DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Security of dynamic authorisation for IoT through Blockchain technology

ALEXANDER SANDOR

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Security of dynamic authorisation for IoT through Blockchain technology

ALEXANDER SANDOR

Master in Computer Science
Date: June 17, 2018
Supervisor: Roberto Guanciale
Examiner: Mads Dam
Swedish title: Säkerheten av dynamisk autentisering för IoT genom Blockchain-teknik
School of Computer Science and Communication

Abstract

The use of Internet of Things devices is an integral part of our modern society. Communication with internet of things devices is secured with asymmetric key encryption that is handled by the centralized certificate authority infrastructure. The emerging Blockchain technology now provides a safe way to change ownership of digital resources through a decentralized system that challenges the traditional centralized view of trust in digital systems. This project studies the security of building public key infrastructures and access communication protocols on Blockchain technology for IoT devices. An informal cryptographic analysis that used proof by contradiction showed that it is cryptographically safe to build Blockchain based Public Key Infrastructures. The analysed Blockchain based public key infrastructure was implemented with smart contracts and tested on the Ethereum platform along with a dynamic access control protocol ensuring dynamic authentication and distributed logging. The project also concluded that advancements in the software clients of nodes are required before Blockchain can be used in Internet of Things devices. This is due to the high storage demands required by currently available nodes.

Sammanfattning

Användandet av ”Internet of Things”-enheter är en integral del av vårt moderna samhälle. Kommunikation med ”Internet of Things”-enheter är säkras genom asymmetrisk nyckelkryptering som hanteras i ett centraliserat system administrerat av certifieringsmyndigheter. Den banbrytande Blockchain-tekniken erbjuder nu ett säkert sätt att byta ägandeskap av digitala resurser i ett decentraliserat system, och utmanar den traditionella synen på tillit i digitala system. Det här projektet studerar säkerheten i att bygga en infrastruktur för publik nyckeldistribuering samt protokoll för accesskontrollering med hjälp av Blockchain-teknik för ”Internet of Things”-enheter. Genom en informell kryptografisk analys och metoden motsägelsebevis visades det att det är kryptografiskt säkert att bygga infrastrukturer för publik nyckeldistribuering på Blockchain-teknik. En Blockchain-baserad infrastruktur för public nyckeldistribuering implementerades med smarta kontrakt och testades på Ethereum-plattformen tillsammans med ett protokoll för dynamisk accesskontroll som säkerställde dynamisk autentisering och distribuerad loggning. Projektet kom även fram till att ny mjukvara för noder behövs för att tekniken ska bli applicerbar i ”Internet of Things”-enheter. Detta eftersom nuvarande noder behöver stort datautrymme för att fungera.

Acknowledgements

Special thanks to my outstanding supervisors Roberto Guanciale at KTH and Petter Lagusson at Tritech. Your feedback, patience and support were invaluable for the outcome of the project.

Contents

List of Figures

List of Tables

Acronyms

The following list contains the central acronyms used in this thesis along with the page of first occurrence.

CA Certificate Authority. 2

CPA Cryptographic Protocol Analysis. 22

DAP Dynamic Access Protocol. 44

DLP Discrete Logarithm Problem. 9

DoS Denial of Service. 6

DSA Digital Signature Algorithm. 12

ECC Elliptic Curve Cryptography. 10

ECDLP Elliptic Curve Discrete Logarithm Problem. 11

ECDSA Elliptic Curve Digital Signature Algorithm. 11

EVM Ethereum Virtual Machine. 18

IoT Internet of Things. 1

MAC Media Access Control. 8

PG1 Project Goal 1. 5

PG2 Project Goal 2. 5

PKI Public Key Infrastructure. 1

RA Registration Authority. 2

WoT Web of Trust. 2

Chapter 1

Introduction

The Internet of Things (IoT) is commonly referred to as embedded electronic devices that have the ability to communicate over a network interface. While some devices can communicate directly to the Internet, others have to relay their traffic through a local hub. The primary goals of having network interfaces on embedded devices are to enable remote management, information gathering and software updates. IoT is an integral part of the modern society and enables remote access to the embedded devices that automate our infrastructure. Hydropower plants, indoor climate control systems and automated cars, to name a few examples, all depend on embedded devices with network interfaces. In order for the safe use of embedded devices with network interfaces, access and communication with them need to be secure.

Public key encryption enables safe communication given access to the public key of the intended recipient. In an ideal world, two parties would share their public keys directly with each other under a guarantee that their keys are not tampered with. This is not the case when a new connection is established through a shared network. Through a shared network an adversary might intercept and change non-authenticated or encrypted messages. Since the two parties might have never communicated before, this leaves them no way of establishing a secure connection [40]. Aside from manually distributing keys to the devices, the use of a Public Key Infrastructure (PKI) is a solution to this problem.

1.1 Public Key Infrastructure

The purpose of a PKI is to distribute and validate the authenticity and integrity of public keys. In a PKI some form of central authority is established that ensures trust for the public keys in the network. The central authority is then used to verify the integrity and authenticity of the public keys. There are two models traditionally used when constructing the trusted central authority: Certificate Authority (CA) and Web of Trust (WoT) [25].

1.1.1 Certificate Authorities

The CA model uses multiple parts to ensure trust for the public keys of the network. New users in the network submit their public keys for registration to a Registration Authority (RA). The RA verifies the identity of the user and its public key. After approval from the RA a CA issues a digital certificate ensuring the connection between the user and the public key. The certificate includes among other things the identity of the user and the public key and is signed with the certificate authorities private key. This makes the certificate verifiable for any user holding the CA public key. The certificates are also stored in a secure central directory that can be queried for certificates of other users. In order for the CA method to be secure two assumptions need to be true:

1. The public key of the CA is obtained in a secure way [25].
2. The administration of the CA is trusted [14].

1.1.2 Web of Trust

In the WoT model, users of the network themselves sign other users' public keys ensuring the authenticity of the connection between a user and a public key. A user can then choose to trust other users signed public keys in order to grow its sphere of trusted public keys. This creates multiple WoTs where the trusted users together act as the central authority that verifies public keys. Compared to the CA method this is a decentralized model that does not need trust to be placed under a central administration [25].

1.1.3 Challenges with current PKI structures

With a PKI structure based on the CA model, a single point of trust is created in the network. This forces the users in the network to place trust in the administration of the CA. A single point of failure is also created where a compromise of the CA would result in a complete compromise of the network [14].

The WoT model mitigates the single point of failure by distributing trust in the network and successfully removes the need for a trusted central administration. The WoT instead has a significant vulnerability in its spread of trust in the network as each user becomes a possible point of attack. The WoT model also makes it difficult for new users to join the network as this typically involves distributing their keys through another secure way, for example meeting in person.

To solve both of the inherent problems of traditional PKI models, new solutions based on the Blockchain technology have emerged [22].

1.2 Blockchain

Blockchain is a technology that enables permanent and verifiable transfer of ownership for digital resources in a decentralized system. It is built upon a distributed ledger that records all transactions between parties in the system. The ledger consists of blocks typically containing transaction data, a hash pointer to the previous block that is secure by cryptography and a time stamp. The transactions are validated, and new blocks are calculated distributed through a peer to peer network which makes it difficult to manipulate the chain. Blockchains are supposedly secure by design and solve the problem of double spending without the need of a trusted authority or a central server. The first Blockchain was implemented in the Bitcoin cryptocurrency in 2009 after its conception by the anonymous person or group Satoshi Nakamoto in 2008 [32].

Today Blockchain is mainly used for cryptocurrencies. However, there are high expectations that the technology can be used in other fields. One such field is communication for IoT devices. Multiple studies suggest the use of smart contracts, explained in section 2.5.5, on Blockchain platforms dedicated for the use of them such as Ethereum, explained in section 2.6, for blockchain based PKIs [17] [27].

1.2.1 Additional properties

A Blockchain based PKI solution is further motivated by the predicted future challenges facing the development of IoT. A recent gap study draws the conclusion that a dedicated marketplace and simple machine to machine communication is needed for the IoT platform to reach its full potential [31]. These conclusions are also reflected in a newsletter published by the IEEE but with a greater focus on the need for a secure communication standard [12].

1.2.2 State of the art

What is generally viewed as the first Blockchain based PKI was the CertCoin presented 2014 [22]. Since then multiple PKI approaches based on Blockchain technology and smart contracts have been developed. Modern commercial examples can be seen in emercoin [2] or beame.io [3], while scientific examples can be seen in the research papers *Dynamic Access Control Policy based on Blockchain and Machine Learning for the Internet of Things* [34], *Smart Contract-Based Access Control for the Internet of Things* [43] and *SCPKI: A Smart Contract-based PKI and Identity System* [13]. The basic ideas behind blockchain based key distribution are to spread public keys through the blockchain ledger.

1.2.3 CertCoin key distribution

The goal of CertCoin is to maintain track of domains and their associated public keys through the blockchain ledger. When a new domain is registered in CertCoin, a transaction is made that contains the signed information about two public keys associated with the new site. The first "public" key is used to authenticate the website while the second "offline" key is used to sign or revoke new keys. If a new key is created for the domain, it is signed with the old key of the same type. This way, both valid keys can be traced back to the initial registration of the domain. To look up or verify the public key pk of a domain d the following steps are taken [22]:

- Check that the domain d is registered exactly once.
- Check that all signatures in subsequent updates to the public key corresponding to d verify with d's previous public key.

- Find the last updated public key corresponding to d, and either check that it corresponds to pk or store it.
- Check that the user claiming to be the owner of domain d knows the secret key sk corresponding to pk using zero-knowledge proof of knowledge.

Even though multiple implementations have been developed from the method, there does not exist enough research studying the security of the approach needed to ensure trust in the model.

1.3 Project Goals

To build any PKI based on Blockchain technology, it must first be proven that there exists a safe way to distribute public keys in a Blockchain ledger. More specifically this involves adding, storing and reading public keys from a Blockchain ledger. The first goal of the project is to perform an informal cryptographic analysis and implement a simple Blockchain PKI to answer this question, this is referred to as Project Goal 1 (PG1). Based on the Blockchain PKI, the second goal of the project is to develop a simple proof of concept protocol that inherently has dynamic authentication, i.e. devices should be able to connect directly without going through a cloud service, and distributed logging, i.e. all data transactions are recorded and distributed to all parties, this is referred to as Project Goal 2 (PG2).

1.3.1 Research Question

Is it cryptographically safe to distribute public keys in public Blockchain ledgers?

1.3.2 Scope

In a typical setting, a cryptosystem is often evaluated on the basis of if a secret or private key of the system can be compromised by an adversary. This would lead to a total breakdown as the adversary would then be able to decrypt any message sent. There are however other attacks to cryptosystems that do not involve retrieving the secret key, such as only gaining some knowledge from the messages that can be

useful for the adversary. This involves being able to decrypt some messages but not all or being able to recognise ciphertexts from random strings. This is called semantic security.

