PKI. The division of the *control plane* and the *data plane* offers certain advantages, but makes the overall architecture

also complex. Where the separated architecture on one hand results in a scalable and secure system, it does require quite some network interaction and large computational investment for small IoT devices. The only way to verify the authenticity of operations in the virtualchain is by downloading the entire blockchain and processing all data. Verification and authentication are not taken into account by the designers of Blockstack, domain names and identifiers can be claimed on a first-come-first serve basis. Key-recovery is also not dealt with in the system. Key-revocation on the other hand is implemented and can be used by executing a `revoke` transaction type in the BNS.

3-1-5 Smart Contract-based PKI and Identity System

In Smart Contract-based PKI and Identity System (SCPKI) [58], an implementation of a decentralized and transparent PKI is introduced using a WoT model. It is built with the help of smart contracts from the Ethereum blockchain. Just as in Blockstack, this system does not store all its data on the blockchain, it uses a distributed storage technique called Inter Planetary File System (IPFS).The design of SCPKI contains two primary components: the earlier mentioned smart contract that dictates the system protocol and functions as an interface connected to the blockchain for the management of identities and attributes. The second element is a client of which the authors from the paper wrote an implementation in Python. The client interacts with the smart contract and other systems such as IPFS, to allow users to fully utilize the system by allowing them to search for and filter attributes. The SCPKI smart contract centers around an entity that is represented by an Ethereum address. This Ethereum address consists out of a private key or a smart contract that an entity has control over and publishes a set of attributes, such as public keys or signatures. Since this system is WoT based, trust needs to be exchanged towards other attributes of different entities in the system in order to become reliable. This can be done by publishing a *binding proof* that consists of a cryptographic signature of the Ethereum address of the entity that wants to verify trust towards an attribute in the network. This signature uses the cryptographic key represented by the attribute, proving that the owner of a private key is associated with an Ethereum address. Because smart-contracts for the Ethereum blockchain are used, users need to pay for operations like signing and revoking attributes or publishing contracts. Users need to pay if they want to sign attributes of other entities. This is unfavorable since it discourages users to submit reverse-binded attributes for entities that they know (since it costs them money). This is a severe downside of the protocol. The implementation does not take privacy into account, it is only suitable for the publication of attributes that the user wishes to make public (such as degree awards). Verification is not taken into account in the current system, users are able to submit the ownership of a public key without proving that they posses the corresponding secret key. Authentication on the other hand is implemented through the WoT architecture. Since the paper mentions nothing about scalability, we assume that nodes need to traverse the entire Ethereum blockchain to lookup values. This makes the protocol unsuitable for extremely large networks with, for example, more than a billion nodes.

3-1-6 Authcoin

Authcoin [59] is introduced in 2016 and also presents a distributed PKI solution with the advantages of a blockchain-based storage system. Authcoin uses a WoT system where users need to verify a users identity. Authcoin implements a challenge response-based validation and authentication scheme. As opposed to other previously discussed implementations, validation and authentication steps need to be succeeded before a node can register the ownership of an identity. This validation step is similar to the validation process deployed by "Let's Encrypt" [60].

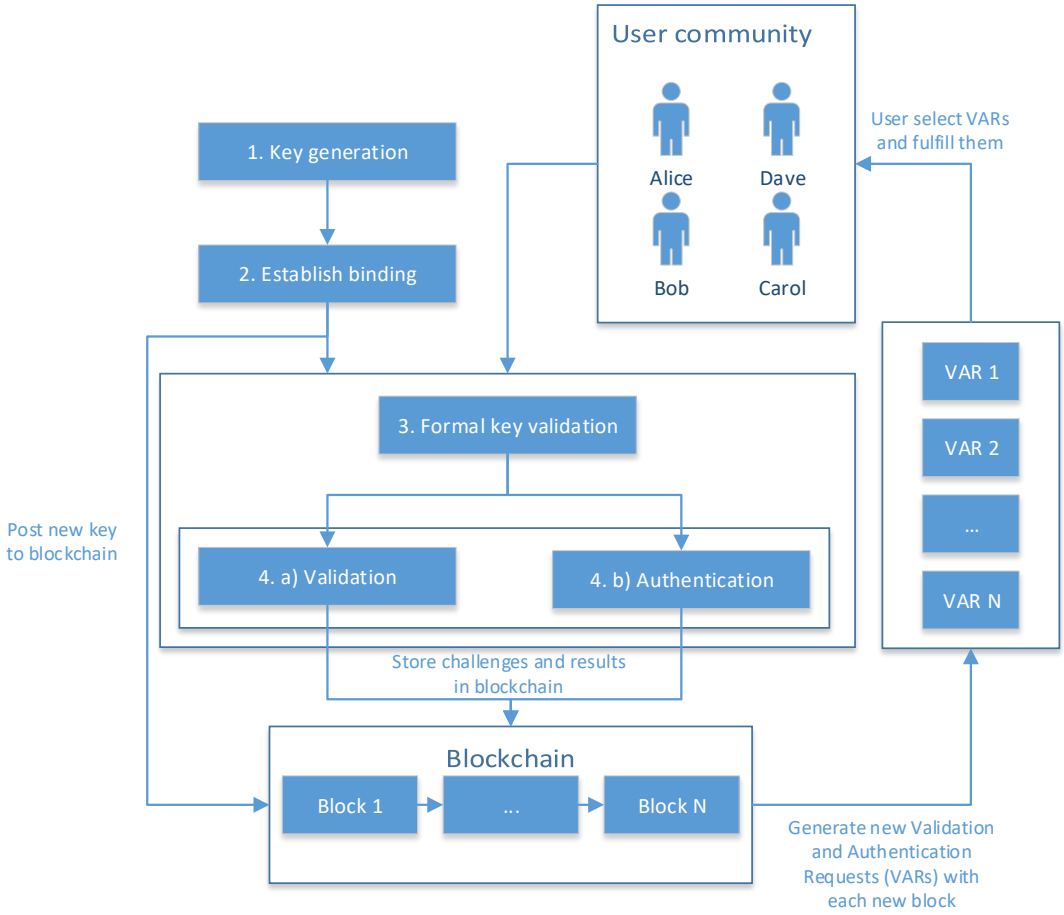

Figure 3-3: Overview of Authcoin workflow.

Image 3-3 shows the Authcoin workflow. A user adds his desired binding to the Blockchain. The validation exists out of, in case of a domainname, a challenger asking the user to provide a certain resource under a specific URI and signing this resource with his private key. The challenger can then verify this domain by simply checking if the signed resource corresponds to the provided public key. This however only verifies that a certain entity actually had access to the domain at the time the Authcoin protocol was run and that this entity has also access to the corresponding key pair [59]. Authentication is also addressed by a challenge response. The

paper does not introduce a standardized technique but gives a couple of possible challenge-response schemes that could be executed. An example could be that a challenger asks a note to send a picture of himself holding a copy of the current issue of a specific newspaper. This type of challenge-based response systems only works for the verification of real identities among known peers.

3-2 Comparison of existing blockchain solutions

Multiple existing techniques have been used and presented, in this section we are going to compare these methods based on their functionalities. As previously discussed in Section 3-1, we have 6 properties that we look at in order to make these widely varying schemes comparable.

1. Validation– Under validation we consider verifying if a user or node has access to the domain or identity that it is trying to register.

2. Authentication– Under Authentication we consider identifying the user or node that is being registered.

3. Scalability– Under scalability we consider if the presented work can be successfully executed by nodes with low storage and low computing power on extremely large (more than a billion nodes) networks.

4. Key recovery– We verify if users are able to recover keys in the presented protocol.

5. Key revocation– We verify if users are able to revoke keys, for example in case of an update of a keypair or if keys are stolen.

6. Privacy preserving– We look if privacy-preserving techniques are implemented for the presented schemes.

Table 3-2 shows a global comparison of the previously discussed work. It should be mentioned that the discussed protocols are all built with their own use-cases in mind. Although they all provide a distributed PKI or DNS, it is not fair to consider a protocol with more checks a better protocol compared to a protocol with less. It only says that, taking our relevant use-case (a distributed PKI optimized for IoT devices), certain protocols could be better to learn from.

3-3 Other non-blockchain based PKI solutions

The drawbacks of current centralized PKI solutions have been acknowledged for over a decade, research is being conducted in solving these issues from multiple angles. This Section discusses non-blockchain based initiatives.

Table 3-2: Comparison of the presented blockchain based PKI solutions

Protocol	Verification	Authentication	Scalable	Key recovery	Key revocation	Privacy preserving
Namecoin	no	no	no	no	no	no
Certcoin	no	no	yes	yes[1]	yes	no
PB-PKI	no	no	no	yes[1]	yes	yes
Blockstack	no	no	yes	no	yes	no
SCPKI	no	yes	no	no	no	no
Authcoin	yes	yes	no	no	yes	no

[1] Key recovery is implemented through the usage of a n-threshold secret sharing scheme. Although this is a key recovery technique, we do not consider it practical for our use-case since it requires a node to trust his environment.

3-3-1 Certificate Transparency Project

A proposed solution for some of the shortcomings of current PKI solutions is the Google Certificate Transparency project [61]. This project aims to audit and monitor certificates in the current SSL infrastructure. The solution proposes maintaining a public, append-only log managed on a number of independent servers throughout the world. Each server, of which each CA could, for example, maintain one, should for itself monitor and audit newly added certificates. This enables domainowners to see all certificates issued for his domain, and thus would easily be able to spot any malicious certificates. Adoption of Certificate Transparency is currently not ubiquitous, starting from April 2018 however, Google Chrome plans to start enforcing the use of Certificate Transparency in all newly issued certificates [62]. It is to be expected that the adoption will rapidly take off after this day since few site operators will be willing to use a certificate that is not trusted by Chrome. Although this does add transparency to the current architecture, the specification cannot guarantee that invalid certificates are not present on the logs. This proposal also does not remove the centralized architecture, where Certificate Authorities are still assumed a trusted role as the only entities who can issue certificates and update logs.

3-3-2 Revocation Transparency

The authors of Certificate Transparency project proposed an extension of CT called Revocation Transparency (RT). RT combats the problem of validity by auditing and monitoring Certificate Revocation List (CRL)s. Clients that have RT implemented will only accept certificates that are provided with proof from the distributed RT log that the certificate is not revoked. Just as with CT, this solution does not combat the existing centralized points of failure.

3-4 Designing a dedicated blockchain versus building on existing solutions

When using blockchain technology, there are three approaches when considering how to embed a blockchain to your application:

1. Build a dedicated blockchain for your application.
2. Build your application through smart-contracts provided by a blockchain.
3. Use an existing blockchain and build your application on top of this.

Not a lot of research has been done on the advantages and disadvantages of these approaches when considering the implementation of a distributed PKI. The Blockstack protocol [54] is one of the few that does look into the advantages and disadvantes. In this paper, the authors describe how they first deployed their Blockstack system in its own dedicated branch of the Namecoin blockchain, and eventually decided to migrate and build their system on top of Bitcoin. The reasons behind this choice were:

1. Blockchain security– A large downside of running your own blockchain is security. A known blockchain attack is the *51 % attack*. This attack can occur if miners have more than 51% of the mining power. A miner has then the ability to attack the network and rewrite recent blockchain history and steal cryptocurrency using *double spend* attacks [11]. The Blockstack designers noticed that at a certain point in time, a node was in the possession of more than 60% of the computing power, and theoretically able to perform attacks on the network. The possibility of this occurring in large blockchains like the Bitcoin network is extremely small because of the large number of participants. The network administrators also indicated the symptoms of selfish mining attacks [51], where miners withheld mined blocks and try to work on new blocks faster than other peers.
2. Network reliability and throughput– The designers of Authcoin noticed that specific network throughput drops and network latency spikes occurred while their system was running on their dedicated fork. After investigating this more thoroughly, the authors discovered that these occurrences originated from minor software bugs in the Namecoin blockchain. These type of irregularities are less likely to occur in blockchains with more stakeholders.
3. Consensus-breaking changes– The designers of Blockstack noticed that, in order to apply major updates like namepricing, *hard forks* needed to be executed in which everyone on the network must upgrade their software, and where nodes on previous versions can no longer participate in the network [51]. Anecdotal evidence suggests that it can be hard to get miners to upgrade their software because they do not have enough incentive to spend engineering hours on maintaining a small cryptocurrency like Namecoin. The designers even noticed that, after a major upgrade of their Namecoin daemon, that a significant number of miners dropped out and never came back online. This fluctuation is less likely to occur with larger blockchains.

The main argument of the Blockstack authors to build their system on top of an existing blockchain solution is security, the authors argue that there does not exist enough computing power to warrant multiple secure blockchains. We argue, however, that the experiences of the Blockstack authors are based on experiments where the total number of nodes is low. For an identity service aimed at providing the IoT a key infrastructure we expect to have a system that will be adopted by millions (or even billions) of nodes, resulting in a secure system.

SCPKI (introduced in Section 3-1-5) is built on top of the Ethereum blockchain with the usage of smart contracts. Similarly to Blockstack, this has as main advantage that the system designers can build (and rely) on a widely adopted existing network solution. A disadvantage is that the system needs to be designed with the rules of the network as starting point. In the case of SCPKI this results in requiring nodes to pay with gas for simple transactions like publishing, signing and revoking attributes and signatures. It can be considered a barrier for nodes to participate or even join in the network, which is undesirable since poor adoption can counter the future viability of the system.

Chapter 4

Research context and methodology

With the introduction of blockchain technology and the increasing importance of scalable network structures, the potential of a blockchain based PKI is promising. While several distributed approaches have been suggested, none of the proposed solutions have combined scalability with identity verification and with practical key revocation and key recovery. The goal of this thesis is to prove that an infrastructure with these features can be constructed. In this chapter, we discuss all the assumptions that we make for our protocol. We end with a description of the research methodology that we used to achieve the objective of this research.

4-1 Research context

The following assumptions were made when designing our protocol DECKIN:

1. All IoT devices are currently accessible through a unique IP-address on the Internet.
2. The network consists of a billion nodes that all want to register a public key on our system to simplify authentication throughout the network.
3. We assume that it is undesirable for nodes to have regularly interaction with other nodes for verification purposes. This makes the network overview cluttered for system administrators. We also assume that the probability of all nodes being able to handle these large volumes of traffic is unlikely.