Since the scope of the report is limited by the time limitations regarding a master thesis project, and because of the complexity of proving semantic security and probabilistic attacks, these will not be considered in the report. Instead, all cryptographic actions will be viewed as secure and atomic given their underlying assumptions hold. The report will only use one cryptographic protocol model and deductive reasoning when answering the research question. Only cryptographic safety with currently available technology will be considered, this means that quantum computing or other emerging technologies will not be regarded.

Denial of Service (DoS) attacks is a common form of adversary tactic that involves cutting of a devices network connection. Because of the complexity of protecting against DoS attacks and the time limitation regarding the project this will not be included in the scope of the project.

1.4 Scenario

In a newly constructed building, there are temperature sensors built into the walls. These devices are IoT devices that can be queried for the temperature of the room. The devices are referred to as producers since they produce data. When a new family moves into the building, they bring their own smart hub responsible for controlling the heating of the home. In order for the smart hub to efficiently control the heating, it will want to query the sensors of the house of their temperature reading. The smart hub is referred to as a consumer since it consumes the data the producers produce. The devices have never communicated before and have different vendors.

All devices are directly connected to the same Ethereum network with a private account. The account was created in a secure way and the device is the only holder of the account key. Once a device has successfully registered to the network, it is publicly listed. For example, the temperature sensors send their account information to the owner and the accounts are listed on the owner's website. In this environment, the integrity of any device is assured and there exists no way to

extract information from them except from the communication protocol. That is, all secrets stored on the devices are secured. Three use cases are derived from the scenario.

1. The consumer wants to read sensor data from a producer.
2. Owner of the producer wants to monitor and audit number of data requests from producer.
3. The producer wants to be able to accept multiple consecutive connection requests from previously unknown consumers.

Therefore, any communication protocol suggested for this environment should fulfil the requirements of the use cases.

Chapter 2

Theoretical framework

The security of the Blockchain technology relies on a couple of well-established security tools and principles. In order to perform a cryptographic analysis of distributing public keys with Blockchain, a basic understanding for the security of hash functions and public key encryption is needed. Then a theoretical background is given for Blockchain, Ethereum, cryptographic protocol analysis and Media Access Control (MAC) addresses.

2.1 Secure hash function

An efficient way to ensure data integrity is to identify the data with a unique fingerprint. The hash function is constructed to solve this problem. A hash function is a one-way function, $f(x) = y$, that takes data of any length, x, and has a seemingly random but unique mapping to a specific fingerprint hash value, y. For a hash function to be regarded secure, three properties need to be ensured [40]:

1. **Pre-image resistance** - Given a fingerprint y, it should be hard to find data x so that $f(x) = y$.

2. **Second pre-image resistance** - Given data x, it should be hard to find another data x' where $x \neq x'$ so that $f(x) = f(x')$.

3. **Collision resistance** - Given the hash function $f()$ it should be hard to find two different datasets x, x' where $x \neq x'$ such that $f(x) = f(x')$.

The produced fingerprint hash value is often of a fixed length. The common so far secure hash function, SHA-256, always produces fingerprint hash values of length 256 bits [40]. By producing a hash value of length 256 bits, 2^{256} unique mappings from data to fingerprint hash values can be created. This can be compared to the approximate number of 2^{272} atoms in the known, observable universe [23].

2.2 Security in cryptography

Unconditional security is never guaranteed in any useful cryptographic system. A useful cryptosystem should provide a way for two parties to communicate over a public channel without an adversary being able to understand the information being sent. There are often multiple assumptions that are required to assert that a cryptosystem is secure [33]. To successfully analyse any cryptosystem, it is essential to understand the fundamental assumptions being made and their impacts on the system. Following are some general assumptions used when analysing the security of cryptosystems.

2.2.1 Computational power

An underlying assumption is that an attacker never has unconditional computational power. If for example an attacker was given unlimited time and computational power, the total key space of every key would eventually be able to be brute force calculated. Therefore, the running time for accepted algorithms of attacks is often polynomial [40].

2.2.2 Computationally infeasible problems

Given polynomial limitation for running time, most cryptographic systems rely on the computational difficulty of the two following problems:

- **Prime integer factorization** - Given integer n, find two prime factors p, q such that $p * q = n$.
- **Discrete Logarithm Problem (DLP)** - Given a multiplicative group $(G, \cdot)$, an element $\alpha \in G$ having order n, and an element $\beta \in \langle\alpha\rangle$, calculate integer a where $0 \leq a \leq n-1$ such that $\alpha^a = \beta$.

So far, there exists no known polynomial algorithm to solve these problems running on classical computers. Given that large integers are used, this makes them computationally infeasible to solve. However, Shor's algorithm is a polynomial algorithm for quantum computers proposed already 1994 that solves both the problems [38]. But since the technology is still not publicly available quantum computing solutions are generally disregarded.

2.3 Public key encryption

The purpose of public key encryption is to enable two parties that have never communicated before to establish a secure connection over an insecure communication channel. The first conceptualisation of this was the Diffie-Hellman key exchange protocol that was introduced in 1976 [20]. This symmetric key encryption algorithm enabled two parties to establish a shared secret key over an insecure channel. However, the first true asymmetric key encryption algorithm, RSA, was published 1978 [37]. Elliptic Curve Cryptography (ECC) is another asymmetric key encryption first invented 1985 that is widely used in many modern systems because of its efficiency [30].

In an asymmetric cryptosystem, a user creates two keys, one public key and one private key. The two keys are connected so that anything encrypted with the public key can only be decrypted by the private key and vice versa. The user can then publish its public key and any party that wants to send an encrypted message can use it for encryption. The party is then ensured that only the user will be able to decrypt the message.

This builds on the basic idea of using a one-way function with a trapdoor. The one-way function creates a representation of a message that seems purely random. The representation will not give any information to an adversary about what the message was about. The message can only be restored by using a linked key, the backdoor. If an adversary does not have the correct key it should be computationally infeasible to restore the original message, while restoring the message with the correct key should be computationally easy.

2.3.1 Security of the RSA encryption scheme

For the RSA encryption scheme, the public and private keys are created so that the only known way to retrieve the private key from the public key is to factor a large integer. Therefore, the security of the RSA encryption scheme relies on the first stated assumption for computationally infeasible problems, that factoring large prime integers is hard [26].

2.3.2 Security of Elliptic Curve Cryptography

In ECC a group of points on an elliptic curve over a finite field replace the subgroup of $\mathbb{Z}_p^*$ [30]. Therefore, ECC relies on the same computational infeasibility of the DLP except in an elliptic curve finite field, this is referred to as the Elliptic Curve Discrete Logarithm Problem (ECDLP). ECDLP is considered computationally harder than the DLP which enables the use of smaller keys while still providing the same level of security [24]. At the time the Elliptic Curve Digital Signature Algorithm (ECDSA) was accepted as a standard by ANSI, IEEE and NIST (1999-2000) [24] no known polynomial algorithm existed for solving ECDLP even including speculations regarding quantum computers. It has since been proven that shor's algoritm can be adjusted to solve also ECDLP with additional estimations of hardware requirements [28].

2.4 Digital signatures

Digital signatures are used to ensure the authenticity of a document or a message. The signature serves as a proof of correctness and authenticity for any intended reader of the document. In order to do this a digital signature has to ensure the three following properties:

1. **Authentication -** A valid signature can only have been produced by the sender.

2. **Non-repudiation -** Once a digital message is signed and published, the sender cannot deny having signed the message.

3. **Integrity -** The message has not been altered after it was signed and send.

The digital signature also has to be easily verified by a third party. The first formally accepted digital signature scheme to achieved legal legitimacy was the Digital Signature Algorithm (DSA) issued by the National Institute of standards and technology 1991 [39]. DSA used properties from secure hash functions and public key encryption to generate a private key for signing and a public key for verification. Simplified, DSA encrypts a fingerprint of the document with a private key. Any user holding the public key can then themselves generate a fingerprint for the document and verify that this is the same as the fingerprint encrypted with the document. Since the introduction of DSA new methods has been developed. A conventional method for digital signatures used today is the ECDSA based on Elliptic Curve cryptography.

2.4.1 Elliptic Curve Digital Signature Algorithm

ECDSA is a Digital Signature Algorithm which uses ECC as a basis. The description of the protocol is outside the scope of the report, instead the properties of it are presented. With ECDSA we have a private key used for signing and a public key used for verification. ECDSA utilises fingerprint hashes together with properties of its keys to ensure that the three required properties of a digital signature hold.

Authentication is ensured by the uniqueness of the signature generated by the private key. Only the holder of the private key can create a valid signature verifiable with the public key. Non-repudiation is ensured by the binding of the private key to an entity. Any signature produced by the private key has legal binding, therefore it is the best interest of the owner of the private key to keep it hidden. Integrity is ensured by the fingerprint of the document together with properties of the key. If the document has been changed since it was signed, the signature will prove invalid.

The security of ECDSA relies on the computational infeasibility of ECDLP. The ECDSA is defined by what properties the elliptic curve it uses has. For example, Ethereum uses the elliptic curve secp256k1 [1]. The choice of the elliptic curve is critical for the security of the algorithm. If the properties of the elliptic curve are badly chosen it might be trivial to solve the ECDLP problem. There have also been instances where NIST recommended elliptic curves might have been deliberately constructed containing backdoors [15].

2.5 Blockchain technology

The Blockchain can be viewed as a state transition system. The states are defined by accounts holding currency or information. To retrieve the full state of the system one has to view all the accounts of the system. In all state transition systems, there needs to be a way to change the state. To change state in Blockchain technology transactions are used to transfer currency or information.

2.5.1 Transactions

In Blockchain the currency, referred to as coin, is defined as a chain of digital signatures. Using public key encryption, the ownership of the coin can be transferred. This is done by digitally signing a hash consisting of the previous transaction together with the public key of the next owner and adding these to the end of the coin. With transactions some Blockchain systems allow additional information to be appended. By the properties of digital signatures, two important features are ensured. First, it is easy for the receiver of a coin to verify the signatures to verify the chain of ownership. Second, only the previous owner of the coin can produce the correct signature to send it forward. This prevents unauthorised state changes [32].

However, in order to ensure that a coin is not double spent, i.e. sent to two receivers at the same time, the transactions need to be atomic. This is ensured by the concept of proof of work and first to file.

2.5.2 Proof of work

Proof of work is an addition to the timestamp server method. In a timestamp server, a collection of data creates a block. The block is then hashed with the previous timestamp, forming a chain, to create a new timestamp. The timestamp is then published. Each timestamp reinforces the one before it and ensures a certain order for the data. The data must have existed at the time of the timestamp to get into the hash. The data will be included in the blocks depending on the time they reach the timestamp server. This is referred to as first to file. This ensures a single line of history for the data since the order of data is important to produce the same hash. Therefore, all the data in the block is locked into place once a new timestamp is created [29].

The timestamp method works for a centralized network, but in order to achieve a distributed state transition system, some modification is needed. The calculation needs to be able to be done at any point in the network at any time. This is solved by creating a challenge with a reward for each new block. The block is extended with a nonce that the block creator can choose. The challenge is to find a nonce that produces a block hash with a certain amount of leading zeros. The challenge can create the following behaviour:

1. New transactions are broadcasted to the network.
2. Each node collects and verifies transactions (verifying the trail of hashes) in a block.
3. Each node tries to find a hash for its block with a certain amount of leading zeros.
4. When a nonce is found the new block is broadcasted to the network.
5. Network accepts the block only if the included transactions are valid.
6. Nodes express their acceptance by working on creating the next block in the chain based on the accepted block as the previous hash.

The speed of block production can then be adjusted by changing the difficulty on the challenge, where the average computations needed are exponential to the number of leading zeros.

The proof of work also solves the problem of determining what line of history that is correct. This is simply done by always viewing the longest chain as the correct representation of history. The longest chain produced by the network will have most computational work in it. The nodes that calculate the proof of work are commonly referred to as miners and producing new blocks is referred to as mining. The probability of creating and propagating multiple correct blocks at the same time is at around 1.78% [19]. Therefore, the risk of maintaining multiple correct chains of the Blockchain history, both chains grow at the rate, is practically negligible.

2.5.3 Incentive

There are two main incentives for miners to calculate the new blocks. The first is a special transaction included in each block that starts a new coin. This is used to initially distribute coins into circulation and the coin is rewarded to the creator of the block. The second is transaction fees that can be paid to the miner. If the output value of a transaction is less than its input, the difference is added to the miner's account. With these incentives, nodes will try to be first when calculating new blocks. This causes the network to quickly agree on a single line of history.

2.5.4 Security of Blockchain

In the cryptocurrency Blockchain, public key encryption prevents coin theft or the creation of false transactions and hashes in the proof of work create a single line of history preventing double spending. The state and correctness of the Blockchain is thus secured. Therefore, the only attack left for an adversary is to change its own transactions. An attack of this kind can be described with the following scenario:

1. Adversary A orders something from B and pays with a crypto currency.

2. B verifies that the transaction from A has been processed in the Blockchain and sends the order.

3. Once the order is sent, A calculates an alternative chain that does not include A's transaction to B.

4. A's alternative chain surpasses the honest chain so that it is accepted as the correct representation of history and the transaction thus never happened.

5. A receives an order that B never gets paid for.

The race for the adversary to surpass the honest chain can be characterised as a Binomial Random Walk. The probability of an attacker catching up is said to be analogous to a Gambler's Ruin problem [32]. The probability can be calculated as follows:

$p =$ probability that an honest node finds the next block
$q =$ probability that the attacker finds the next block
$q_z =$ probability that the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (\frac{q}{p})^z & \text{if } p > q \end{cases} \tag{2.1}$$

Equation 2.1 shows that given at least 51% of the computational power in the network are honest nodes, the probability that this attack succeeds drops exponentially with the number of blocks the adversary is behind. From this equation, the legitimacy of the transaction can be ensured to B by creating a lead for the honest chain. Without knowing the amount of progress the attacker has made since the transaction was registered, the attacker's potential progress is a Poisson distribution with an expected value described by equation 2.2.

$$\lambda = z\frac{q}{p} \tag{2.2}$$

To get the probability of the attacker catching up, the Poisson density for each amount of progress is multiplied by the probability of catching up from that point. This can be calculated by equation 2.3 where k represents the number of blocks the adversary already has calculated:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} * \begin{cases} (\frac{q}{p})^{z-k} & \text{if } k \leq z \\ 1 & \text{if } k > z \end{cases} \tag{2.3}$$

It has been proven that the probability drops off exponentially with z [32]. Equation 2.3 can be used by B to calculate a comfortable probability before accepting a transaction as final.

2.5.5 Smart contracts

Smart contracts are a concept of ideas proposed already 1994 [41]. Smart contracts enable enforcement of agreement through the Blockchain

by having parties admit self-executing code to the ledger that works on an if else premise. The contracts are distributed throughout the network, so they inherit the properties of transactions on the Blockchain. This makes them reliable and immutable. The contracts are thus visible and verifiable by all parties.

Users interact with smart contracts by performing transactions with function calls towards them. The miner verifies that the transactions are valid try to execute the function call towards the smart contract. The status of the execution is recorded in along with the transaction in the mined block. If the execution is successful, function calls could cause a state transition in the system. If not, an error was encountered mining the transaction and no changes were made to the state. Either way, the transaction is included in the Blockchain and removed from circulation so that miners won't process it again.

This technology is viewed as one of the cornerstones for Blockchains use in IoT communication. Ethereum is a cryptocurrency platform specifically designed with smart contracts in mind.

2.6 Ethereum

Ethereum is a cryptocurrency platform powered by Blockchain technology with a built in turing-complete programming language. Ether is the cryptocurrency whose Blockchain is generated by the Ethereum platform. By differentiating the platform from the cryptocurrency, it more resembles a distributed computing platform than just a pure cryptocurrency platform. The platform supports the implementation of advanced smart contracts and pure decentralized applications formatted to fit any purpose. Accounts operating on the Ethereum platform consist of both externally owned accounts, controlled by public and private keys and contract accounts, controlled by their contract code [16]. The "first class citizen" property of Ethereum states that externally owned accounts and contract accounts have equivalent power.

In Ethereum all accounts are said to represent a state and transactions between accounts represent state transitions. Since the first to file attribute hold for transactions the system can be viewed as a state transition machine where each transaction transforms the state of the Blockchain to the next. At any given time, all transactions combined therefore give the current state for the system. The platform also pro-

vides an open sandbox environment which makes it suitable for research purposes.

2.6.1 Ethereum virtual machine

The Ethereum Virtual Machine (EVM) is where contracts are run in Ethereum. Contract accounts cannot send transactions by themselves but can launch them as a response to a transaction. Therefore, a transaction from an externally owned account is required to start a sequence of contract executions. The contract code execution is performed by the miner when it's mining a new block. When contracts send transactions to other contracts this is referred to as message calls. Message calls can only exist in the runtime environment. Therefore, the result of all contract calls caused by an externally owned accounts transaction are mined in the same block. The order of the execution of the transactions are recorded in the block and indexed. The order of the transactions prevents data races from happening in the EVM. The resulting state changes together with the indexed transactions are then included in the next block mined by the miner.

Upon receiving the new block, other nodes of the system can retrace the miner's transaction order and execute the contract code. If the block is accepted by the other nodes, a consensus has been reached about the contract codes execution and results. Therefore, the consensus of the execution of the virtual machine is reached by the same way as the other transactions in the system. Security regarding the execution of a contract is assured by the trailing child blocks after the block the contract execution transaction is included in, described in section 2.5.4.

Return values from functions of smart contracts that invoke state transitions can only be accessed by contract accounts. If externally owned accounts wish to retrieve a return value for functions that invoke state transitions, event logs are used. Event logs are distributed together with their respective mined block. This is to allow externally owned accounts to assert the validity of a returned value by the trailing child blocks. However, functions that do not invoke state transitions returns directly to externally owned accounts.

2.6.2 Ethereum Blockchain data structure

The Ethereum Blockchain is best viewed as a decentralized, replicated database in which the current state of all accounts is stored. All full nodes of the Ethereum network are expected to hold a local key/-value database that maintains a mapping of byte arrays to byte arrays, referred to as a state database. The state of all accounts in the system, that represents the state of the system is then stored in the state database according to the Merkle Patricia trie structure described in section 2.6.3 [42]. Therefore, looking up information in the storage is done locally on each node's internal database.

The state of each account is stored in a four-tuple containing:

- **Account nonce -** The number of transactions sent from the account, kept to prevent replay attacks.
- **Ether balance -** Ether balance of the account.
- **Code hash -** Hash of the code if the account is a contract, otherwise an empty string. The Code is stored in the state database under the corresponding hash.
- **Storage root -** Root of another Merkle Patricia trie that stores the data for the account.

The state root works as the start node for looking up information in the state. The state root is distributed with each new mined block and the address of the accounts is used as a path to the state of the specified account, illustrated in figure 2.1. Because of the properties of the Merkle Patricia trie structure, the shared state of the system is verifiable with the state root. The state root can therefore be used to check that the state transitions recorded by the miner were correct. If not, another state root would be derived by the node. Because of the unlikelihood of a hash collision and as long as the state database is not purged, any older state of the system can also easily be explored starting from an older state root.

2.6.3 Merkle Patricia trie

A Merkle Patricia trie is a key/value storage that utilises nodes to build deterministic paths to resources. Each node is stored in an un-

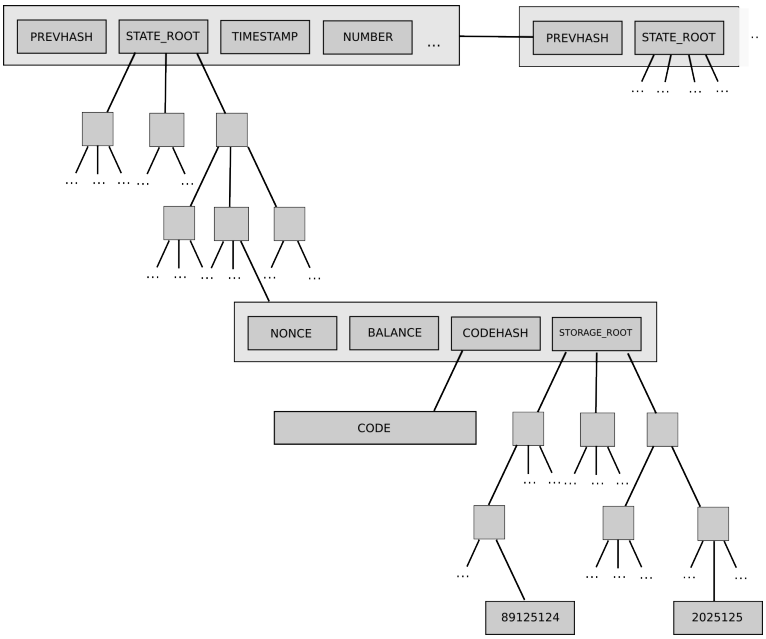

Figure 2.1: Ethereum data structure, *Source:* [4]

derlying key/value storage by its own hash value. A node can be one of the following four types:

1. **NULL -** represented as the empty string.
2. **branch -** A 17-item node [v0 ... v15, resource].
3. **leaf -** A two item node [encodedPath, resource].
4. **extension -** A two item node [encodedPath, node hash].

The first sixteen items of a branch node are used to store hash values to other nodes or the NULL node, that is an empty node. The last item is used to store a resource. The leaf is used as an end node that allows skipping ahead directly to the resource. The first item of the node stores the remainder of the encoded path and the second item stores the resource. The extension node is used when all stored resources for a path share some part of their path. The shared part of their path is stored as the first item and the hash value of the next node is stored as the second item.

The path is parsed from the key to the resource, for example in Ethereum the key to the accounts state is its address. The path is built

up of the hex representation of a key. So if the key is *dog* the path would be the hex representation of dog, giving *64 6f 67*. The path is traversed by starting from a set root node and parsing parts of the path depending which node is encountered. Table 2.1 describes the parsing taking place for each node type.