We assume devices want to use our PKI for verification purposes. Verification of the devices is done as follows: Assume IoT device Alice wants to authenticate with device Bob, both devices have registered their IP address and public key on our presented blockchain solution. Alice now sends Bob a message of the form (IP_A, pk_A). Bob now first needs to verify that pk_A indeed belongs to Alice her IP-address. If this is the case, he sends a random challenge message h to Alice. If Alice is able to respond with $\sigma = sig_{sk_A}(h)$ such that $ver_{pk_A}(\sigma, h) = 1$, Bob is able to trust that she is, indeed, the owner of the IP address. This makes Alice authenticated successfully.

Blockchain Our protocol is built on top of Namecoin. Namecoin is one of the first Bitcoin forks as it was was released in 2011. The underlying blockchain technology of Namecoin is almost identical to that of Bitcoin, only adding different transaction types. As discussed in Section 2-2-4, the traditional Bitcoin blockchain has scalability issues. A maximum of seven transactions can be executed per second, this amount is insufficient when considering a PKI solution that needs to serve the ever-growing IoT infrastructure. Our protocol does not focus on scalability of the underlying blockchain since this is a research topic on itself, we consider it to be future work on which we elaborate slightly in Chapter 7-2. We focus on what is posted on the blockchain and what interaction is done between nodes and miners.

For scalability reasons, we do assume that our blockchain is built on top of Ethereum with Sharding implemented. Sharding allows nodes and transactions to be divided into smaller groups. Because of this, more transactions can be validated and verified at once, lowering the required capacity on nodes. We assume that our system uses Casper, a consensus algorithm designed for Ethereum based on the proof of stake consensus model. As discussed in Section 2-2-1, this model does not require nodes to invest computationally demanding resources but only requires them to invest a stake. Nodes have an incentive to participate in this model because of the transaction rewards. This means that nodes that want to add their identity to the blockchain, will have to pay transaction costs.

4-2 Research methodology

To achieve our main research goal of proving the feasibility of a distributed PKI that is scalable, adds identity verification, practical key revocation and recovery we design and implement our protocol called DECKIN. The protocol is built with the assumptions defined in Section 4-1. Our research methodology is based on the design science type. According to Hevner's 7 design guidelines for design research [63], we need to demonstrate the utility, quality, and efficacy of our proof of concept. In Section 4-2-1 we discuss our Design requirements and in Section 4-2-2 we elaborate on how we evaluate our proof of concept that we introduce in the following chapters.

4-2-1 Design requirements

In order for our research to successfully be accomplished, we create our protocol DECKIN of which its functionality should be sufficient to solve our research question:

"How can a scalable PKI be designed with practical key recovery, key revocation and identity verification functionality without a single point of failure to improve IoT security?"

We will therefore have to create a basic PKI that, not only is able to offer basic PKI functionality like registering and updating an identity, but also to provide the following characteristics:

Scalability The system should be able to be provide functionality for a large number of participants. We focus on the operations that are performed on top of the blockchain. We

use 1 billion as the threshold when evaluating if a system is scalable or not. For reference, as of June the 15th 2018, the Bitcoin network has 9908 nodes*.

Key recovery The system needs to have a key recovery mechanism implemented. If the secret key of an entity is lost, a node must be able to retrieve this key to remain control over his identity.

Key revocation This system needs to provide nodes with the possibility to revoke a public key that is corresponding to their identity. This enables nodes to revoke a key if their secret key has been stolen by an adversary or lost. Revoking the stolen key will prevent the adversary to misuse it.

No single point of failure Our system should be built in such a way that it does not have a single point of failure.

Identity Verification functionality Our system should provide identity verification functionality for nodes that want the register an identity. The function of this functionality is confirming that a node actually has access to the identity he claims to possess

4-2-2 Evaluation

In order for our research to successfully be accomplished, we need to implement our protocol and evaluate the utility, quality and efficacy of it. It would be inaccurate to present a comparison between DECKIN and the previous work due to a lack of existing work that aims to satisfy the same goals as our research. We therefore discuss each element of our system individually in Section 7-1 and evaluate if it meets the requirements defined in Section 4-2-1. In order to draw conclusions on scalability, we measure the practical performance of our proof of concept implementation and lookup if it meets our expectations and requirements.

*According to https://bitnodes.earn.com/

Chapter 5

DECKIN: DECentralized Key INfrastructure

In this chapter we introduce our protocol called DECKIN which is a distributed PKI that is designed for IoT devices with little resources, a functional key-management solution and a verification protocol that is executed whenever a node wants to register an identity. We built our protocol with the pre-defined requirements from Section 4-2-1. In the first section of this chapter we discuss what existing protocol we extended to create DECKIN. We then proceed with two sections explaining what types of keys exist in our system and how accumulators are used in our system for efficient key lookup in Section 5-2 and Section 5-3. Section 5-4 discusses the Identity Validation Algorithm that we use to verify if nodes possess the identity that they want to register. Finally, in Section 5-5, we present a system overview of DECKIN. In this section each transaction type is defined and all belonging interactions are explained. We conclude with a theoretical performance analysis, in which we evaluate the computational complexity and the storage complexity of DECKIN.

5-1 Protocol fundament

Our existing literature analysis in Section 3-2 shows that the Certcoin [25] protocol is one of the few protocols that has been implemented scalable, theoretically supporting more than a billion nodes. Certcoin is built on top of Namecoin [51], an open source altcoin. Being an open source solution, Certcoin is easily extendable. This is why we choose Certcoin as the basis of DECKIN, extending it to meet our requirements. As described in Section 3-1-2, Certcoin provides us with a simple PKI. In contrast to the Certcoin design, we argue that it is not necessary for our protocol to add efficient lookup functionality if verification is implemented efficiently. Adding an efficient lookup mechanism makes the system unnecessarily complex. In Certcoin, for example, this requirement results in the system designers to add a publicly accessible DHT in which all system nodes need to participate. We argue that scalability and accessibility is more important, key lookup functionality can simply be replaced by asking

nodes what their key is, and then verifying that the provided key is, indeed, the key received from the node.

5-2 System keys

There are two different keys in DECKIN, the *general* key (GEN-key) and the *PUF* key (PUF-key). Both keys are part of an asymmetrical cryptographic scheme based on ECC. We choose this scheme because ECC requires smaller keys compare to non-ECC cryptographic schemes to provide an equal level of security. The exact workings of ECC are explained in Section 2-1-2, in this Section we only focus on the role of the GEN-key and the PUF-key. From now on, the public and secret key of the GEN-key and PUF-key are respectively referred to as pk_{gen}, sk_{gen} and pk_{puf}, sk_{puf}. All participants in our system are required to use a GEN-key. For compatibility reasons, the usage of a PUF-key is optional (but recommended). We will elaborate more on the what the exact requirements for PUF-technology technology is further in this Section. Both the GEN-key and the PUF-key are 256 bit keys, similar with keys in the bitcoin protocol.

5-2-1 General key

When Alice wants to register her identity in DECKIN, she needs to submit a `register` transaction to the blockchain. This transaction always contains a pk_{gen} key. Now that this key is publicly link-able to Alice (more details on how these transactions are constructed in our system overview, Section 5-5), Alice can submit transactions in which she proves that she is the owner of sk_{gen}^{Alice}, that `revoke` or `update` this initial registration.

5-2-2 PUF key

Contrary to the pk_{gen}, a pk_{puf} key does not have to be included with a registration transaction, this is optional. It is, however, recommended to include a PUF-key. Advantages to include a PUF-key when registering an identity are:

1. PUF keys do not require a node to store his sk_{puf} physically on the device, this makes the keys less vulnerable for attackers trying to extract keys.
2. A sk_{puf} can always be recovered as long as the hardware of a node stays the same. This makes this key less vulnerable to for example system crashes.
3. DECKIN makes an hierarchical distinction between the GEN-key and the PUF-key. The GEN-key is in hierarchy lower than the GEN-key. This means that if two `update` transactions are submitted to the blockchain for a same identity value, with the only distinction that the first transaction only is accompanied by a GEN-key, and the second transaction by a GEN-key and PUF-key, the second transaction is considered to be the correct one and the first transaction is ignored. Even if the first transaction was submitted earlier than the second transaction.
4. Nodes are not able to submit a `revoke` transaction if they do not have a PUF-key.

We choose for this approach, where system users have the ability to choose if they want to include a pk_{puf} during registration for compatibility reasons. The *PUF* keypair in DECKIN needs to be generated with an SRAM PUF key extraction system. For our design, we choose to go for a working SRAM PUF solution provided by Intrinsic ID called BROADKEY*. This SRAM PUF system has a couple of basic requirements. It needs, for example, to shield a part of its SRAM to prevent other applications to access these cells and read their initial value. Although all requirements can technically be fulfilled by almost each IoT device, we argue that the likelihood of this happening in a short period of time is low. By using this hybrid system in DECKIN, we enable devices that cannot fulfill these PUF requirements to also be able to participate in our system. We believe that this is essential for a system in order for it to be widely adopted.

5-3 Accumulator lookup

To make DECKIN scalable, it is crucial that nodes can verify if private keys belong to a certain identity without having to traverse the entire blockchain. Traversing the entire blockchain would require nodes to store the whole blockchain locally, which is not scalable and therefore undesirable. This is why we use the same approach as in the Certcoin [25] protocol. We store a cryptographic accumulator (introduced in Section 2-1-6) and its corresponding witness in the blockchain, added to each transaction. Each time a block miner adds a transaction that creates, updates or revokes a public key, the block miner who processes this transaction will have to update the accumulator and include the updated accumulator value in the block. Our protocol requires an accumulator that has the public verifiability property, so users can check if the updated accumulator value that is provided by the block miner is correct. This is why we use the extended Merkle tree accumulator [25], discussed in more detail in Section 2-1-6. In Appendix A a summary can be found of all the operations belonging per transactions type (`register`, `update` and `revoke`) and that need to be executed to implement our accumulator functionality. Image 5-1 displays a simplified example of how accumulators are used in our protocol.

5-4 Identity validation algorithm

Transaction structure A transaction $T_{m,n}$ is defined as transaction number n in block number m. Each transaction has the structure as shown in Equation 5-1, and has four elements. The first element, called ι, is an identifier. In DECKIN this identifier is always an IP address. θ is the transaction type, this is always one of the following 3 values: `register`, `update` or `revoke`. τ is an array containing the index of all valid `register` transactions in the latest added block to the blockchain. The last part of each transaction, ρ, is called the payload. It contains a varying number of items, depending per transaction type. Table 5-1 shows all different payloads belonging to their transaction type.

$$T_{m,n} = (\iota|\theta|\tau|\rho) \tag{5-1}$$

*https://www.intrinsic-id.com/products/broadkey/

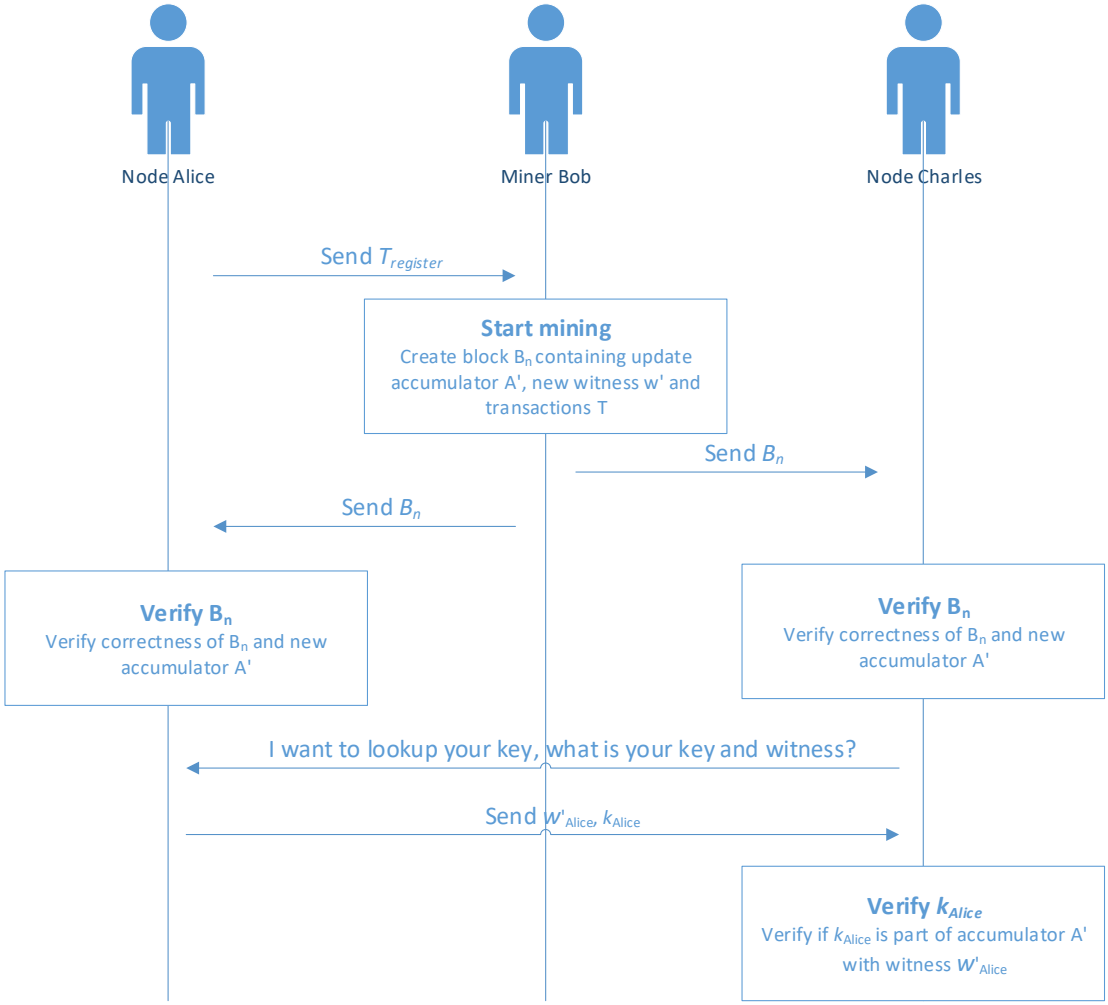

Figure 5-1: A simplified example of how accumulators are used in our protocol. Node Alice registers an identity. The miner adds this transaction, an updated accumulator A' and its corresponding witness w' to the blockchain. If node Charles now wants to verify the identity of Alice, she first asks Alice for her key and witness. Charles is now able to verify Alice' identity efficiently by verifying these values with the most recent Accumulator value of the blockchain.