NULL	Represents an empty value, meaning no resource is stored for the path.
branch	Uses the value of the first nibble of the remaining path as an index of the array. The value of the index is used to lookup the next node in the underlying database. If there are no more steps in the path, returns the resource of the node. If the index is empty, no resource is stored for the path.
leaf	If the remaining path is the same as the encodedPath, the resource is returned. Else, no resource is stored for the path.
extension	If the encodedPath corresponds to the next steps for the remaining path, the node hash is used to lookup the node after the skip ahead. Else, no resource is stored for the path.

Table 2.1: Path parsing for each node type

So given the path *64 6f 67* and that there are branch nodes all the way to the resource. The resource is found by starting at the root node and then looking up the entries in the underlying database corresponding to the value of the node indexes $6 \rightarrow 4 \rightarrow 6 \rightarrow 15 \rightarrow 6 \rightarrow 7$.

Merkle Patricia trie ensures integrity by using the hash value of the node as its key in the state database. And since each node is pointed to from previous parts of the path, this ensures that any change in data for a node is propagated upwards to all parent nodes.

Merkle Patricia trie also has the property that if a node is changed for the trie, only the affected branch needs to be updated in the state database. Therefore, unaffected branches of the trie can still point to the old nodes. This is used in the Ethereum storage to only update affected storages and is illustrated in figure 2.2 [4].

2.6.4 Solidity

One programming language that can be used to generate the binary code used by smart contracts is Solidity. The language was influenced by C++, Python and JavaScript and is designed to implement smart

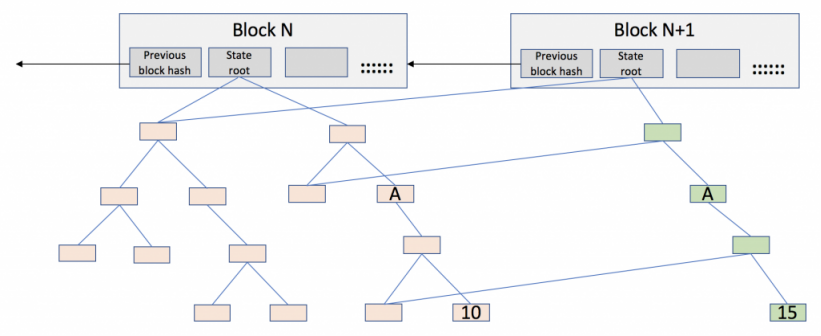

Figure 2.2: Merkle Patricia Trie update, *Source:* [5]

contracts. Solidity is a statically typed programming language that has support for inheritance, libraries and complex user-defined types [6].

One important aspect of the language is the data type *mappings* that together with the Ethereum data structure becomes a powerful tool for smart contracts to store values. The mappings data type can be seen as a hash table which is virtually initialised such that every possible key exists with a mapping to a value whose byte-representation is all zeros. Therefore, a lookup in mappings is always successful, even if nothing is stored for the value. When a new key/value is stored in the mapping, this can be added without re saving the whole mappings structure. This is due to the properties of the underlying Ethereum data structure.

2.6.5 Geth

Geth is a command line interface for running full Ethereum nodes implemented in Go. Geth supports all actions an Ethereum account can take along with mining ether and exploring the block history of the Ethereum network. Geth is developed as an open source project and is available through Github [7].

2.7 Cryptographic protocol analysis

Cryptographic Protocol Analysis (CPA) is the field of analysis of cryptographic protocols. The goal of CPA is to ensure trust in the tested protocol in regard to certain security goals. The security goals are spe-

cific for each protocol but can for example be to ensure secure authentication between two parties through the protocol. CPA consists of a suite of models and verification methods. Because reality is often too complex to model, abstractions of reality are used. One of the most common abstractions is to remove the probabilistic uncertainty of a protocol [18]. A common model that abstracts away probability is the Dolev-Yao model.

2.7.1 Dolev-Yao model

The Dolev-Yao model was specifically created to study the security of public key protocols. The method has been tried and tested for over 30 years and is a standard method used in protocol analysis. The Dolev-Yao model is a CPA model used as base for verifying the behaviour of a protocol. The model is based on the assumption that all cryptographic functions are safe and atomic. More precisely the Dolev-Yao model states that the following four assumptions hold [21]:

1. Perfect public key system.
   (a) One-way functions used are unbreakable.
   (b) Public directory is secure and cannot be tampered with (place where the public keys are distributed).
   (c) Everyone has access to all public keys.
   (d) Only the owner knows his private key.
2. Assistance of third party in decryption or encryption is not needed.
3. Uniform protocol, same format is used by every pair of users to communicate.
4. Saboteur is an active eavesdropper. Someone first taps the communication to obtain messages and then tries everything he can in order to discover the plaintext. More precisely the following assumptions applies to the saboteur:
   (a) He can obtain any message passing through the network.
   (b) He is a legitimate user of the network and can initiate a conversation with any other user.
   (c) He will be received by any user.

These assumptions are commonly referred to as perfect encryption since they abstract away ways to break encryption without a key. In order to use results achieved by the Dolev-Yao model, mapping from the assumptions to the system of operation is needed. It has been argued that proofs based on the Dolev-Yao model should be sufficient to consider a protocol secure [11]. An important note from this is that there exist secure protocols analysed in the Dolev-Yao model, such as the SSH protocol [36]. Therefore, any system proven to be equivalent to the Dolev-Yao model, as seen later in section 4.2, can be used to run these protocols as well. The model gives an easier representation of reality that then can be used in protocol verification tools. An example of this is an algorithmic method for checking if a name-stamp protocol is secure included in the Dolev-Yao paper [21].

2.7.2 Protocol verification

The abstracted model created by the Dolev-Yao can then be used together with verification tools for protocols. The inductive verification approach often portrays the protocol as a state machine where all states are explored. If a state that break any of the security goals specified for the CPA is reached, the protocol is unsafe in regard to its security goals [35] [18].

When only properties of an already proven secure protocol is used to create a new protocol, deductive verification can instead be used. The goal of the CPA is then to prove that the additions to the protocol does not break any of the pre-existing security goals. Deductive verification through proof by contraction is a common method used in the field of mathematics and cryptology [40]. The method involves assuming that the security goal under test has been breached and deducts that this leads to a contradiction with other properties of the protocol or system. Therefore, either the security goal is not breached, or the other properties of the protocol or system do not hold.

2.8 Media Access Control address

In order for devices to be uniquely identified in a network, each device has a MAC address assigned to the network interface controllers. The MAC address is unique for each network connected device in the world and is typically never changed for the devices life time. The

MAC address is often stored in hardware and is assigned by the component manufacturer but there also exists software defined MAC address schemes.

Chapter 3

Methods

This chapter explains the methodology used to reach the two project goals and answer the research question.

3.1 Literature research

The first step of the project was a thorough literature research. The goals of the literature research were to gain enough understanding in the fields of PKI, Blockchain and cryptographic analysis to reach the two project goals. Initially, the google scholar platform was used to identify important research papers related to the study. The search started using keywords such as *Public key infrastructure, Blockchain, Public key security*. The search was then broadened using an iterative method where references from identified important research papers were used. An important reference for the project was also the book *Cryptography Theory and Practice* [40].

3.2 PKI security goals

The literature research formed a general set of security goals required for any secure PKI. Any functioning public key infrastructure, as mentioned in section 1.1, requires safe registration and distribution of public keys. Therefore, the following security goals were defined to represent the requirements of a secure PKI:

1. **Safe key deployment -** Only the owner of a device should be able to safely tie a public key to a physical ID in the PKI.

2. **Safe key retrieval -** Given the ID of a device, safely retrieve its connected public key.

The research question could then be answered by analysing the integrity of the security goals on a Blockchain based PKI.

3.3 Analysis model

To remove complexities of the real world, a model is often used in cryptographic analysis. Dolev-Yao model was specifically created to study the security of public key protocols. The method has been tried and tested for over 30 years and is a standard method used in protocol analysis. As explained in section 2.7.1, many authentication protocols such as SSH have been proven secure in regard to the model. The Dolev-Yao model abstracts away probability and semantic security. However, the first assumption of the model, *perfect public key system* listed in section 2.7.1, is used to abstract away difficulties involved with public key distribution. This assumption undermines any analysis ensuring the integrity of the security goals for a PKI. Therefore, a modified model based on the Dolev-Yao model was constructed.

3.3.1 Blockchain model

In order to preserve the integrity of the Dolev-Yao model the underlying assumptions for the Dolev-Yao model were not changed. The difference for the constructed Blockchain model was that the assumption of a perfect public key system was removed and the Blockchain assumption was added. Therefore, the Blockchain model used for the analysis was built on the following assumptions:

1. All cryptographic functions are safe.
2. One-way functions used are unbreakable.
3. Only the owner knows his private key.
4. Assistance of third party in decryption or encryption is not needed.
5. Uniform protocol for all users.

6. Saboteur is an active eavesdropper - Can obtain any message, is legitimate user, will be received by any user.
7. Majority of the computational power in the Blockchain network are honest nodes.

More specifically the assumptions removed from the Dolev-Yao model were:

A. Public directory is secure and cannot be tampered with
B. Everyone has access to all public keys

The Blockchain model provided two things. First, a model where the security goals could be analysed for a PKI to answer the research question. Secondly, by analysing if the two removed Dolev-Yao assumptions held true given the security goals were true, it provided a way to verify that the security goals represent the state of a perfect public key system. Assuming this was true, any PKI proving to hold the security goals in the Blockchain model would represent the state of a perfect public key distribution system in the Dolev-Yao model and thus also rendering any protocol, such as the SSH protocol, proven secure in the Dolev-Yao model to be considered secure in the Blockchain model.

3.4 Analysis method

For formal proofs in cryptography the whole environment and all possible actions must be formally defined. Providing this would require more time than was available for the research project. Informal analysis simplifies the requirements of an analysis, but at the cost of the scientific accuracy of the results. For the purpose of the project, informal analysis was viewed as an adequate method to answer the research question.

Proof by contradiction is a common method used in the field of mathematics and cryptography. It is an effective method used as base for many mathematical proofs. It provides a safe way to assert the validity of propositions from existing knowledge. In cryptography it is often very hard to prove that certain security properties hold. Therefore, proofs are often constructed by asserting that the opposite

statement results in a contradiction for the properties of the system. Both the necessary informal proofs needed to evaluate the Blockchain model and answer the research question could be constructed using proof by contradiction.

The Blockchain models equivalence to the Dolev-Yao model was evaluated by analysing propositions that represented a breach of the two removed assumptions when the security goals were assumed true. The constructed propositions were:

1. Public directory is not secure and can be tampered with
2. Some public keys are not accessible

The Blockchain PKI was evaluated by a similar method. First the two propositions representing a breach of the corresponding security goal were constructed:

1. Imposter can register a public key for another device ID.
2. Imposter can provide its public key in place for another device ID.

The propositions were then analysed for the Blockchain PKI using the Blockchain model. If a contradiction was found when the proposition was assumed true, the corresponding security goals could be assumed to hold. An informal proof could then be constructed evaluating if the Blockchain PKI could be used to build a secure key distribution infrastructure, thus answering the research question.

3.5 Implementation

The following describes the configuration and environment used for implementing and testing the Blockchain PKI smart contract and the access control smart contract.

3.5.1 Hardware

The network and all participating nodes were run on a personal laptop computer with the specifications presented in table 3.1.

OS	Processor	Ram	Storage
macOS Sierra, Version 10.12.6	2,3 GHz Intel Core i5	16 GB DDR3	256 GB

Table 3.1: Computer hardware specifications

3.5.2 Software

The Ethereum cryptocurrency platform was used for testing the implementations. The Geth software was used to configure a private network and run all participating nodes in the network. All smart contracts were constructed using the Solidity programming language using version 0.4.0. The contracts were compiled using the built-in command line compiler and documentation for the language and best practice guidelines were gathered from the solidity documentation [6]. Scripts for setting up the testing environment was constructed in Python. Scripts for automating the actions of the nodes were constructed in JavaScript, available in Appendix A and B, and loaded into the Geth software.

3.5.3 Ethereum network configuration

A private Ethereum network was initiated using a constructed genesis block, presented in listing 3.1.

```
1  {
2    "config": {
3      "chainId": 1994,
4      "homesteadBlock": 0,
5      "eip155Block": 0,
6      "eip158Block": 0,
7      "byzantiumBlock": 0
8    },
9    "difficulty": "400",
10   "gasLimit": "2000000",
11   "alloc": {
12     "43b35057e4a3fed1bc1ec8cec5b9d763a8672e57": {
13       "balance": "100"
14     },
15     "dd485ab9c96ac346ca01ed86769475a55f9d5cd8": {
16       "balance": "100"
17     },
18     "2d36b7e3f6a90c6659cc3ba1b25bc8e10cb720fb": {
19       "balance": "1000000000000000000"
20     },
21     "a0d56b2dc4045895d57c430bebead2fa637d23e2": {
22       "balance": "1000000000000000000"
23     },
24     "3fc3b686a3e688bd1be4d31fe1bfcb232a64fde4": {
25       "balance": "1000000000000000000"
26     }
27   }
28 }
```

Listing 3.1: Genesis block

The configuration specifies the difficulty for mining new blocks, maximum gas limit for transactions and five accounts. The first two accounts, initiated with a smaller amount of Ether on row 12 and 15, were intended as mining accounts. The three last accounts, initiated with a higher amount of Ether on row 18, 21 and 24, were intended as actors in the tests able to perform transactions.

For each test the Ethereum network was restarted from the genesis node. Two miners were used to simulate the state of a consensus network.

3.5.4 Blockchain PKI test suite

The purpose of the Blockchain PKI test suite is to assure the functional requirements for a Blockchain PKI defined in section 4.1.1 are fulfilled. For all tests in the Blockchain PKI test suite, the following setup was performed upon initialisation:

1. The two miner accounts are instructed to start mining.

2. The setup account deploys the Blockchain PKI smart contract, and the address to the smart contract is recorded. This account is then never used again.

3. The device accounts are given the address to the Blockchain PKI smart contract.

The state of the network after initialisation is illustrated in figure 3.1.

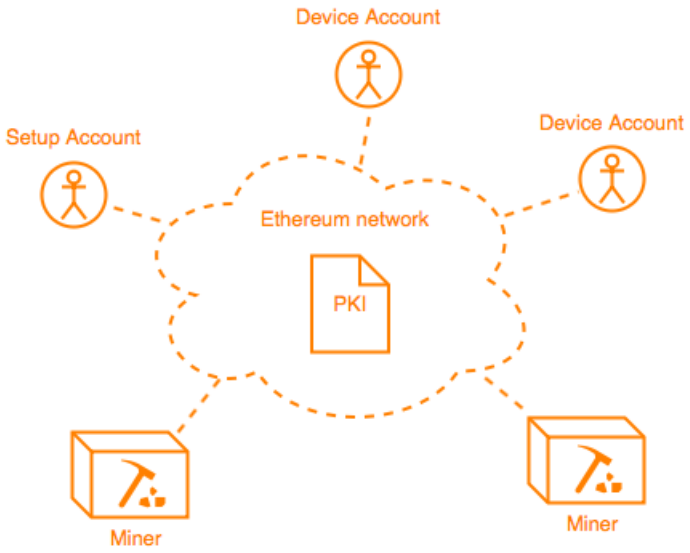

Figure 3.1: Blockchain PKI test suite network

The test cases of the test suite are defined in table 3.2. For each test the required action is stated along with the expected result and a verification method to verify the result.

3.5.5 Access control test suite

The purpose of the access control test suite is to assert that the required functions from an access control smart contract, defined in section 4.5.1, are fulfilled. For all tests in the access control test suite, the following setup was performed upon initialisation:

1. The two miner accounts are instructed to start mining.

No.	Action	Expected result	Verification
1.	Register one device.	Registration transaction is accepted and executed, device is registered with PKI.	Public key and account address is retrievable from PKI with device ID.
2.	Register two different devices.	Registration transactions are accepted and executed, devices are registered with PKI.	Public keys and account addresses are retrievable from PKI with device IDs.
3.	Register a device on an occupied ID.	Registration transaction is accepted but not executed.	Public key and account address for first the already registered device is returned when lookup is performed in PKI for device ID.
4.	Request public key and account address for not registered ID.	Returns nothing.	Nothing is stored for not registered ID.

Table 3.2: Blockchain PKI test suite

2. One account is picked as a producer account and the access control smart contract is deployed, the address to the smart contract is recorded.

3. Two accounts are picked as consumer accounts and are given the address to the producer access control.

The state of the network after initialisation is illustrated in figure 3.2.

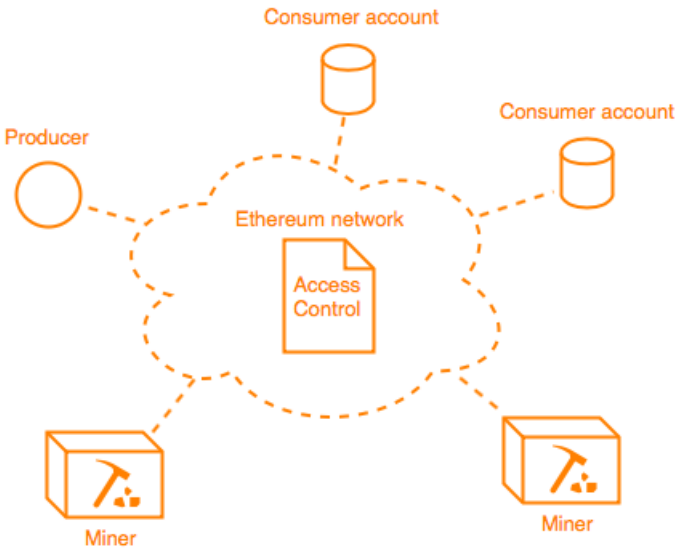

Figure 3.2: Access control test suite network

Test cases of the test suite are defined in table 3.3. For each test the

required action is stated along with the expected result and a verification method to verify the result.

No.	Action	Expected result	Verification
1.	Consumer requests access to free producer.	Consumer ID is added to producer access control.	Consumer ID is retrievable from smart contract.
2.	Consumers requests access to occupied producer.	Consumer request is denied, no change in access control.	Old consumer ID is retrievable from smart contract.
3.	Occupied producer requests ID of consumer from access control.	Access control returns consumer ID.	ID retrieved is the same as consumer granted access.
4.	Free producer requests ID of consumer from access control.	Nothing is returned.	No consumer ID is stored in the smart contract.
5.	Occupied producer resets its access control.	Producer is marked as free and can accept a new request from consumer ID, stored consumer ID is removed from access control.	No consumer ID is stored in the smart contract and access control accepts next request.
6.	Free producer resets its access control.	Nothing changes in the smart contract.	No consumer ID is stored in the smart contract.

Table 3.3: Access control test suite

Chapter 4

Results

In this chapter, the results of the project are presented. This involves a constructed Blockchain PKI, two informal analysis and a proof of concept dynamic access protocol.

4.1 Blockchain PKI

The basic functions of a PKI, mentioned in section 1.1 and given by the literature research, is registration, where new public keys are connected to physical identities, and key distribution, where users can query the PKI for public keys tied to IDs in the system. The Blockchain PKI was therefore defined with the following configuration:

- **Registration -** Binds the MAC address and account address of a device with a public key through a transaction in the Blockchain.
- **Distribution -** Public keys are stored and distributed through the public ledger.

Because of the properties of MAC addresses, explained in section 2.8, connecting a MAC address to a public key was deemed fit to serve the purpose of binding a physical entity with a digital key. The first registration for a MAC address is viewed as the valid connection between a digital key and a physical entity. To stop an adversary from allocating all MAC addresses on the Blockchain a registration cost could be added. The device verifies its registration towards the state of the Blockchain. If the device does not see its MAC address is registered

with the correct key and account address, a new MAC address is generated by the device and is sent with a new registration transaction. An adversary therefore cannot know the correct MAC address of a device before registration. Since the Blockchain system naturally distributes the state of the public ledger to all nodes in the system, the distribution of the public keys is done through the public ledger.

To register in the Blockchain PKI, a device performs the following steps:

1. Creates an account in the Blockchain system and generates a public, private key pair for data transfer.
2. Generates a MAC address.
3. Sends a registration transaction with its MAC address and public key to the Blockchain PKI.
4. Waits until the transaction is mined and enough trailing child blocks to the transaction block has been created.
5. Checks if its MAC and account address are registered in the Blockchain. If so the device has successfully registered. Else, repeat all steps from step two.

There are two important remarks about step four. The first is that If the Blockchain PKI is designed as a smart contract, the registration transaction is always eventually mined, as explained in section 2.5.5, regardless if the MAC address is already occupied in the Blockchain. Therefore, the device can successfully wait for the transaction to be mined. The second is that the that the trailing child blocks of the transaction block are the devices guarantee of its registration, as explained in section 2.6.1.

4.1.1 Functional requirements

For the Blockchain PKI the minimum functional requirements identified were:

1. **Register public key -** Registering a public key and an account address with a MAC address as ID together on the Blockchain.
2. **Retrieve public key and account address -** Query after public key and account address connected to MAC address as ID on the Blockchain.

4.2 Informal Blockchain model analysis

In order to evaluate the equivalence of the Blockchain model, given that the security goals are true, and the Dolev-Yao model, abstracting a perfect PKI, the identified requirements of the PKI security goals, listed in section 3.2, and the implications of the propositions for the removed assumptions were listed. The requirements and implications are then analysed using proof by contradiction.

4.2.1 Requirements and implications

The first security goal, safe key deployment, involves safe key registration and publication of a public key together with a physical identity. In the traditional CA PKI this is handled by a RA. Therefore, the following were the identified requirements by the first security goal:

A. Safe key deployment

1. Only owner of ID can deploy public key for ID.
2. Public key is not changed in transit and authenticated towards provider upon arrival.

The second security goal, safe key retrieval, involves safe retrieval of all keys deployed in the system. In a traditional CA PKI this is handled by the CA and therefore involves safe storage of public keys, safe validation of received keys and safe delivery of all stored keys in the system. Therefore, the following were the identified requirements by the second security goal:

B. Safe key retrieval

1. All IDs in storage are accessible.
2. Public key and ID deployed is preserved in storage so that the same public key is provided on request.
3. All ID and public key connections can be verified by the user.