One of the unique features of DECKIN is that we introduce an identity validation algorithm. This algorithm validates the identity that a node wants to register when submitting a `register` transaction. The validation procedure is in detail described in Algorithm 2. If a Node A wants to validate a transaction $T_{m,n}$, he extracts the IP address and port number that are included in $T_{m,n}$. It then sends a `HTTP GET` query to the IP address on the port number. If the node maintaining this IP address returns a SHA256 hash of the `register` transaction that is being validated, validation is successful.

Algorithm 2 The identity validation algorithm for $T_{m,n}$ by node A

if $T_{m,n}.\tau ==$ `register` **then** ▷ non-`register` transactions always fail
  $h =$SHA256$(T_{m,n})$
  A sends a `HTTP GET` query to `http://IP:port` ▷ IP and port as in $T_{m,n}$
  if query returns a response r **then**
    if r equals h **then**
      validation succeeded
    end if
  end if
end if
validation failed

In DECKIN, each new transaction has an array containing all indexes of all valid `register` transactions in the latest added block to the blockchain. So if $T_{m,n}$ is a transactions that is going to be added to block B_m, the array would contain the indexes of all valid `register` transactions in block B_{m-1}. This means that each node that wants to add a transaction to the blockchain, first needs to fetch the last block of the blockchain, and then loop over each transaction. If the transaction is a `register` transaction, the identity validation algorithm needs to be executed. The node ends up with a list of valid `register` transactions, and adds the indices of these valid transactions to the transaction that he is going to submit to a miner. Image 5-2 gives a demonstration of this concept. The miner in turn receives a number of transactions. The miner also creates an array with transaction indices of valid transactions, and ignores all transactions that provide an array that is not identical to the array of the miner. It therefore executes the protocol defined in Algorithm 3.

A `register` transaction in DECKIN is only considered valid if it has been confirmed in a succeeding block. This means that all nodes that have been added in the latest block of the blockchain through a `register` transaction need to wait for the addition of another block before they are recognized by the DECKIN network. This is demonstrated in Image 5-3.

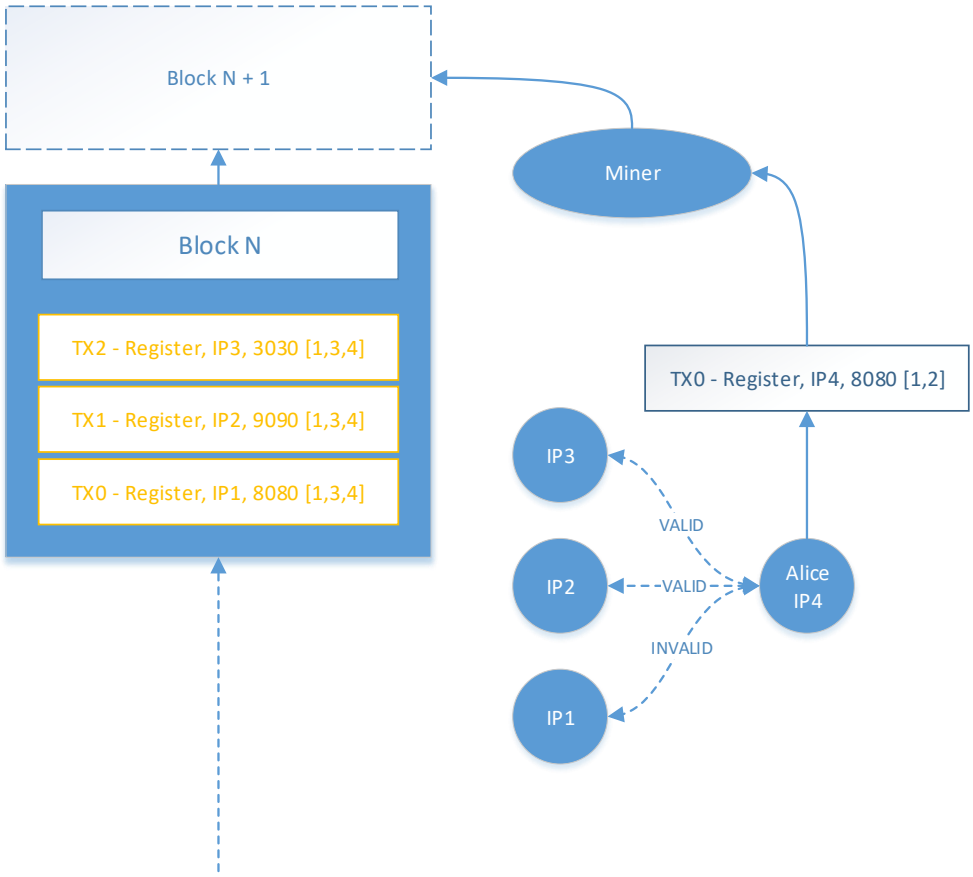

Figure 5-2: A simplified example demonstrating the Identity validation algorithm executed by node Alice. In order to submit a transaction, Alice needs to lookup all the register transactions from Block N. The indices of all transactions that are considered valid are added to her own transaction.

Algorithm 3 The identity validation procedure for miner M

$tx \leftarrow$ `received transactions`
$txValid \leftarrow []$
$ref \leftarrow []$
for each transaction $T_{m,n}$ in block B_m **do** ▷ Latest block from blockchain
  if $T_{m,n}.\theta ==$ `register` and Valid($T_{m,n}$) **then** ▷ Algorithm 2
    $ref.push(n)$
  end if
end for
for `each transaction` T_i `in` tx **do**
  if $T_i.\tau = ref$ **then**
    $txValid.push(T_i)$
  end if
end for
return txValid

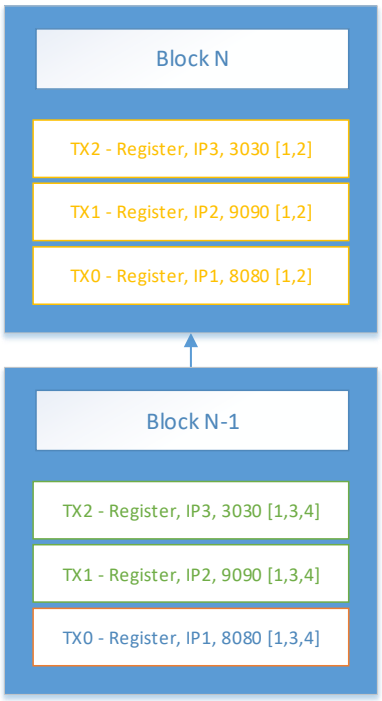

Figure 5-3: A simplified example demonstrating how `register` transactions are only valid after confirmation in the succeeding block. Transaction number 0 from block N-1 is not confirmed in block N, and therefore not considered valid.

5-5 System overview

DECKIN is built on top of Certcoin, this means that our system consists out of nodes and miners. In the upcoming subsections, we describe for each transaction type what steps are required to submit a transaction.

Table 5-1: Example of the different payloads in the different transactions types.

Protocol	Payload	Explanation
`register`	pk_{gen}, σ_{gen}, `port#`, pk_{puf}[†], σ_{puf}[†], AC[†]	$\sigma_{gen} = sign_{sk_{gen}}(\iota \|\| \tau)$ $\sigma_{puf} = sign_{sk_{puf}}(\iota \|\| \tau)$
`update`	$pk^{old}_{gen}, \sigma^{old}_{gen}, pk^{new}_{gen}, \sigma^{new}_{gen}$[‡]	$\sigma^{old}_{gen} = sign_{sk^{old}_{gen}}(\iota \|\| pk^{new}_{gen})$ $\sigma^{new}_{gen} = sign_{sk^{new}_{gen}}(\iota \|\| \tau)$
`revoke`	pk_{puf}, σ_{puf}	$\sigma_{puf} = sign_{sk_{puf}}(\iota \|\| \theta \|\| \tau)$

[†]As discussed in Section 5-2, the addition of these elements is optional (but recommended).

[‡]If added during registration, the user can add pk_{puf} and σ_{puf} to submit an `update` transaction that is higher in hierarchy (as explained in Section 5-2) compared to an update transaction without a PUF-key.

5-5-1 Register

If a node A wants to register an identity to the PKI, A needs to create a `register` transaction and submit this transaction to a miner in the system. Once this miner has submitted this transaction to the blockchain, A needs to be able to participate in the identity validation algorithm for the length of the creation of a succeeding block. Assuming A would want to register IP address 131.180.77.82 (TU Delft website), and the execution of the identity validation algorithm for each register transaction of the latest block returned [1,5], A wants to use port 8080 to execute the validation algorithm, in that case the transaction would have the structure as in Equation 5-2.

$$T_{m,n} = (131.180.77.82|\texttt{register}|[1,5]|pk_{gen}, \sigma_{gen}, 8080, pk_{puf}, \sigma_{puf}, AC) \tag{5-2}$$

We will now discuss all the steps that are taken per actor, step by step. We start with the registrar.

Registrar If A would want to register identity ι, he would have to execute the following steps in the defined order:

1. Generate a GEN-keypair (pk_{gen} and sk_{gen}) to be used for future communication.
2. Extract a PUF-keypair and the activation code AC through the software based SRAM PUF.
3. Execute the identity validation algorithm (Algorithm 2) for each `register` transaction of the latest block on the blockchain. Add the outcome to the array τ.
4. Submits a `register` transaction $T_{register}$ (as defined in Equation 5-2). The payload in this transaction is defined as follows:

$$\sigma_{gen} = sign_{sk_{gen}}(\iota||\tau)$$
$$\sigma_{puf} = sign_{sk_{puf}}(\iota||\tau)$$

Miner Each miner m that participates in the network and receives the register transaction $T_{m,n}$ does the following:

1. The miner compares the transaction indices in τ with his own array, if the arrays match the miner proceeds. Else the miner ignores the transaction.
2. Verifies that ι is not yet registered on the blockchain.
3. Verifies that signatures σ_{gen} and σ_{puf} are constructed correctly according to their definition defined in Table 5-1
4. The block miner now adds a tuple containing the identity and the registered key to the accumulator. The block miner includes the transaction, the new accumulator and its corresponding witness to the new block.

Network nodes Each node i in the network, that receives a newly added block B_m from the network, does the following:

1. Verify that ι is not registered yet.
2. Verify that both signatures in ρ are correct.
3. Verify that the accumulator was updated correctly, if it was, update the locally stored accumulator and the witness linking to the key of node i. Else, ignore the block.

5-5-2 Update

Update transactions remain similar compared to the original Certcoin protocol [25]. If a node A wants to update his pk_{gen}^{old} with the new key pk_{gen}^{new}, an `update` transaction needs to be constructed by A and once again submitted to a miner. Using the same values for ι and τ as in our previous example, a valid `update` transaction is shown in Equation 5-3.

$$T_{m,n} = (131.180.77.82|\texttt{update}|[1,5]|pk_{gen}^{old}, \sigma_{gen}^{old}, pk_{gen}^{new}, \sigma_{gen}^{new}) \tag{5-3}$$

Updater Node A executes the following steps when he wants to create an `update` transaction:

1. Submit an `update` transaction with the same structure as Equation 5-3. The variables in this transaction are defined as follows:

$$\begin{aligned} pk_{gen}^{old} &= \text{The old key that needs to be updated.} \\ \sigma_{gen}^{old} &= sign_{sk_{gen}^{old}}(\iota, pk_{gen}^{new}) \\ pk_{gen}^{new} &= \text{The new key.} \\ \sigma_{gen}^{new} &= sign_{sk_{gen}^{new}}(\iota||\tau) \end{aligned}$$

Miner Each miner m that participates in the network and receives the `update` transaction $T_{m,n}$ does the following:

1. It first verifies that pk_{gen}^{old} currently corresponds to ι in DECKIN.
2. Verifies that both signatures are correct.
3. Remove the tuple containing the identity and the *old* key from the accumulator, thereafter adds the tuple with the new key. The miner then includes the transaction, the new accumulator and its corresponding witness to the new block.

Network nodes Each node n in the network, that receives a newly added block B_m from the network, does the same verification steps as the miner. Each node verifies if miner m correctly removed the old key and after that added the new key to the accumulator. If it did, it updates its own accumulator and witness value. Else, it ignores the block.

5-5-3 Revoke

If a node A wants to revoke his public key that is submitted to the PKI, he needs to prove he has access to the secret key of this public key and to the secret key of his PUF public key that was registered during initialization. Assuming we once again use the same values for ι and τ, an example of a valid `revoke` transaction is given in Equation 5-4.

$$T_{m,n} = (131.180.77.82|\texttt{revoke}|[1,5]||pk_{puf}, \sigma_{puf}) \tag{5-4}$$

Revoker Node A executes the following steps if it wants to revoke his key from the PKI.

1. Reconstruct sk_{puf} with a PUF algorithm and the activation code that is registered during initial registration of the identity.
2. Generate σ_{puf}, a signature over $(\iota||\theta||\tau)$ signed with sk_{puf}.
3. Submit the `revoke` transaction with a similar structure as in Equation 5-4 and submit it to a miner.

Miner Each miner m that participates in the network and receives the revoke transaction $T_{m,n}$ does the following:

1. Verify that pk_{puf} corresponds to the *puf* key that belongs to ι in the system.
2. Verify that signature σ_{puf} is constructed correctly.
3. The block miner now removes the tuple containing the identity and the registered key from the accumulator. The block miner includes the transaction, the new accumulator and its corresponding witness to the new block.

Network nodes Each node n in the network, that receives a newly added block b from the network, does the same verification steps as the miner. It should verify that the accumulator is updated correctly by the miner. If it is correctly updated, its local accumulator value and witness should be updated. Else, it should ignore the transaction.