The implications from the propositions representing breaches of the two removed Dolev-Yao assumptions were then identified and

listed. The first propositions, public directory is not secure and can be tampered with, implies that the public directory providing the public keys is insecure. Therefore, it is possible for an adversary to change any information in the public directory or control the connections between a user and the public directory. The following implications were identified for the first proposition:

C. Public directory is not secure and can be tampered with

1. Public keys or IDs for public keys can be changed in storage.
2. Requests can be hijacked to provide another public key for storage or delivery.

The second proposition, some public keys are not accessible, implies that there are users in the system that have deployed their public keys, but the deployed public key is not accessible. Therefore, only one implication was identified for the second proposition:

D. Some public keys are not accessible

1. Not possible to get public key of ID in storage.

4.2.2 Contradictions

If we assume that the security goals, A and B, and the first proposition, C, are true, we can informally identify the following contradictions in the system for all the implications of C:

$$\text{B.2 contradicts C.1} \tag{4.1}$$

$$\text{A.1, A.2 and B.3 together contradicts C.2} \tag{4.2}$$

Since requirement B.2 ensures that the same public key that was delivered for storage is the same delivered upon request to a user this does not allow a public key or ID to be changed in storage which is a direct contradiction to C.1. This contradiction is defined as contradiction 4.1.

Since requirement A.1 ensures the authenticity of delivered keys and requirement A.2 ensures the safe delivery of deployed keys, hijacking deploy attempts are made impossible for an adversary. Requirement B.3 also ensures that the received keys from storage can be

verified by the user. This leaves no room for hijacking connections and replacing keys. Therefore, A.1, A.2 and B.3 create a contradiction when regarded as true in the same system as C.2. This contradiction is defined as contradiction 4.2.

If we assume that A, B and the second propositions, D, are true, we can informally identify the following contradiction in the system for the implication of D:

$$\text{B.1 contradicts D.1} \tag{4.3}$$

Since requirement B.1 ensures that all deployed keys in storage are accessible, there does not exist public keys with IDs in storage that are not available. Therefore, B.1 contradicts D.1. This contradiction is defined with equation 4.3.

The contradictions 4.1, 4.2 and 4.3 show that the propositions representing breaches of the two removed Dolev-Yao assumptions creates contradictions in a system where the security goals hold. This shows that assuming the security goals are true for the Blockchain model, the removed Dolev-Yao assumptions are also ensured. Therefore, the Blockchain model together with the security goals is equivalent to the Dolev-Yao model with an abstracted perfect PKI.

4.3 Informal Blockchain PKI security goals analysis

In order to informally analyse the PKI security goals, listed in section 3.2 for the Blockchain PKI in the Blockchain model we list important properties of a Blockchain PKI, construct propositions representing breaches or the PKI security goals and list the implications for those breaches. The properties of the Blockchain PKI and proposition implications are then analysed by proof of contradiction.

4.3.1 Properties and implications

Since the Blockchain model includes the Blockchain assumptions, the following are important properties for the analysis of the Blockchain PKI given the Blockchain model:

D. Blockchain PKI

1. Distributed immutable public directory by the public ledger.
2. Only account holder can perform transactions for accounts.
3. MAC addresses are unique and unpredictable for each device.
4. First registered public key for MAC is valid.
5. MAC address registration has a cost preventing occupation of all MAC addresses.

The proposition for first breached security goal is unsafe key deployment. The implications of the proposition are given by breaches for each of the requirements of A. Therefore A.1 and A.2 give the following implications:

E. Unsafe key deployment

1. Imposter can register public key for any ID.
2. Public key can be changed in transit.

The proposition for the second breached security goal is Unsafe key retrieval. The implications of the proposition are given by breaches for each of the requirements of B. Therefore B.1, B.2 and B.3 give the following implications:

F. Unsafe key retrieval

1. Some IDs in storage are not accessible.
2. Public key or ID can be changed in storage.
3. Some IDs and public key connections cannot be verified by users.

4.3.2 Contradictions

If E is assumed true in a Blockchain model system running a Blockchain PKI, the following contradictions can be identified:

D.3, D.4 and D.5 together contradicts E.1 (4.4)

D.2 contradicts E.2 (4.5)

Since the Blockchain PKI uses the MAC address of the device as ID and the MAC address is unique and unpredictable for each device, property D.3, it is impossible for an adversary to know what MAC address to register to impersonate another device. The cost, property D.5, prevents an adversary from registering all MAC addresses and since the first MAC is considered valid, property D.4, an adversary cannot later hijack a registered ID. Therefore, E.1 creates a contradiction in the system. This contradiction is defined as contradiction 4.4.

By the properties of the Blockchain system, only the owner of the account can create valid transactions for the account, property D.2. This ensures that any information changed in transit for a transaction would be detected and rejected by the miners. And since the registration of the public key is a transaction, E.2 creates a contradiction in the system. This contradiction is defined as contradiction 4.5.

If F is assumed true while we have a Blockchain based PKI the following contradictions can be identified in the system:

D.1 contradicts F.1 (4.6)

D.1 contradicts F.2 (4.7)

D.1, D.3 and D.4 together contradicts F.3 (4.8)

Public key and ID connections are stored in the distributed public ledger, property D.1, and all connected devices hold a local copy of the distributed ledger. Therefore, it is not possible that registered IDs are not accessible and F.1 creates a contradiction in the system. This contradiction is defined as contradiction 4.6.

The same property of the Blockchain PKI, property D.1, ensures that there is an immutable public directory. Therefore, it is not possible to change a stored public key or ID. This contradiction is defined as contradiction 4.7.

Since all devices have a unique and unpredictable MAC address, property D.3, and the first registered public key for a MAC address is considered valid, property D.4, it is easy for all users with access to the public ledger to verify ID and public key connections. And since the

connections are stored in the distributed public ledger, D.1, all users will have access to the ledger. Therefore, F.3 creates a contradiction in the system. This contradiction is defined as contradiction 4.8.

The contradictions 4.4, 4.5, 4.6, 4.7 and 4.8 shows that the propositions representing a breach of the security goals leads to contradictions for a Blockchain model system using a Blockchain PKI. Therefore, the security goals must hold for the Blockchain PKI in the Blockchain model.

4.4 Blockchain PKI smart contract

Based on the Blockchain PKI, defined in section 4.1, and the functional requirements, defined in section 4.1.1, the Blockchain PKI was implemented using a smart contract. The contract allows devices with accounts in the Ethereum network to register a unique MAC address together with a public key and to query for a public key connected to a MAC address. The PKI also stores the address of the account used to register creating a connection between the account and the MAC address. The PKI contract can be deployed by any account in the system. See listing 4.1 for the code of the contract.

```
1  pragma solidity ^0.4.0;
2
3  /* Smart contract for handling PKI through Blockchain */
4  contract PKI {
5
6     /* Struct to differenciate if there is a key */
7      struct keyHolder {
8          bool isSet;
9          string publicKey;
10         address account;
11     }
12
13     /* Keys stored in the PKI */
14     mapping(string => keyHolder) keys;
15
16     /* Add new key if MAC is unused */
17     function addKey(string mac, string pubKey) public {
18         require(!keys[mac].isSet);
19         keys[mac] = keyHolder(true, pubKey, msg.sender);
20     }
21
22     /* Get the public key for a MAC */
23     function getKey(string mac) public constant returns (string) {
24         return keys[mac].publicKey;
25     }
26
27     /* Get the account address for a MAC */
28     function getAccount(string mac) public constant returns (address) {
29         return keys[mac].account;
30     }
31 }
```

Listing 4.1: Blockchain PKI smart contract

A device registers its MAC address and public key by performing a transaction with a call to the addKey function. As parameters for the function call it supplies its MAC address and public key. If the MAC address is previously unused the transaction is mined and the execution is successful. This creates a mapping between the MAC address, the senders account and the submitted public key in the storage of the contract. If a public key is already registered for the MAC address, the transaction is mined but the execution fails, making no changes to the state of the contract.

To query the PKI for a public key tied to a MAC address a function call to the getKey function is performed with the MAC address as a parameter. Given the MAC address is registered in the PKI the function returns a public key. If the MAC address is not registered, the function call returns the empty string. Since the function does not change the state of contract, the function call is performed locally on the device.

To query the PKI for the address of the account that registered a MAC a function call to the getAccount function is performed with the

MAC address as a parameter. Given the MAC address is registered in the PKI the function returns the address of account. If the MAC address is not registered, the function call returns an address with all zeros. Since the function does not change the state of contract, the function call is performed locally on the device.

4.4.1 Storage

Notice that this creates a global data structure that is shared with all the hosts. In order for this to be practically possible adding information to the data structure has to be relatively cheap and lookups virtually free. Luckily the Merkle Patricia trie structure, explained in section 2.6.3, used for storing data on the Ethereum platform allows cheap state changes for the storage. Remember that only the data that is changed and the path to that data will be updated on state changes. These are relatively cheap actions for each node to perform when new devices register. As explained in section 2.6.2, since each node hosts its own database containing the full state of the Ethereum network, looking up values in the Blockchain PKI is practically free.

4.4.2 Blockchain PKI test suite results

All tests specified in the test suite passed so the smart contract should support all the specified functional requirements for the Blockchain PKI.

4.5 Dynamic Access Protocol

This section describes the constructed Dynamic Access Protocol (DAP) and the requirements to run it on the Ethereum platform. Based on three use cases presented for the scenario in section 1.4 the DAP was designed with these properties in mind:

- **Dynamic Authentication -** Devices connected to the Blockchain platform should be able to establish a secure connection.
- **Distributed Logging -** All data requests in the system are logged and auditable in the public ledger.

- **Non-repudiation -** Data requests should be non-reputable in the system.

Dynamic authentication allows consumers to connect to producers and producers to accept multiple consecutive connections. Distributed logging and non-repudiation ensures that owners of producers can monitor and audit the number of data requests from a producer. The properties therefore ensure that the DAP can be used for all use cases listed in section 1.4.

The DAP utilises two smart contracts deployed in the Ethereum network, the Blockchain PKI smart contract, presented in section 4.4, and a new access control smart contract. Before presenting the DAP the access control smart contract is defined.

4.5.1 Access control

Workings from the definition of producers and consumers in section 1.4, the purpose of the access control is to regulate and log access requests to a producer. The access control should only allow one consumer access rights to a producer at a time and a producer should only exchange data with consumers granted access by the access control. Therefore, the minimum functional requirements for the access control were:

1. **Request access -** Consumer should be able to request access to a producer.
2. **Retrieve access holder -** Producer should be able to retrieve access holder.
3. **Reset access holder -** Producer should be able to reset the access holder.