5-6 Theoretical performance

In this section we discuss the theoretical performance of DECKIN. We focus on the computational complexity and the storage complexity of the protocol. Since DECKIN is built on top of Namecoin, we disregard the complexity of the basic operations that DECKIN has; these are well studied [64]. We will focus on the newly added features of our protocol, Accumulator operations, the Identity Validation Algorithm and PUF Key generation. Table 5-2 describes all the variables that are being used in our analysis of DECKIN.

Table 5-2: Symbols used in the performance analysis of DECKIN.

Symbol	Description
n	Number of registered identities in DECKIN
m_i	Number of transactions in block i
$l_{i,j}$	Number of validate lookups for transaction i in block j
$m_j^{register}$	Number of `register` transactions in block j

5-6-1 Computational complexity

Accumulator operations

In Table 5-3 we summarize the computational complexity of each operation that we use for our cryptographic accumulator. The creation of a cryptographic accumulator is constant in time, it always is the same operation no matter how many elements the accumulator will contain. Element addition is of cost $\mathcal{O}(\log(n)^2)$. If an accumulator, containing n Merkle roots, needs to be updated, and each Merkle root contains hash values, n hash operations are done to update the accumulator (combining all the nodes to create a new hash root). The accumulator now contains $n+1$ Merkle roots. Element removal is done by replacing a value in a Merkle root with `null`, making it publicly verifiable for other nodes in the system. This operation will cost at most $\log n$ calculations, depending on the location of the key in the accumulator. Updating a witness is constant in time, since it is always the same operation, regardless of the accumulator location.

Table 5-3: The complexity of the different operations for our cryptographic accumulator usage.

Protocol	Operation	Cost
Accumulator	Initialization	$\mathcal{O}(1)$
Accumulator	Element addition	$\mathcal{O}(\log(n)^2)$
Accumulator	Element removal	$\mathcal{O}(\log(n))$
Accumulator	Witness update	$\mathcal{O}(1)$

Identity Validation Algorithm

In Table 5-4 we summarize the computational complexity of all the operations in our Identity Validation Algorithm. Looking up the validity of a register transaction $T_{i,j}$ is constant in computational complexity, it is the execution of one GET query and one hash generation. A node, that is stand-by waiting for its `register` transaction $T_{i,j}$ to be confirmed, has a computational complexity of $\mathcal{O}(l_{i,j})$. For each lookup the node needs to respond with a hash value of the transaction that the node wants to append to the blockchain. Since this hash value is constant, only one hash needs to be calculated. As last, the computational complexity of looking up the valid transactions of the latest block B_j is equal to the number of `register` transactions that this block contains. For each transaction, a hash needs to be generated, a GET query needs to be executed and a response needs to be compared.

Table 5-4: The complexity of the different operations for our Identity Validation Algorithm.

Operation	Cost
Looking up validity register transaction $T_{i,j}$	$\mathcal{O}(1)$
Being stand-by to get confirmed	$\mathcal{O}(l_{i,j})$
Looking up valid transactions block j	$\mathcal{O}(m_j^{register})$

5-6-2 Storage complexity

This Section focuses on the storage requirements for DECKIN. We analyze what amount of storage is required for nodes for each type of operation, to be able to analyze if each operation can be executed by each type of node. The Identity Validation Algorithm does not require an extensive storage complexity analysis. Each operation only requires a node to store a single block of the blockchain. The accumulator does require an analysis. Table 5-5 shows a summary of the result of our analysis. As can be seen, the storage complexity has similarities with the computational complexity. The accumulator has a storage complexity of $\mathcal{O}(\log(n)^2)$ and a witness has a complexity of $\mathcal{O}(\log(n))$. If an accumulator would need to be recreated, a node would need to access all elements in a blockchain, therefore this operation has a storage complexity of $\mathcal{O}(n)$. Since all accumulator operations are publicly verifiable, this operation does not need to be executed and we therefore do not consider this a problem.

Table 5-5: The storage complexity for elements of DECKIN.

Element or action	Storage complexity
Accumulator	$\mathcal{O}(\log(n)^2)$
Witness	$\mathcal{O}(\log(n))$
Recreating accumulator	$\mathcal{O}(n)$

Chapter 6

Evaluation of DECKIN

In Chapter 5 we designed a distributed PKI in which nodes are able to register, update, revoke, verify and recover an identity. We finished with a theoretical performance analysis in Section 5-6 by looking at the computational and communication complexity. In this chapter we will evaluate our protocol. We start with a security analysis of DECKIN in Section 6-1. Section 6-2 discusses the results of our own implementation of elements of DECKIN, which we used to verify scalability claims that we made during the design. We also try to validate if the claimed theoretical performance from Section 5-6 can be approached in our own implementation. Finally, we briefly discuss the results of our validation and the DECKIN protocol in general in Section 6-3.

6-1 Security analysis

DECKIN uses the Elliptic Curve Digital Signature Algorithm (ECDSA) as defined in FIPS 182-6 [65] for its key generation. We discussed the details about this protocol in Section 2-1. DECKIN further uses the SHA-256 hashing algorithm as defined in FIPS 180-4 [66] as its secure hashing algorithm. These protocols have been proven secure in their respective papers. In this section, we provide a security analysis of the Identity Validation algorithm and of each transaction type in DECKIN. All the transactions rely heavily upon the ECDSA, we will discuss each transaction on itself.

6-1-1 Update transaction

The update transaction is used to update the key that belongs to a registered identity in DECKIN. It is essential that a key can only be updated by the identity that initially registered the old key. We call this property *authenticity* and prove that our update transactions are authentic.

Lemma 6-1.1. *(Authenticity). An adversary is not able to submit a valid update transaction to the network for an identity of which he does not posses the secret key.*

Proof: If a node wants to submit an `update` transaction, he needs to add his former public key pk_{gen}^{old}, his new public key pk_{gen}^{new} and two signatures σ_{gen}^{old} and σ_{gen}^{new} to the payload of his transaction. These signatures are constructed as shown in Equation 6-1 and Equation 6-2.

$$\sigma_{gen}^{old} = sign_{sk_{gen}^{old}}(\iota||pk_{gen}^{new}) \tag{6-1}$$

$$\sigma_{gen}^{new} = sign_{sk_{gen}^{new}}(\iota||\tau) \tag{6-2}$$

In order to create σ_{gen}^{old} the user needs to posses sk_{gen}^{old}. This means that a user needs to possess the to be updated secret key (sk_{gen}^{old}) in order to reconstruct σ_{gen}^{old} and subsequently σ_{gen}^{new}. Since each miner and network node can publicly verify these signatures, the only way an adversary can submit an `update` transaction to the network for an identity they do not possess is if the adversary is able to recreate sk_{gen}^{old} and successfully construct a valid σ_{gen}^{new} and σ_{gen}^{old}.

This would mean that the user is able to break the ECDSA, which is, assuming the algorithm is implemented correctly, as hard as solving the discrete logarithm problem [67] and therefore unfeasible for a computationally bound adversary.

We therefore consider the update transaction to be secure. □

6-1-2 Revoke transaction

The `revoke` transaction is used to revoke a key that has been registered in combination with an identity in DECKIN. Similarly to the `update` transaction, it is essential that a key can only be updated by the identity that initially registered that specific key. We therefore once again define *authenticity*, this time in the context of an `revoke` transaction, and prove that the `revoke` transactions are authentic.

Lemma 6-1.2. *(Authenticity). An adversary is not able to submit a valid revoke transaction to the network for an identity of which he does not posses the secret key.*

Proof: If a node wants to submit a `revoke` transaction, he needs to add his public PUF key pk_{puf} and a signature σ_{puf} constructed as shown in Equation 6-3.

$$\sigma_{puf} = sign_{sk_{puf}}(\iota||\theta||\tau) \tag{6-3}$$

In order to create σ_{puf} the node must create a signature with sk_{puf} over the concatenation of the identity (ι), the transaction type (θ) (in this case `revoke`) and τ. Without having the possession over sk_{puf}, an adversary would not be able to create this signature. Just as in the `update` transaction, the only way an adversary is able to submit a `revoke` transaction to the network for an identity of which it does not posses the secret key is if the adversary is able to recreate sk_{puf}, meaning the user is able to break the ECDSA, which is unfeasible for a computationally bound adversary.

We therefore consider the revoke transaction to be secure. □

6-1-3 Register transaction

The `register` transaction is one of the fundamental transaction types in DECKIN. It is used to register an identity and an accompanying public key to the blockchain. In contrary to the `update` and `revoke` transactions, nothing is submitted to the blockchain that can be used to prove ownership of an identity. DECKIN introduces the Identity Validation Algorithm (in Section 5-4) to prevent users to claim identities that they do not possess. Similarly with the `update` and `revoke` transactions. This operation would be considered completely secure if users would only be able to register an identity if they also control it. The large number of actors in this operation makes this less trivial than the previous two operations. The registration of a node consists of 2 phases:

1. The initial registration where it is added to the latest block N.
2. Only after confirmation by the transactions in the succeeding block $N + 1$, the transaction is considered to be valid.

First stage: During the initial registration of an identity, a node provides his pk_{puf}, pk_{gen} and two signatures as defined in Equation 6-4 and Equation 6-5.

$$\sigma_{gen} = sign_{sk_{gen}}(\iota||\tau) \tag{6-4}$$
$$\sigma_{puf} = sign_{sk_{puf}}(\iota||\tau) \tag{6-5}$$

Both signatures prove that the registering node is able to sign messages with the secret key belonging to the public key that this node is adding with his register transaction. Each node in the network can verify that this signature key combination is correct.

Second stage: Once the `register` transaction is placed in the latest block of the blockchain B_m, it is still not officially registered. It now needs to successfully cooperate with nodes that initiate the Identity validation algorithm (Algorithm 2), to be a confirmed transaction in the new to be created block B_{m+1}.

Image 6-1 shows an example of an attack where a user can register an identity that he does not possess. If a miner would be able to add two consecutive blocks to the blockchain, he could first add a malicious transaction and in the following block only add a transaction confirming the authenticity of this malicious transaction.

Because of this attack, we cannot consider the Identity Validation Algorithm secure by default.

We consider the fact that the Identity Validation Algorithm cannot be regarded as secure by default, as an open question in our research. We elaborate on this in our future work chapter (Section 7-2).

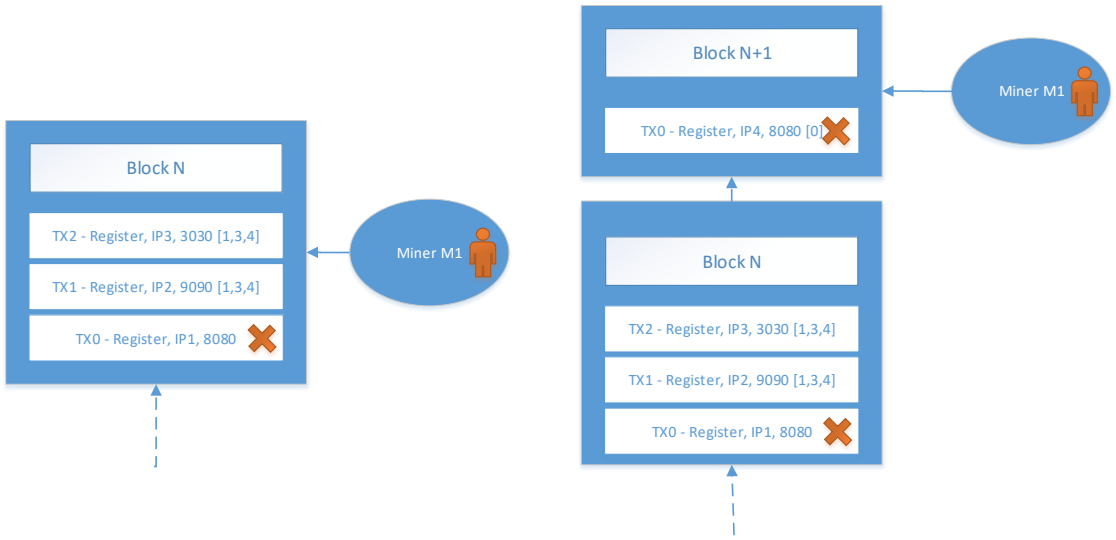

Figure 6-1: A scenario demonstrating how an adversary would be able to submit an invalid registar transaction if it is able to construct two succeeding blocks to the blockchain.

6-2 Performance analysis

To validate the scalability of DECKIN, we created a proof-of-concept implementation of relevant elements of DECKIN. Since we did not focus on the scalability of the blockchain itself, but rather on the what messages are posted on the blockchain and what interaction is done between nodes and miners, we focus on specific elements of DECKIN. We therefore did not implement an arbitrary blockchain to facilitate the underlying requirements to execute our system. We focused on:

1. Cryptographic accumulator - The cryptographic accumulator is used to enable nodes to quickly verify if an elements' identity-key pair is valid in the blockchain, without having to use a large number of resources. We implemented this to verify the scalability of this accumulator.

2. The Validation protocol - The validation protocol that we use requires a relatively large amount of network interaction. We implemented this protocol to be able to investigate what the practical limits of this protocol are.

6-2-1 Implementation details

We built and implemented these elements in the Go programming language (version 1.10) that can be found online on Github*. For the execution of our tests we used Docker† and Docker Compose‡. Docker is used for operating-system-level virtualization. This helps us to mimic the behavior of IoT devices. Docker Compose is a tool that helps defining and running multi-container Docker applications, enabling us to create a testing environment with 40 different nodes. With Docker Compose we were able to limit the resources of each container (to correspond with the performance of IoTs devices).

6-2-2 Accumulator

This section evaluates our implementation of the cryptographic accumulator for its usage DECKIN. To evaluate the scalability of our cryptographic accumulator we created the following tests, for each test we try to analyze what the performance of the nodes would become of 1 billion nodes would work in the network.

1. Time it takes to lookup values - We evaluated the relation between an accumulator size and time to look up if an accumulator correctly possesses an element or not.
2. Time it takes to add values to an accumulator - We evaluated the relation between an the number of elements that an accumulator possesses and the time it takes to add a new element.
3. Accumulator size - We evaluated the relation between the number of elements that are stored in an accumulator and the total accumulator size.

Time to lookup values For the execution of this test, we filled an accumulator with 10.000 values. We then calculated how much time it took to verify if an element was part of the accumulator or not. We did this for 10 runs, and took the average value of these 10 different runs. Image 6-2 shows the the results of this test. The x-axis represents the location of the element in the accumulator. As you can see, apart from some outliers, the time that it takes to verify if an element belongs to an accumulator set is dependent on the location of the element in the accumulator. If it is added as last, it is almost instantaneously verified. If it is added as the first element, after which 9999 other elements are added, the lookup time lies somewhere between 5 and 8 milliseconds. From the image, an linear upper bound and under bound can be identified. Assuming our accumulator would have 1 billion elements, the worst-case validation time would be 8×10^5 milliseconds or 800 seconds.

Time it takes to add values to an accumulator For the execution of this test, we measured the amount of time that it took to add a value to an accumulator. We did this for multiple accumulator sizes. We ran 10 similar tests with 10.000 different accumulator sizes. Image 6-3 shows a graph displaying boxplots visualizing the average time it took for a value to be added to an accumulator. The x-axis is the accumulator size, splitting the values into bins of 2000

*https://www.github.com/mathijshoogland/deckin
†https://www.docker.com/
‡https://docs.docker.com/compose/

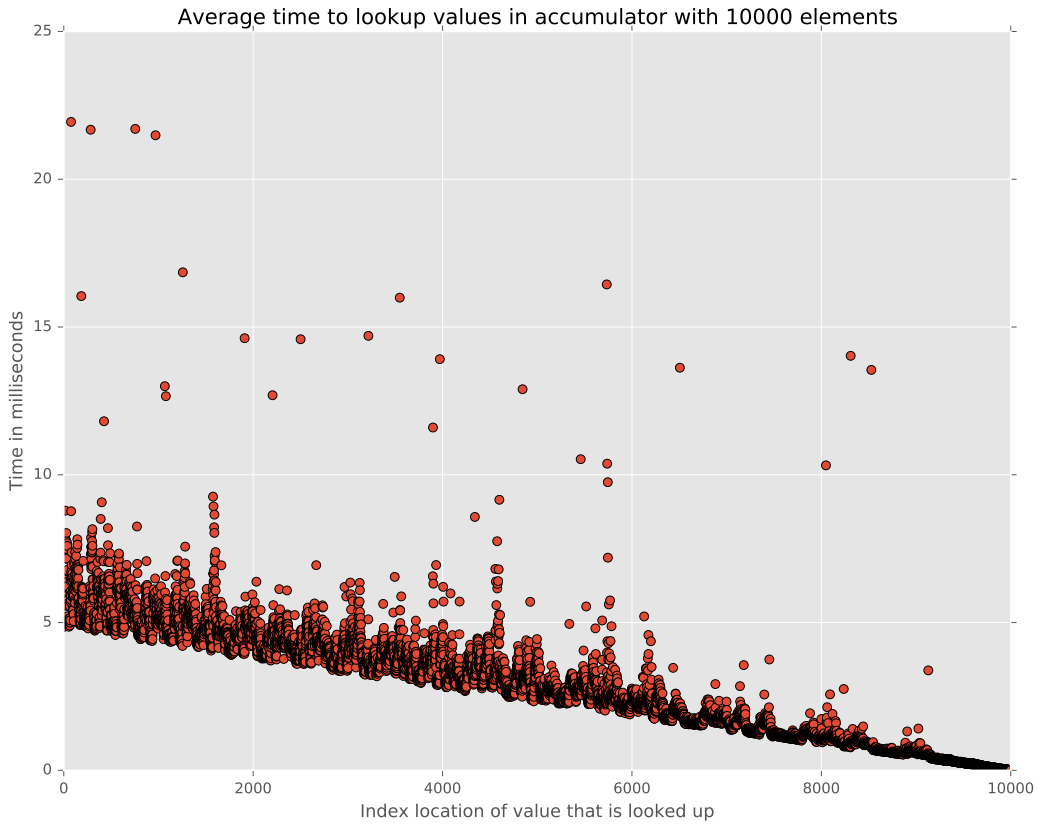

Figure 6-2: Average time it takes to lookup elements in an accumulator with 10000 elements. Average of 10 runs.

elements. As can be seen, the average time it takes to add an element to an accumulator is constant (on average it takes 1.5 microsecond). There are, however, quite a lot of outliers visible at the upper end of all boxplots. This is mainly driven due to system lags, being highlighted because of the extremely short time durations. We can, however, conclude that the amount of time that it takes for an accumulator to add values is not an issue for a network with a billion nodes since the time to add an element can be considered constant.

Accumulator size The last test that we did to evaluate the accumulator scalability is the accumulator size. We tried to calculate the relation between number of elements and the size, and see if this is approximately the same as our theoretical size of $\mathcal{O}(\log(n)^2)$ as calculated in Section 5-6-2.

Image 6-4 shows the size of our accumulator based on the number of elements that the accumulator contains. The x-axis shows the number of elements that the accumulator contains, the y-axis shows the size in bytes. The initial size of our accumulator is 25 bytes. The size of our accumulator with 10000 elements is roughly 330 bytes. This is an increase of 330/25=13,2. The theoretical maximum increase would be $(\log_2(10.000))^2 \approx 176$. This means that our accumulator is in line with the theoretical size. If we would have an accumulator with 1 billion nodes, we would have an accumulator of less than $(\log_2(1000000000))^2 \times 25 \approx 22346$ bytes,

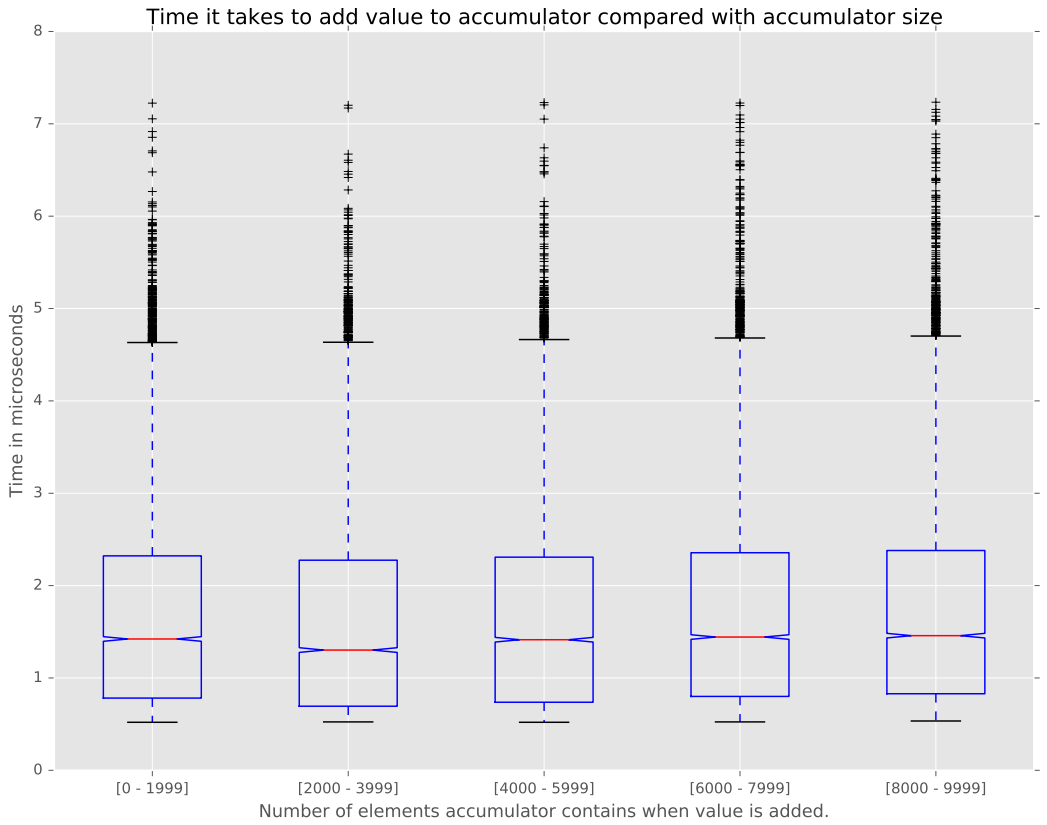

Figure 6-3: The time it takes to add a value to the accumulator compared to number of elements that an accumulator posesses.

what is less than an MB.

6-2-3 Identity Validation Algorithm

In this subsection we evaluate our implementation of the Identity Validation Algorithm. This algorithm is responsible in DECKIN to validate if a node has control over the identity that he is claiming to register. To execute the Identity Validation Algorithm, we implemented the mechanism through Docker enabling us to launch multiple standalone nodes to execute our tests. Docker Compose enabled us to limit the amount of resources that each service would receive. We created the following test-cases.

1. Average duration for varying system resources - In order to map the impact of little resources we executed the protocol on systems with a varying number of resources.
2. Average execution time of 1000 protocol runs - We executed 1000 protocol runs to analyze if protocol execution time is constant or not.
3. Maximum number of http requests that a node can handle - We analyzed what the maximum number of http requests was that our node could handle on systems with a varying number of resources.

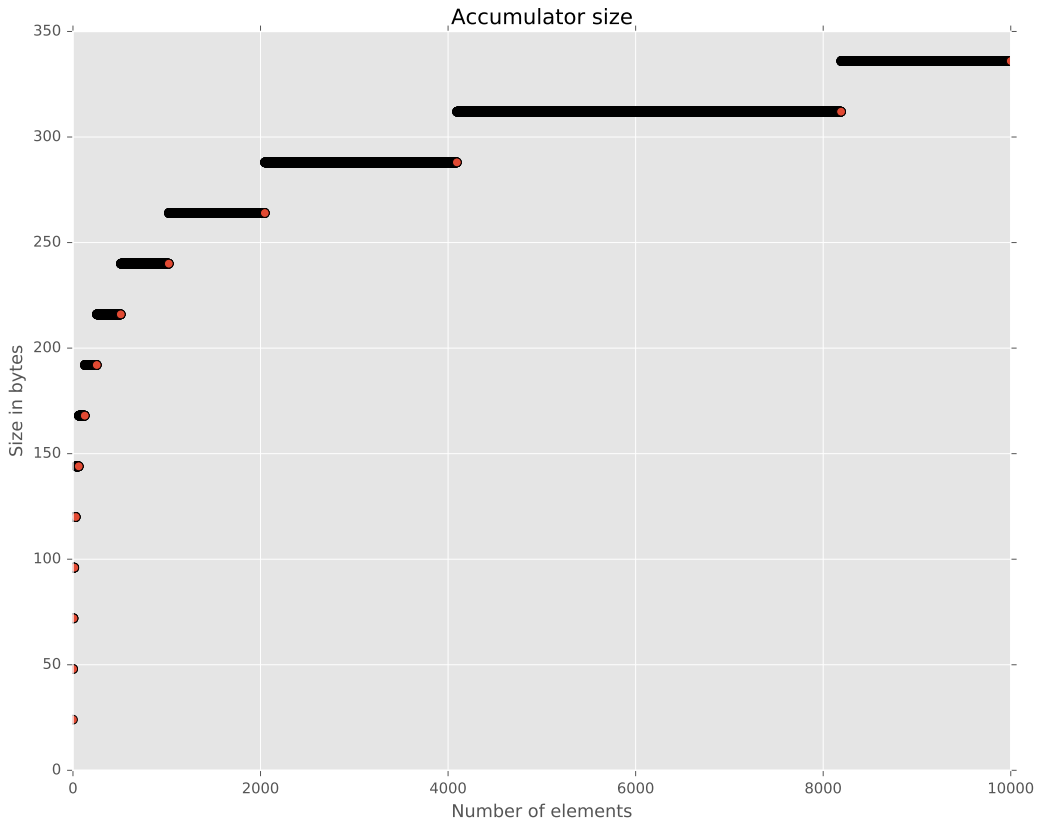

Figure 6-4: Size of our accumulator.

Average duration for varying system resources The total number of calculations that a node needs to do is little, it only needs to hash a transaction and verify if this hash is the same as the returned hash when executing a `GET` query to a node. The main bottleneck for IoT devices when executing our protocol will therefore mainly be contacting all the different network nodes and waiting for responses, that is why we focused on the amount of RAM that each device has and the execution time of our protocol. Image 6-5 shows the average duration of 1000 protocol executions for systems with varying resources. The first system has 4MB RAM, the second system 8MB RAM and the third system 16MB ram. As can be seen, the average duration for a protocol run on the first machine is roughly 35 milliseconds. On the second machine this is 25 milliseconds, and on the third machine it is 12,5 milliseconds. Although there are significant timing differences, we conclude that even with bare minimum of 4MB the execution of our protocol can be done so quick (within a second) that differences are not noticeable in real life.

Average execution time Image 6-6 shows the time it takes for a node with 4MB of RAM to execute the Identity Validation Algorithm for a block containing 40 `register` transactions. For the execution of this test each `register` transaction was valid and therefore, when querying the identity that is being registered, the correct hash value was returned by the 40 different nodes. As you can see, executing times lie between 15 and 130 milliseconds. If should be mentioned that network latency was not simulated in our tests as all the nodes

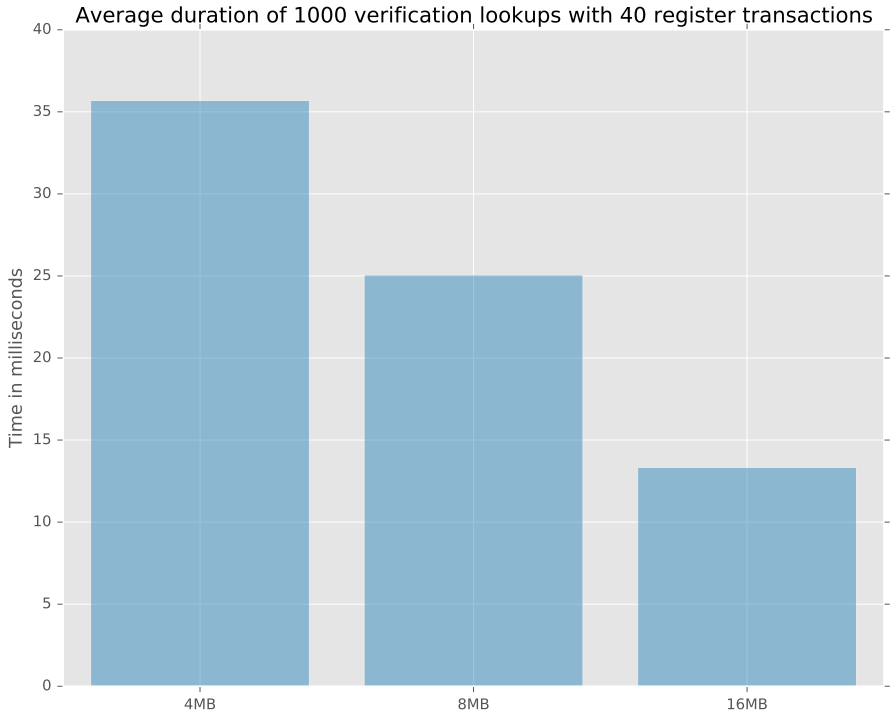

Figure 6-5: The average duration of 1000 verification look ups for machines with different amounts of RAM available.

were on the same Docker network[§].