4.5.2 Access control smart contract

The access control smart contract implements the access control, described in section 4.5.1, on the Ethereum network. The contract stores information related to the access granted consumer, so the producer can retrieve the consumers public key in the Blockchain PKI. After that, any mutual authentication protocol can be used to establish a secure connection between the devices. Each producer deploys its own

access control smart contract to the Ethereum network. This enables reserving certain rights of the contract to only being accessible by the producer. See listing 4.2 for the code of the contract.

```
pragma solidity ^0.4.0;

/* Access control contract for a producer */
contract AccessContract {

    /* Marks if the producer is already occupied */
    bool occupied;
    /* Storage for the consumer ID */
    string consumerID;
    /* Storage for consumer address */
    address consumerAddress;
    /* Address for the owner of the contract, the producer */
    address owner;

    /* Modifier to check if caller is the owner of the contract */
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    /* Event to transmit an allert to the log */
    event Alert(address producerAddress);

   /* Constructor for the contract, sets the creator as owner and initiate
       values */
    function AccessContract() public {
        owner = msg.sender;
        occupied = false;
    }

   /* Destroys contract and sends all funds to owner */
    function kill() public onlyOwner {
        selfdestruct(owner);
    }

    /* Allows consumers to set their key for access */
    function requestAccess(string id) public returns (bool){
        require(!occupied);
        occupied = true;
        consumerID = id;
        consumerAddress = msg.sender;
        emit Alert(owner);
    }

    /* Shows if the producer is occupied or not */
    function isOccupied() public constant returns (bool) {
        return occupied;
    }

    /* Allows producer to retrieve ID for consumer */
    function getConsumerID() public onlyOwner constant returns (string) {
        return consumerID;
    }

    /* Allows producer to retrieve ID for consumer */
    function getConsumerAddress() public onlyOwner constant returns (address)
         {
        return consumerAddress;
    }

    /* Allows producer to reset the key and set status for last transmission
        */
    function reset() public onlyOwner {
        occupied = false;
        delete consumerID;
        delete consumerAddress;
    }
}
```

Listing 4.2: Access control smart contract

After the contract is deployed to the Ethereum network by the producer, the producer is set as the owner of the contract and the producer's address is stored in the contract. This is important since the address of incoming transactions are compared to the owner for some functions of the contract. Upon initialisation the status of the access control is set to not occupied so that the contract will accept incoming access requests.

A consumer requests access to a producer by performing a transaction with a call to the requestAccess function. As parameters for the function call it supplies its ID in the PKI. If the producer is not occupied the transaction is mined and the execution is successful. The contract then stores the consumers ID along with its account address and sets the producer's status to occupied. The contract also emits a log alert that the producer can look for to notice changes in its access control contract. If the producer is already occupied the transaction is mined but the execution fails, making no changes to the state of the contract. The consumer can also check if the producer is occupied by making a function call to isOccupied.

Once an ID and address for a consumer is stored in the contract the consumer can query the contract for that information. To get the ID the producer makes a function call to getConsumerID, the contract will then return the stored ID. To get the address of the account used to request access the producer makes a function call to getConsumerAddress, the contract then returns the stored address. These functions will only return an answer if they are called by the owner of the contract, that is the producer that deployed the contract. In any other case, getConsumerID returns an empty string and getConsumerAddress returns an all zero address. Note that the values of the consumer address and consumer ID might be reachable through another method since they are stored in the distributed ledger. The purpose of limiting the function calls is to encourage correct use of the smart contract.

When the producer wants to accept a new connection, it can reset its status and clear the stored consumer ID and address by making a transaction with a call to the reset function. When the transaction is executed it will mark the status of the producer as not occupied, remove the stored consumer ID and address. The contract is then ready to accept a new access request attempt. This function is also only accessible for the owner of the contract, that is the producer that deployed the contract.

The producer can remove the access control contract from the Ethereum network by performing a transaction with a call to the kill function. When the transaction is executed the contract will be destroyed and if any funds are stored in the contract, they will be transferred to the producer. This function is also only accessible for the owner of the contract.

4.5.3 Storage

Since the access control contract only stores four values for the producer the storage size required for each contract is constant. Upon state changes in the contract the Merkle Patricia trie structure, explained in section 2.6.3, allows that only the affected path and values are updated. This makes changes to the state of the contract effective actions for nodes in the network. All lookups are done in the producers' local storage which are free function calls for the producer.

4.5.4 Access control test suite results

All tests specified in the test suite passed so the smart contract should support all the specified functional requirements for the access control.

4.5.5 Protocol structure

After the producer and consumer are both registered in the PKI and the producer has published an access control smart contract, the consumer can initiate contact to retrieve information from the producer. The protocol is defined by the following seven steps, illustrated in figure 4.1:

1. The Consumer sends a request to the producer that it wishes to establish a connection.

2. The producer answers with its ID, registered in the Blockchain PKI, and the address to the access control smart contract deployed in the Ethereum network.

3. The consumer requests access to the producer through the smart contract. This is repeated until access is granted.

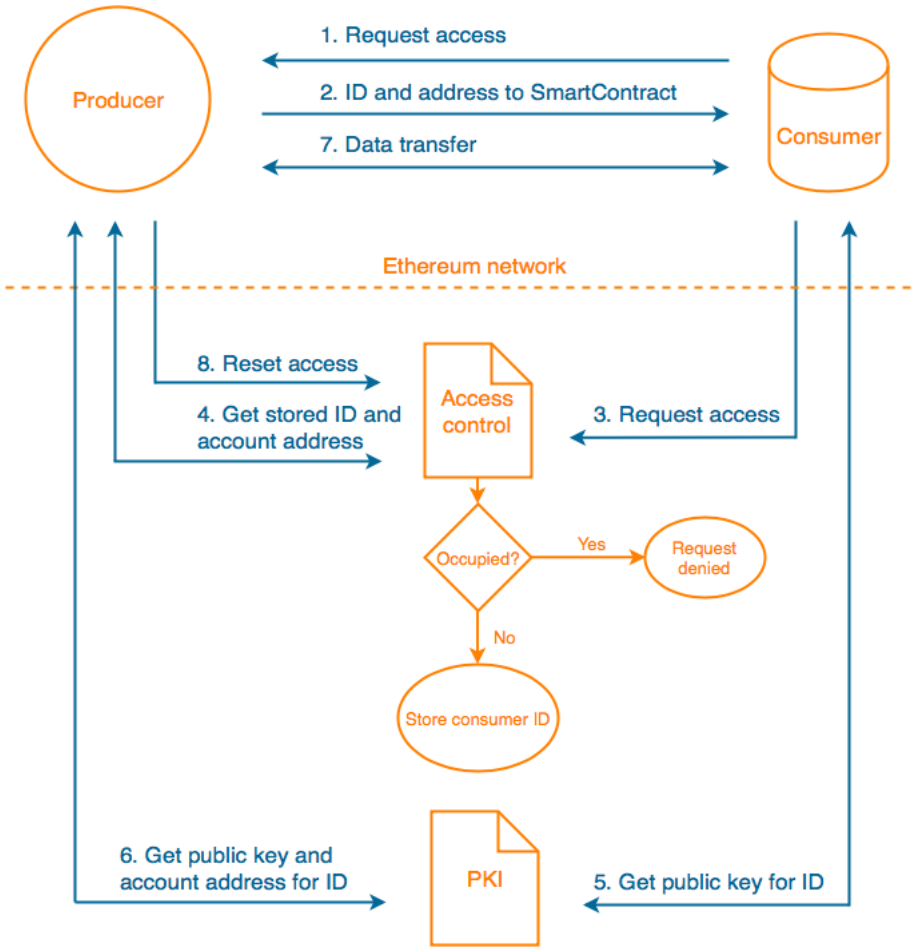

Figure 4.1: Dynamic access protocol

4. The producer queries its access control smart contract for the next consumer to accept a data transfer from and retrieves its ID and account address.

5. The consumer retrieves the public key for the producer through the Blockchain PKI.

6. The producer verifies the account address towards the ID in the Blockchain PKI and retrieves the consumers public key.

7. The producer and the consumer shares data through a public and private key mutual authentication protocol, such as SSH.

8. The producer resets its access control smart contracts so that a

new consumer can be granted access.

Since the devices use the Blockchain PKI to retrieve their respective public key, new devices can connect in a safe way. This ensures the first property, dynamic authentication, of DAP.

By managing all access rights through transactions in the Ethereum network, a complete transaction log is constructed in the Blockchain and distributed to all parties. The transaction log can then be used to monitor what access requests were granted. This ensures the second property, distributed logging, of DAP.

Since all access requests are essentially transactions in the Ethereum network, access requests cannot be done by other parties than the account holder. By having the producer verify the account and ID connection in the PKI, all access requests in the system had to be performed by the owner of the ID and are non-repuidable. This ensures the third property, non-repudiation, of DAP.

Chapter 5

Discussion

This chapter contains a discussion about the results of the report, feasibility for using the results and identifies future work and accomplishments in the field of Blockchain technology.

5.1 Research question and project goals

By defining the properties of a simple Blockchain PKI, section 4.1, and constructing a new CPA model influenced by the Dolev-Yao model, section 4.2, the Blockchain PKI could be evaluated in an informal analysis towards a set of security goals for PKIs, section 4.3. The conclusion of the informal analysis was that the security goals for PKIs where assured in the Blockchain PKI showing that it is cryptographically safe to distribute public keys in public Blockchain ledgers thus answering the research question. The successful construction and implementation of the Blockchain PKI smart contract also supported the theoretical conclusions by practical results, section 4.4. By successfully implementing and testing the Blockchain PKI, PG1 was reached.

By complementing the Blockchain PKI with a successfully implemented and tested access control, section 4.5.2, devices could exchange public keys and regulate their connections through the Blockchain network. By having the access requests being regulated by a smart contract, all state changes of the contract were logged in the distributed ledger. This created an immutable record of successful access requests distributed to all parties. Combining this together in the DAP, section 4.5, ensured that all use cases in the scenario, section 1.4, could be fulfilled and thus PG2 was reached.

5.2 Research methods

The confidence in the results of the research project is defined by the validity of the research methods used. This section discusses the methods used by the research project from a critical perspective.

5.2.1 Model construction

Because the CPA model used for the analysis of the Blockchain PKI was constructed by the author of this paper, this greatly influences the credibility of the results. By proving the models equivalence, when the PKI security goals hold, through an informal analysis to the Dolev-Yao model, this has to be regarded when the results of the report are used. In order to preserve the full credibility of the Dolev-Yao model for the Blockchain model the equivalence should be backed by a formal proof.

5.2.2 Informal analysis

To ensure the validity of any cryptographic proof, it is hard to substitute a formal proof for an informal proof. It is often the case that new proofs are first presented informally, where the scientists abstract the intricate details in order to conclude if the new results are of any value and to get a feeling for the structure of the real proof. If the results are valuable those proofs are then backed up with a formal analysis of the results. If no formal analysis can be conducted, it is hard for other scientists to conclude the same results. This pattern is also observable in academic literature, where an informal analysis and the results are often first presented before the formal proof.

This research project can be viewed as the step of determining if the results would be of any value and determining the order of the full formal proof. The results should be extended with a formal proof to gain full credibility.

5.2.3 Proof of concept as verification

By implementing the PKI on the Ethereum network it is shown that Blockchain can be used to construct PKIs. The proof of concept does however not improve the validity of the results from the informal analysis. The proof of concept could be viewed as an induction approach

to the scientific study. The results are true in this specific setting and could be repeated time and time again, but it does not provide any broader truth to the question regarding the security of PKIs on the Blockchain.