Maximum number of http requests As seen in Section 6-2-1 the complexity of a node that is stand-bye and being validated by other nodes in the system is dependent of by the number of look ups. For this test, we used the Go library `vegeta`[¶]. This library is an HTTP load testing tool that we used to look-up the maximum number of requests that our nodes could handle. We executed the command as defined in Equation 6-6 for each device 10 times to look-up the maximum number of requests before the system crashed.

```
echo "GET http://localhost:8082/tx" | vegeta attack -duration=5s
-rate=number_of_requests | tee results.bin | vegeta report
```
(6-6)

Table 6-1 shows the average result for each device. Devices with 4MB of RAM could handle 160 requests per second, devices of 8MB could handle 1300 requests per second and devices with 16MB could handle 2000 requests per second. These results clearly show the effect of system resources to the number of http requests that a node with limited resources can serve.

[§]https://docs.docker.com/network/
[¶]https://godoc.org/github.com/tsenart/vegeta/lib

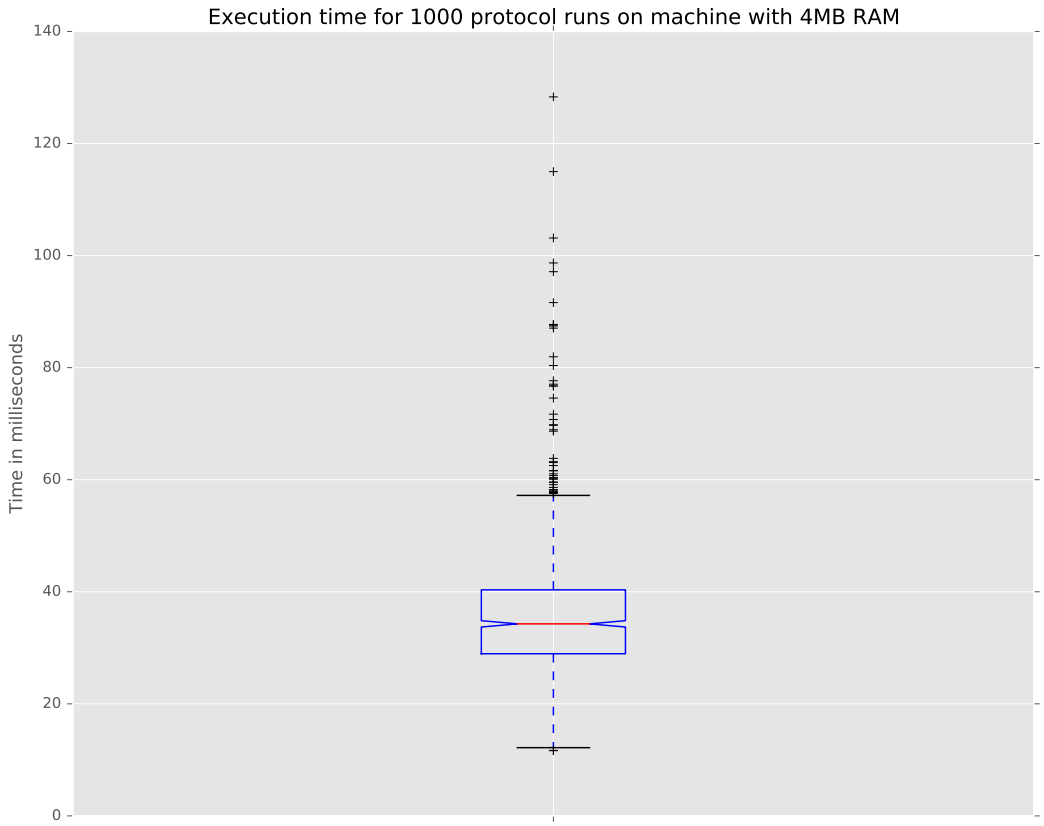

Figure 6-6: Duration of 1000 validation protocol runs on a machine with 4MB of RAM.

6-3 Discussion

In Section 6-1 the security properties of DECKIN are analyzed. It is shown that the `update` and `revoke` transactions are considered secure. This is not by default the case with `register` transactions, it is shown that an attack is possible if a malicious miner is able to add two consecutive blocks, it would then theoretically be able to add a malicious identity. A possible way to mitigate this attack is ensuring that miners are never able to add two consecutive blocks, this constraint could for example be implemented in a variation of the PoW algorithm. Another approach is ensuring that miners are honest by implementing them in a controlled network, this however would make the network more centralized, bringing back some of the downsides that we were trying to solve with our decentralized PKI.

Section 6-2 validated the performance of our implementation of elements of DECKIN. It is shown that most of the operations of the accumulator are well scalable for large networks. Looking up if an accumulator contains elements is however linearly dependent of the location of the element in the accumulator. If an element would be stored in the back of an accumulator with 1 billion nodes, the worst-case lookup time would be 800 seconds. Although this is relatively long, we argue that this is not a large problem. In our use-case we try to facilitate IoT devices that do not have quick communication as a requirement. We argue that it is therefore not a problem when looking up a key for an identity takes some time. We

Table 6-1: The maximum number of http requests per second (average of 10 runs) that nodes could serve. An increment resulted in a server crash.

Amount of RAM	Number of requests
4MB	160 requests per second
8MB	1300 requests per second
16MB	2000 requests per second

also analyzed the time it takes to add a value to an accumulator. It was shown that this operation was well scalable, the time to add an element to an accumulator was almost constant and enables users to add values to accumulators containing millions of elements. It is also shown that our implementation of our accumulator is within the theoretical boundary of $(\log(n)^2)$. We have shown that this does makes accumulators containing a billion elements still small enough to be stored by resource-constrained devices. Our practical analysis of the Identity Validation Algorithm analyzed how constraining the computational resources of devices influenced the execution of the algorithm. It was shown that the computational requirements for nodes that are stand-by are relatively low. The influence of computational resources like RAM memory are measurable, but a low amount of RAM will not form a limitation for our protocol. The largest drawback in our measurements is that we were not able to simulate network latency since for our tests we used a docker network. Another point that needs to be taken into account is the fact that network nodes will receive a lot of traffic when they want to register their identity to the network, although this traffic is only for the duration of a single block, it could be considered undesirable for network administrators because it makes their network cluttered.

Chapter 7

Discussion and future work

It is to be expected that the ever-evolving IoT landscape will change our day-to-day life drastically. IoT devices will be part of basic operations of which 50 years ago it seemed unnecessary to involve complexity in the form of a computer. The security risks that come with this massive adoption are an evolving domain in which scalability and little computational resources are hurdles that need to be addressed when designing security solutions.

Several approaches to increase security by developing a distributed PKI have been suggested in previous work, but none of the previously proposed solutions have tried to develop a decentralized solution that addresses scalability, key management for recovery and revocation and verification of the identity that is to be registered. With DECKIN we demonstrate that development for these types of systems is genuinely promising by focusing on the following research question:

"How can a scalable PKI be designed with practical key recovery, key revocation and identity verification functionality without a single point of failure to improve IoT security?"

This chapter will discuss how our presented protocol has achieved our research goal and, further on, identifies remaining open problems and future research directions.

7-1 Discussion

In this thesis we presented DECKIN, a distributed PKI that adds a validation layer. This layer is executed when nodes want to register an identity to the blockchain. This Identity Validation Layer tries to answer the first sub-question of our research question:

Research subquestion 1: "How can an IoT device identifier and key pair be verified without a central party?"

A significant challenge that arises when trying to answer this question is public verifiability. The power of current blockchain solutions is the fact that every node can verify each computation. Verifying if a certain node has access to the identifier that he is trying to possess is more difficult. Two directions can be taken:

1. Either the node will have to participate in interactive challenge-response schemes throughout the lifetime of the identity registration, which is, in our opinion, undesirable because this would introduce large amounts of network traffic without a clear benefit.
2. Or the public verifiability requirement needs to be loosened, leaving the verifiability to a pre-determined set of nodes that collectively can be trusted.

In DECKIN we choose for the second option with our Identity Validation Algorithm. Nodes that want to add an element to the blockchain will first need to verify all the `register` transactions from the latest block. If they do not correctly validate these transactions, their transaction will not be included by the miner. As discussed in Section 5-4, this solution lacks security if a malicious miner would be able to add two consecutive blocks. We consider this however as an open problem that could be solved by adding specific constraints to the underlying blockchain. Another direction that could be considered when trying to solve this problem is adding centralized verification nodes, we argue however that a full decentralized system would be more preferable since this removes single points of failure.

DECKIN requires users to register two different keys during the initial registration of their identity, a *general* key that is used for future interaction between nodes and a *puf* key that is used for revoking and key recovery. This construction with PUF technology tries to answer our second and third sub-question from our research:

Research subquestion 2: "How can practical key recovery be implemented without a central party?"
Research subquestion 3: "How can key revocation be implemented?"

By using PUF technology, users are able to rely on their hardware fingerprint and an authentication code that is submitted during the initial registration of an identity. Whenever a key is lost, it will be able to be recovered. We consider this solution superior compared to solutions that were proposed and discussed in the Prior Art Section (3-2). A downside of the technique is that is has certain device specific requirements. The system that we for example suggested, the Intrinsic ID BROADKEY solution *, is not open source and is therefore not publicly available for everybody at this time. The system also requires devices to shield a part of their SRAM cells to prevent other applications to access these cells and read their initial value. We reason however that the hybrid setup of our system, enabling nodes without a PUF key to participate, does make this downside acceptable. In DECKIN, `revoke` transactions enable the owners of a secret key to revoke the identity and key combination submitted to the system. Users need to use their *puf* key in order to submit a `revoke` transaction. This functionality is not comparable to a centralized PKI system where an authority revokes the keys that are assumed to be malicious, in DECKIN, systems are powerless if an adversary has

*https://www.intrinsic-id.com/products/broadkey/

correctly registered an identity that he uses for malicious practices. If a secret key would be stolen by an adversary after it was registered on the DECKIN blockchain, the original node would be able to submit a `revoke` transaction with his pk_{puf} and disable the ability of the adversary to misuse this stolen key.

The last distinctive feature of DECKIN is the usage of cryptographic accumulators to enable nodes with little computational power to be able to verify if certain identity-key combinations or rightfully registered in the DECKIN system. This feature tries to answer the fourth sub-question from our research:

Research subquestion 4: "How can this system be designed in a well scaling manner to facilitate PKI functionality for millions of devices?"

In the evaluation of DECKIN in Chapter 6 we put a lot of focus on the scalability of our protocol. We have shown that the usage of accumulators enables nodes with little computational resources to lookup the validity of key-value pairs in our system. The scalability of our system however is also largely depending on the underlying blockchain protocol. Although we have analyzed the scalability of all elements of our protocol, this can only be fully answered if a complete system is deployed and evaluated. The accumulator that we use is also not compact, meaning that the size increases as the number of elements increases.

We argue that, by reviewing our sub-questions and discussing the possible downsides of each proposed solution, we have demonstrated a promising way to design an efficient PKI with functional key recovery, key revocation and verification services without a single point of failure to improve IoT security.

7-2 Future work

Our presented protocol DECKIN is, to the best of our knowledge, the first distributed PKI that provides a framework for devices with low computational resources where to-be registered identities are verified and where key management is simplified with the help of PUF technology. However, there is still a lot to be done before similar systems would be deployed in real life. This section gives an overview of further research to extend this work or to explore the implementation in other sub-systems.

Model malicious behavior Modelling malicious behavior is one of the domains that should be worked out more extensively for our distributed PKI to be widely deployed. The Identity Validation Algorithm that we presented is susceptible for adversaries if a malicious miner would be able to append two consecutive blocks to blockchain. It should be noted that in general, based on our previous work study, we can say that modeling malicious behavior is not yet a priority for most research. Most research is done on working out proof of concepts. In order for distributed PKIs to present a worthy security solution, more research needs to be done on the different attacker models.

Implement underlying blockchain For DECKIN we choose to focus on a protocol that works on top of a functional blockchain system. Because blockchain scalability is a research topic on itself, we reasoned that our research would else lie too much on a subset of elements where we would not be able to go too deep into detail. It is, however, a topic that should be looked into further. If this research would evolve and DECKIN would be fully implemented with an underlying Blockchain, scalability could be evaluated in a more satisfactory manner.

Research IoT requirements The IoT landscape is Our system interacts with IoT devices that want to register an identity by requiring them to be reachable for the duration of the addition of one block. During this period, nodes try to verify if the `register` transaction that is submitted is valid and if the identity is really in the possession of the registering node. We have shown that the computational requirements to respond to these requests is very low, and in general it should not be considered a reason for IoT devices to not join in the DECKIN network. There is, however, not a lot of research done on the requirements of IoT devices. It is for example not known if many network administrators would consider it to be a burden if their devices would have to be connect reachable by a large number of devices. Dedicating more research on what the requirements of an average node is in an IoT network is would help adoption in the future.

Embedding PUF technology in PKI protocols In DECKIN we propose the usage of PUF technology to help with our key-management challenges. More research should be done on the practical implementation and the validation of this technology, since our research mainly focuses on the theoretical feasibility.

Appendix A

Accumulator maintenance operations

This chapter displays all operations that need to be executed when an identity is updated, revoked or registered for accumulator maintenance.

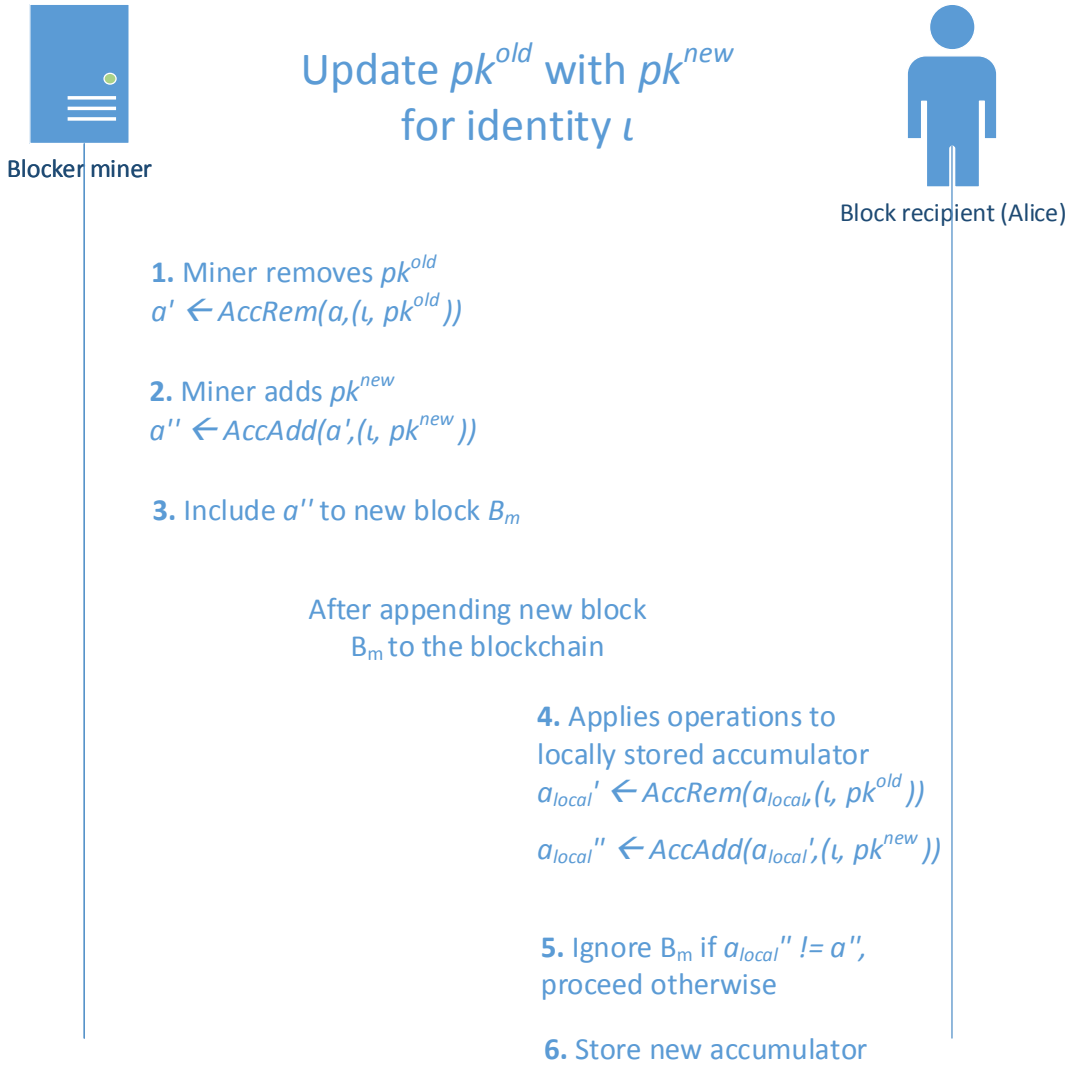

Figure A-1: An example of the operations that need to be executed for accumulator maintenance when a key pk^{old} is updated with pk^{new} for identity ι.

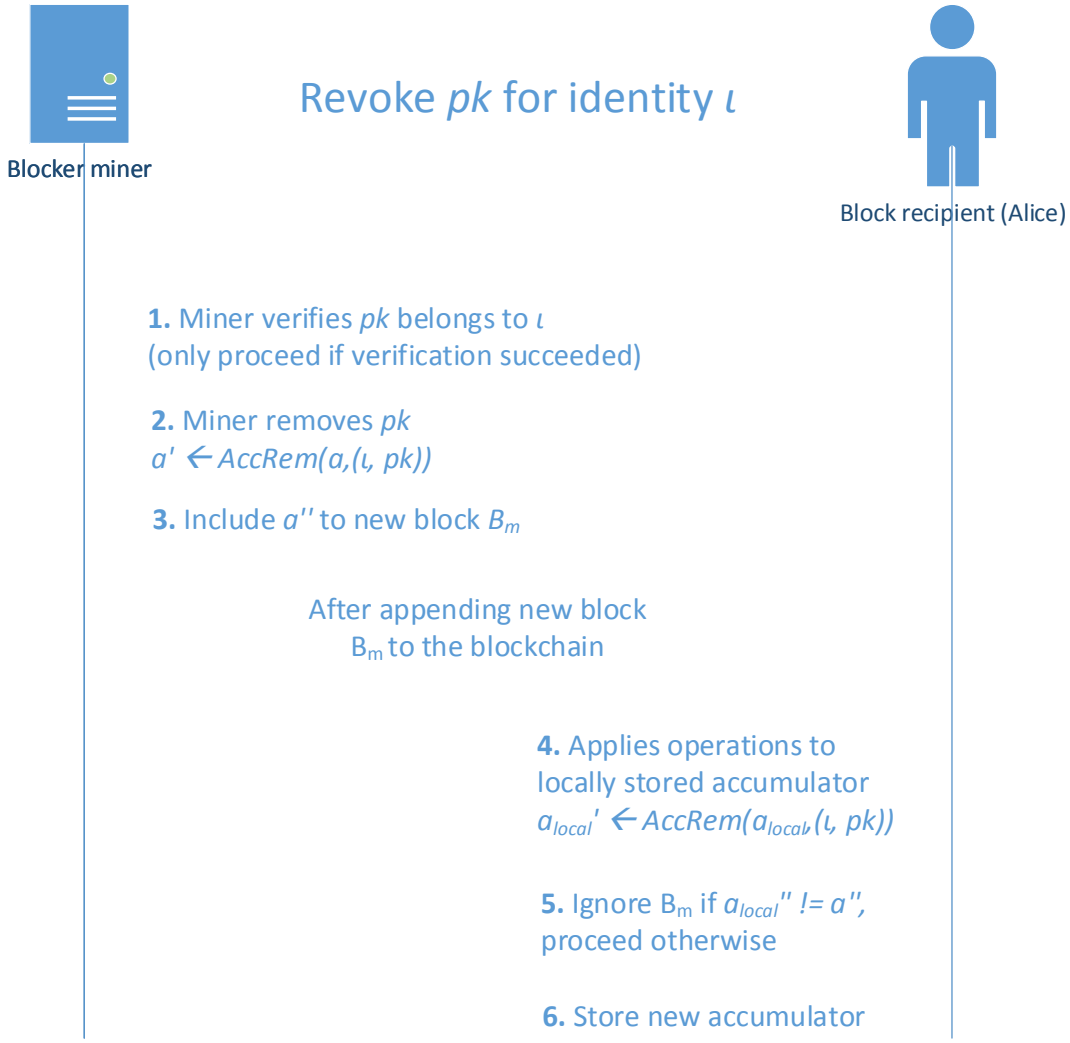

Figure A-2: An example of the operations that need to be executed for accumulator maintenance when a key pk is revoked for identity ι.

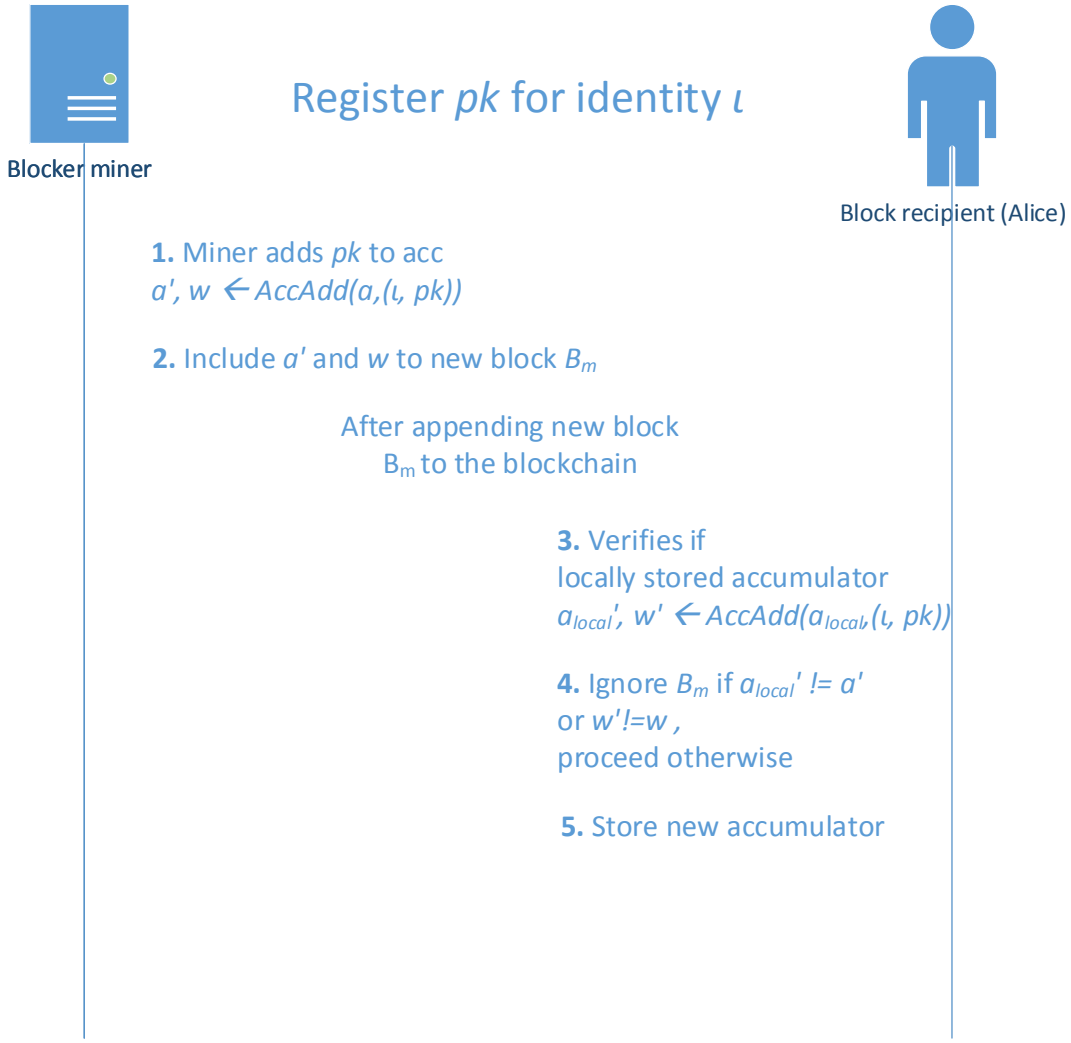

Figure A-3: An example of the operations that need to be executed for accumulator maintenance when a key pk is registered for identity ι.

Bibliography

[1] H. Koçak, S. Dündar, and H. Arslan, "A comparative survey on wireless Internet usage and future expectations of university students-the cases of USF, AKU and IUS," *Epiphany*, vol. 8, no. 1, pp. 113–132, 2015.

[2] F. Wortmann and K. Flüchter, "Internet of things - technology and value added," *Business & Information Systems Engineering*, vol. 57, no. 3, pp. 221–224, 2015. [Online]. Available: https://doi.org/10.1007/s12599-015-0383-3

[3] R. van der Meulen, "Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016," 2017. [Online]. Available: https://www.gartner.com/newsroom/id/3598917

[4] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Computer Networks*, vol. 76, pp. 146–164, 2015. [Online]. Available: https://doi.org/10.1016/j.comnet.2014.11.008

[5] E. Bertino and N. Islam, "Botnets and Internet of Things Security," *IEEE Computer*, vol. 50, no. 2, pp. 76–79, 2017. [Online]. Available: https://doi.org/10.1109/MC.2017.62

[6] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, "Security of the Internet of Things: perspectives and challenges," *Wireless Networks*, vol. 20, no. 8, pp. 2481–2501, 2014. [Online]. Available: https://doi.org/10.1007/s11276-014-0761-7

[7] Pedro Abreu, "Why manufacturers make insecure IoT devices and how you can protect them - IoT Agenda," 2017. [Online]. Available: http://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/Why-manufacturers-make-insecure-IoT-devices-and-how-you-can-protect-them

[8] C. Kolias, G. Kambourakis, A. Stavrou, and J. M. Voas, "DDoS in the IoT: Mirai and Other Botnets," *IEEE Computer*, vol. 50, no. 7, pp. 80–84, 2017. [Online]. Available: https://doi.org/10.1109/MC.2017.201

[9] B. Krebs, "DDoS on Dyn Impacts Twitter, Spotify, Reddit," 2016. [Online]. Available: https://krebsonsecurity.com/2016/10/ddos-on-dyn-impacts-twitter-spotify-reddit/

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

[11] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf," 2008.

[12] N. Kshetri, "Can Blockchain Strengthen the Internet of Things?" *IT Professional*, vol. 19, no. 4, pp. 68–72, 2017. [Online]. Available: https://doi.org/10.1109/MITP.2017.3051335

[13] S. Azouvi, M. Al-Bassam, and S. Meiklejohn, "Who Am I? Secure Identity Registration on Distributed Ledgers," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2017 International Workshops, DPM 2017 and CBT 2017, Oslo, Norway, September 14-15, 2017, Proceedings*, 2017, pp. 373–389. [Online]. Available: https://doi.org/10.1007/978-3-319-67816-0_21

[14] L. Axon and M. Goldsmith, "PB-PKI: A privacy-aware blockchain-based PKI," in *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT, Madrid, Spain, July 24-26, 2017.*, 2017, pp. 311–318. [Online]. Available: https://doi.org/10.5220/0006419203110318

[15] A. Dorri, S. S. Kanhere, and R. Jurdak, "Blockchain in internet of things: Challenges and Solutions," *CoRR*, vol. abs/1608.05187, 2016. [Online]. Available: http://arxiv.org/abs/1608.05187

[16] N. Vatra, "Public key infrastructure overview," in *Communications (COMM), 2010 8th International Conference on*. IEEE, 2010, pp. 481–484.

[17] J. Wolff, "How the 2011 hack of DigiNotar changed the internet's infrastructure." 2016. [Online]. Available: http://www.slate.com/articles/technology/future_tense/2016/12/how_the_2011_hack_of_diginotar_changed_the_internet_s_infrastructure.html

[18] M. Björkqvist, C. Cachin, R. Haas, X. Hu, A. Kurmus, R. Pawlitzek, and M. Vukolic, "Design and implementation of a key-lifecycle management system," in *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers*, 2010, pp. 160–174. [Online]. Available: https://doi.org/10.1007/978-3-642-14577-3_14

[19] H. Delfs and H. Knebl, *Introduction to Cryptography - Principles and Applications, Third Edition*, ser. Information Security and Cryptography. Springer, 2015. [Online]. Available: https://doi.org/10.1007/978-3-662-47974-2

[20] N. P. Smart, *Cryptography Made Simple*, ser. Information Security and Cryptography. Springer, 2016. [Online]. Available: https://doi.org/10.1007/978-3-319-21936-3

[21] NIST FIPS, "FIPS-197: Advanced Encryption Standard (AES)," *Fed. Inf. Process. Stand. Publ.*, p. p. 311, 2001.

[22] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, 1987, pp. 369–378. [Online]. Available: https://doi.org/10.1007/3-540-48184-2_32

[23] C. Coronado, "On the security and the efficiency of the Merkle signature scheme," *IACR Cryptology ePrint Archive*, vol. 2005, p. 192, 2005. [Online]. Available: http://eprint.iacr.org/2005/192

[24] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, 1989. [Online]. Available: https://doi.org/10.1137/0218012

[25] C. Fromknecht, D. Velicanu, and S. Yakoubov, "A decentralized public key infrastructure with identity retention," *IACR Cryptology ePrint Archive*, vol. 2014, p. 803, 2014. [Online]. Available: http://eprint.iacr.org/2014/803

[26] J. C. Benaloh and M. de Mare, "One-way accumulators: A decentralized alternative to digital sinatures (extended abstract)," in *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, ser. Lecture Notes in Computer Science, T. Helleseth, Ed., vol. 765. Springer, 1993, pp. 274–285. [Online]. Available: https://doi.org/10.1007/3-540-48285-7_24

[27] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Annual International Cryptology Conference*. Springer, 2002, pp. 61–76.

[28] J. Li, N. Li, and R. Xue, "Universal accumulators with efficient nonmembership proofs," in *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*, 2007, pp. 253–269. [Online]. Available: https://doi.org/10.1007/978-3-540-72738-5_17

[29] P. Camacho, A. Hevia, M. A. Kiwi, and R. Opazo, "Strong accumulators from collision-resistant hashing," in *Information Security, 11th International Conference, ISC 2008, Taipei, Taiwan, September 15-18, 2008. Proceedings*, ser. Lecture Notes in Computer Science, T. Wu, C. Lei, V. Rijmen, and D. Lee, Eds., vol. 5222. Springer, 2008, pp. 471–486. [Online]. Available: https://doi.org/10.1007/978-3-540-85886-7_32

[30] J. Camenisch, M. Kohlweiss, and C. Soriente, "An accumulator based on bilinear maps and efficient revocation for anonymous credentials," in *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*, ser. Lecture Notes in Computer Science, S. Jarecki and G. Tsudik, Eds., vol. 5443. Springer, 2009, pp. 481–500. [Online]. Available: https://doi.org/10.1007/978-3-642-00468-1_27

[31] L. Reyzin and S. Yakoubov, "Efficient asynchronous accumulators for distributed PKI," in *Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016, Proceedings*, ser. Lecture Notes in Computer Science, V. Zikas and R. D. Prisco, Eds., vol. 9841. Springer, 2016, pp. 292–309. [Online]. Available: https://doi.org/10.1007/978-3-319-44618-9_16

[32] A. Baliga, "Understanding blockchain consensus models," Tech. rep., Persistent Systems Ltd, Tech. Rep, Tech. Rep., 2017.

[33] A. de Vries, "Bitcoin's growing energy problem," *Joule*, vol. 2, no. 5, pp. 801–805, 2018.

[34] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]y," *SIGMETRICS Performance Evaluation Review*, vol. 42, no. 3, pp. 34–37, 2014. [Online]. Available: http://doi.acm.org/10.1145/2695533.2695545

[35] A. Beikverdi and J. Song, "Trend of centralization in bitcoin's distributed network," in *16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2015, Takamatsu, Japan, June 1-3, 2015*. IEEE Computer Society, 2015, pp. 377–382. [Online]. Available: https://doi.org/10.1109/SNPD.2015.7176229

[36] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds., vol. 9057. Springer, 2015, pp. 281–310. [Online]. Available: https://doi.org/10.1007/978-3-662-46803-6_10

[37] J. J. Roberts, "Bitcoin Spinoff Hacked in Rare 51% attack," 2018. [Online]. Available: http://fortune.com/2018/05/29/bitcoin-gold-hack/

[38] A. Norry, "The History of the Mt Gox Hack: Bitcoin's Biggest Heist," 2017. [Online]. Available: https://blockonomi.com/mt-gox-hack/

[39] J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow, "Elliptic curve cryptography in practice," in *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, ser. Lecture Notes in Computer Science, N. Christin and R. Safavi-Naini, Eds., vol. 8437. Springer, 2014, pp. 157–175. [Online]. Available: https://doi.org/10.1007/978-3-662-45472-5_11

[40] C. Decker, "On the scalability and security of bitcoin," Ph.D. dissertation, ETH Zurich, 2016.

[41] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00, 2018, pp. 19–34. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SP.2018.000-5

[42] imbrex, "Sharding, Raiden, Plasma: The Scaling Solutions that Will Unchan Ethereum," 2017. [Online]. Available: https://medium.com/imbrexblog/sharding-raiden-plasma-the-scaling-solutions-that-will-unchain-ethereum-c590e994523b

[43] V. Buterin, "Plasma: Scalable Autonomous Smart Contracts," 2017. [Online]. Available: https://plasma.io

[44] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[45] R. Maes, *Physically unclonable functions: Constructions, properties and applications*. Springer Science & Business Media, 2013.

[46] A.-R. Sadeghi and D. Naccache, *Towards hardware-intrinsic security*. Springer, 2010.

[47] C. H. Chang, Y. Zheng, and L. Zhang, "A retrospective and a look forward: Fifteen years of physical unclonable function advancement," *IEEE Circuits and Systems Magazine*, vol. 17, no. 3, pp. 32–62, thirdquarter 2017.

[48] G. N. Selimis, M. Konijnenburg, M. Ashouei, J. Huisken, H. de Groot, V. van der Leest, G. J. Schrijen, M. van Hulst, and P. Tuyls, "Evaluation of 90nm 6t-sram as physical unclonable function for secure key generation in wireless sensor nodes," in *International Symposium on Circuits and Systems (ISCAS 2011), May 15-19 2011, Rio de Janeiro, Brazil*. IEEE, 2011, pp. 567–570. [Online]. Available: https://doi.org/10.1109/ISCAS.2011.5937628

[49] R. Maes, V. van der Leest, E. van der Sluis, and F. M. J. Willems, "Secure key generation from biased pufs," *IACR Cryptology ePrint Archive*, vol. 2015, p. 583, 2015. [Online]. Available: http://eprint.iacr.org/2015/583

[50] D. Tapscott and A. Tapscott, *Blockchain Revolution*. Brilliance Audio, 2016.

[51] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of namecoin and lessons for decentralized namespace design," in *14th Annual Workshop on the Economics of Information Security, WEIS 2015, Delft, The Netherlands, 22-23 June, 2015*, 2015. [Online]. Available: http://www.econinfosec.org/archive/weis2015/papers/WEIS_2015_kalodner.pdf

[52] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, 2002, pp. 53–65. [Online]. Available: https://doi.org/10.1007/3-540-45748-8_5

[53] Bitcoinarmory.com, "Armory FAQ," 2018. [Online]. Available: https://www.bitcoinarmory.com/faq/

[54] M. Ali, J. C. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016.*, 2016, pp. 181–194. [Online]. Available: https://www.usenix.org/conference/atc16/technical-sessions/presentation/ali

[55] Blockstag.org, "Blockstack DNS vs. Namecoin," 2018. [Online]. Available: https://blockstack.org/docs/blockstack-vs-namecoin

[56] J. Li and D. Mazières, "Beyond one-third faulty replicas in byzantine fault tolerant systems," in *4th Symposium on Networked Systems Design and Implementation (NSDI 2007), April 11-13, 2007, Cambridge, Massachusetts, USA, Proceedings.*, 2007. [Online]. Available: http://www.usenix.org/events/nsdi07/tech/li.html

[57] D. Mazières and D. Shasha, "Building secure file systems out of byzantine storage," in *Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing*, ser. PODC '02. New York, NY, USA: ACM, 2002, pp. 108–117. [Online]. Available: http://doi.acm.org/10.1145/571825.571840

[58] M. Al-Bassam, "Scpki: A smart contract-based pki and identity system," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, ser. BCC '17. New York, NY, USA: ACM, 2017, pp. 35–40. [Online]. Available: http://doi.acm.org/10.1145/3055518.3055530

[59] B. Leiding, C. H. Cap, T. Mundt, and S. Rashidibajgan, "Authcoin: Validation and authentication in decentralized networks," in *10th Mediterranean Conference on Information Systems, MCIS 2016, Paphos, Cyprus, 4-6 September 2016*, 2016, p. 5. [Online]. Available: http://aisel.aisnet.org/mcis2016/5

[60] Letsencrypt.org, "Let's Encrypt - How it works," 2018. [Online]. Available: https://letsencrypt.org/how-it-works/

[61] B. Laurie, A. Langley, and E. Käsper, "Certificate transparency," *RFC*, vol. 6962, pp. 1–27, 2013. [Online]. Available: https://doi.org/10.17487/RFC6962

[62] R. Sleevi, "Certificate Transparency in Chrome - Change to Enforcement Date - Google Discussion Groups," 2018. [Online]. Available: https://groups.google.com/a/chromium.org/forum/{#}!topic/ct-policy/sz{_}3W{_}xKBNY

[63] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q.*, vol. 28, no. 1, pp. 75–105, Mar. 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=2017212.2017217

[64] J. Herrera-Joancomartí and C. Pérez-Solà, "Privacy in Bitcoin Transactions: New Challenges from Blockchain Scalability Solutions," in *Modeling Decisions for Artificial Intelligence - 13th International Conference, MDAI 2016, Sant Julià de Lòria, Andorra, September 19-21, 2016. Proceedings*, ser. Lecture Notes in Computer Science, V. Torra, Y. Narukawa, G. Navarro-Arribas, and C. Yañez, Eds., vol. 9880. Springer, 2016, pp. 26–44. [Online]. Available: https://doi.org/10.1007/978-3-319-45656-0_3

[65] P. Gallagher, D. D. Foreword, and C. F. Director, "Fips pub 186-3 federal information processing standards publication digital signature standard (dss)," June 2009, u.S.Department of Commerce/National Institute of Standards and Technology.

[66] "Fips pub 180-4, secure hash standard (shs)," 2001, u.S.Department of Commerce/National Institute of Standards and Technology.

[67] M. Fersch, E. Kiltz, and B. Poettering, "On the provable security of (EC)DSA signatures," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1651–1662. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978413

Acronyms

AES Advanced Encryption Standard. 10

BNS Blockchain Name System. 27

CA Certificate Authority. 3, 4, 5

CRL Certificate Revocation List. 32

CRP challenge-response-pair. 19

CT Certificate Transparency. 4, 32

DDoS Distributed denial-of-service. 3

DECKIN DEcentralized Key INfrastructure. iii, xi, 6, 9, 10, 11, 35, 36, 37, 39, 48, 50, 51, 52, 53, 59, 60, 63, 64, 65, 66

DHT Distributed Hash Table. 14, 25, 27, 39

DNS Domain Name System. 23, 24, 27, 31

ECC Elliptic Curve Cryptography. 12, 40

ECDSA Elliptic Curve Digital Signature Algorithm. 12, 51, 52

IoT Internet of Things. iii, 1, 2, 3, 5, 6, 26, 27, 31, 33, 35, 36, 39, 40, 54, 57, 60, 63, 65, 66

IPFS Inter Planetary File System. 29

PB-PKI Privacy-Aware Blockchain-Based PKI. ix, 26, 31

PKI Public Key Infrastructure. iii, 3, 4, 5, 6, 9, 23, 24, 25, 26, 27, 29, 31, 33, 35, 36, 39, 45, 47, 48, 51, 59, 63, 64, 65, 66

PoS Proof of Stake. 16

PoW Proof of Work. 15, 16, 59

PUF Physical Unclonable Function. iii, 6, 9, 18, 19, 46, 47, 48, 64, 65, 66

RA Registration Authority. 4

RT Revocation Transparency. 32

SCPKI Smart Contract-based PKI and Identity System. 29, 31, 34

VA Validation Authority. 4

WoT Web of Trust. 3, 4, 5, 29

List of symbols

List of Symbols

ι	Identifier in DECKIN, this is always an IP address.
ρ	The payload of a DECKIN transaction.
τ	Array containing all the indices of all valid register transactions in the latest added block to the blockchain.
θ	Transaction type for DECKIN, this is always `register`, `update` or `revoke`
$l_{i,j}$	Number of validate lookups for transaction i in block j
m_i	Number of transactions in block i
$m_j^{register}$	Number of `register` transactions in block j
n	Number of registered identities in DECKIN