5.3 Technological limitations

The project found three main technological limitations for further use of Blockchain in IoT devices. These issues can be addressed from the viewpoint of the Ethereum network.

5.3.1 Ethereum storage demands

Running a full node on for the Ethereum network requires that the whole state of the network is stored on the device. On the 30 of April 2018, the size of the Ethereum database required to sync a full node was approximately 69 GB [8]. Syncing a full node also involves verifying all state transitions on the Blockchain. IoT devices often have very restricted hardware limitations. Therefore, syncing a full node wouldn't be a viable option. The author believes that another node type is a requirement before the Ethereum network can be successfully used in IoT devices.

5.3.2 Ethereum security parameter

When working with the Ethereum network each node has to set its own security parameter in regards to the number of trailing child blocks before a transaction is viewed as irreversible. Disregarding an adversary and just focus on the effects on the node if the security property is wrong two possible scenarios might happen. First is that the nodes registers with the Blockchain PKI and sees its transaction registered but later on the transaction belonged to a deselected chain. The node would believe it has registered and try to point to its place in the Blockchain PKI where another node would be registered with its public key. The second scenario is that the nodes believe its transactions were not registered with the Blockchain PKI and send a second transaction but later on the chain of the first transaction is chosen and both transactions are registered. This would make the node occupy two spaces in the Blockchain PKI. As mentioned in section 2.5.2, the

probability of this happening is almost negligible. Therefore, a security parameter of two should suffice to protect a node from the two scenarios.

Adversary network control	Security parameter
10%	5
15%	8
20%	11
25%	15
30%	24
35%	41
40%	89
45%	340

Table 5.1: Parameters for less than 0.1% adversary state change probability

Taking adversaries into account the requirements for the security parameter is higher. In section 2.5.4, equation 2.3 is presented that allows a node to calculate the probability of an adversary changing the state of the network after a number of trailing child blocks have been created. The problem with the equation is that the node has to make an assumption of how big percentage of the network an adversary might control. Table 5.1 shows the required security parameter depending on the adversaries control of the network to have a less than 0.1% probability of an adversary changing the state of the network [32]. Assuming the correct network influence be an adversary therefore directly influences producers' vulnerability to manipulation.

5.3.3 Ethereum transaction times

At the time this project was conducted, a new block is mined approximately every 15 seconds in the main Ethereum network [9]. Average time before a transaction is mined into a block is about $20 \sim 30$ seconds depending on the load of the network. Assuming that a trail of five blocks are enough to assure that a transaction is included in the main Blockchain this gives an approximate maximum transaction time of 1 minute and 45 seconds. Therefore, it takes about 1 minute and 45 seconds before a consumer can retrieve data from a producer once a

request has been made. This greatly limits the use-case of the protocol to devices without time sensitive data requests. For time sensitive data requests another Blockchain network or platform would have to be used. Because of the many speculated use cases of Blockchain technology, the author believes improved transaction times are necessary for the future use of the technology.

5.4 Ethical considerations

Speculations for what Blockchain Technology can be used for are wide ranging. Some even say that Blockchain might be the most important invention for the digital era since the internet. The author of this paper can only speculate what the end use case for Blockchain technology in the future could be. But since smart contracts can hold currency, it is not unthinkable that smart contract could gain the same juridical rights as a company. For example, IoT devices that are not owned by anyone but finance themselves and the installation of new devices by charging for sensor access with little to no human supervision.

This would probably lead to a paradigm shift in our view of technology and ownership. Instead of always having companies run by people with profit goals, the goals of a company could be coded in a smart contract and executed deterministically. From that perspective, it becomes very important to analyse the security of the underlying technology. Trying to answer questions such as, what would the effect of self-owned IoT devices be? is hard and out of scope for this research project but very important.

Having a deterministic PKI would greatly increase the credibility of the PKIs function. By not having to deal with the human factor, the behaviour of a PKI could be defined in a smart contract and then operate after its instructions without exception. However, since the code of the PKI would be accessible for anyone it is important that such code and the cryptographic methods used are safe.

5.5 Future work

This section gives a brief overview of some possible further research topics encountered during the project.

5.5.1 Physical to digital identity

In the project, it was assumed that the MAC address of a device was enough the ensure the physical identity of a device. In reality, the MAC address of a device is often hard coded and is a weak identity of a device over the internet. To enable devices to register themselves to a PKI some form of strong identification would be required that ties the physical device to a digital key. Possible ways to physically ensure the identify of a device during registration is required for device self-registration.

5.5.2 Light nodes

Light nodes are a new node type currently under construction for the Ethereum network. Light nodes allow devices to participate without verifying the full state of the network. This is done by allowing light nodes to request specific pieces of the Ethereum database from other nodes in the network. Light nodes can still view all new blocks that are mined, but they don't store the complete state of the network. Light nodes are still under development, but the author believes they are a requirement for successfully running the Ethereum network on IoT devices [10].

5.5.3 Security of light nodes

While light nodes are still being developed their fundamental technology build on retrieving the Blockchain state from other nodes in the network. Ensuring that the information retrieved from other nodes is correct is paramount for the use of light nodes. Since light nodes is probably the best solution for running Blockchain on IoT devices, ensuring its security is important.

5.5.4 Denial of service and Blockchain

DoS is one of the hardest form of cyber-attacks to protect against. It is often left out of cryptographic analysis of networked systems because a simple congestion of a network channel could leave a device disconnected completely from a network. Since Blockchain is secured by the distributed contribution to the state it is especially vulnerable to DoS

attacks. Nodes cut off from the network would store and act on an outdated state which an adversary might be able to leverage. The effects of such adversary attacks are still mostly unexplored.

Chapter 6

Conclusions

The project shows that it is safe to distribute public keys in a public Blockchain ledger and that it is possible to implement a secure PKI on Blockchain technology using smart contracts. The project also shows that it is possible to construct protocols with dynamic authentication and distributed logging on Blockchain technology. The informal cryptographic analysis provides a basis for the security of the system but needs to be extended with a formal cryptographic analysis to ensure full credibility for the results. Before Blockchain PKIs can replace traditional CA systems there needs to be a better way of ensuring the physical identities of devices for public key registration. Because of hardware limitations the author believes that the technology is not really viable for IoT devices until other node types are available that reduces the necessary amount of data stored.

Bibliography

[1] July 2015. URL: https://blog.ethereum.org/2015/07/05/on-abstraction/.

[2] URL: https://emercoin.com/en.

[3] URL: https://www.beame.io.

[4] URL: https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial.

[5] URL: https://luxsci.com/blog/understanding-blockchains-part-3-ethereum-moving-beyond-bitcoin.html.

[6] URL: https://solidity.readthedocs.io/en/latest/index.html.

[7] URL: https://github.com/ethereum/go-ethereum/wiki/geth.

[8] URL: https://etherscan.io/chart2/chaindatasizefast.

[9] URL: https://etherscan.io/chart/blocktime.

[10] URL: https://github.com/ethereum/wiki/wiki/Light-client-protocol.

[11] Michael Backes, Birgit Pfitzmann, and Michael Waidner. "Soudness Limits of Dolev-Yao Models". In: *Workshop on Formal Computational Cryptography*. 2006.

[12] A. Banafa. *Three Major Challenges Facing IoT*. Mar. 2017. URL: https://iot.ieee.org/newsletter/march-2017/three-major-challenges-facing-iot (visited on 01/17/2018).

[13] Mustafa Al-Bassam. "SCPKI: A Smart Contract-based PKI and Identity System". In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. BCC '17. Abu Dhabi, United Arab Emirates: ACM, 2017, pp. 35–40. ISBN: 978-1-4503-4974-1. DOI: 10.1145/3055518.3055530. URL: http://doi.acm.org/10.1145/3055518.3055530.

[14] J. A. Berkowsky and T. Hayajneh. "Security issues with certificate authorities". In: *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*. 2017, pp. 449–455.

[15] Daniel J. Bernstein, Tanja Lange, and Ruben Niederhagen. "Dual EC: A Standardized Back Door". In: *The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*. Ed. by Peter Y. A. Ryan, David Naccache, and Jean-Jacques Quisquater. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 256–281. ISBN: 978-3-662-49301-4. DOI: 10.1007/978-3-662-49301-4_17. URL: https://doi.org/10.1007/978-3-662-49301-4_17.

[16] Vitalik Buterin. *Ethereum White Paper*. Tech. rep. ethereum.org, 2013.

[17] K. Christidis and M. Devetsikiotis. "Blockchains and Smart Contracts for the Internet of Things". In: *IEEE Access* 4 (2016), pp. 2292–2303.

[18] Hubert Comon and Vitaly Shmatikov. "Is it possible to decide whether a cryptographic protocol is secure or not". In: *Journal of Telecommunications and Information Technology* (2001).

[19] Christian Decker and Roger Wattenhofer. "Information Propagation in the Bitcoin Network". In: *13-th IEEE International Conference on Peer-to-Peer Computing* (2013).

[20] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[21] Danny Dolev and Andrew C. Yao. "On the security of Public Key Protocols". In: *IEEE Transactions on Information Theory* (1983).

[22] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. *CertCoin: A NameCoin Based Decentralized Authentication System*. Tech. rep. Massachusetts Institute of Technology, 2014.

[23] Anna Marie Helmenstine. *Number of Atoms in the Universe*. 2017. URL: https://www.thoughtco.com/number-of-atoms-in-the-universe-603795 (visited on 02/01/2018).

[24] Don Johnson, Alfred Menezes, and Scott Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)". In: *International Journal of Information Security* 1.1 (Aug. 2001), pp. 36–63. ISSN: 1615-5262. DOI: 10.1007/s102070100002. URL: https://doi.org/10.1007/s102070100002.

[25] Audun Jøsang. "PKI Trust Models". In: *Theory and Practices of Cryptography Solutions for Secure Information Systems*. IGI Global, 2013.

[26] Jonathan Katz. *Digital Signatures*. 1st ed. Springer US, 2010.

[27] N. Kshetri. "Can Blockchain Strengthen the Internet of Things?" In: *IT Professional* 19.4 (2017), pp. 68–72. DOI: 10.1109/MITP.2017.3051335.

[28] Roetteler Martin et al. "Quantum resource estimates for computing elliptic curve discrete logarithms". In: *AISACRYPT* (2017).

[29] H. Massias, X. Serret Avila, and J.-J. Quisquanter. "Design Of A Secure Timestamping Service With Minimal Trust Requirements". In: *the 20th Symposium on Information Theory in Benelux*. 1999.

[30] Victor S. Miller. "Use of Elliptic Curves in Cryptography". In: *Advances in Cryptology —CRYPTO '85 Proceedings*. Ed. by Hugh C. Williams. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 417–426. ISBN: 978-3-540-39799-1.

[31] Julien Mineraud et al. "A gap analysis of Internet-of-Things platforms". In: *Computer Communications* 89-90 (2016). Internet of Things Research challenges and Solutions, pp. 5–16. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2016.03.015. URL: http://www.sciencedirect.com/science/article/pii/S0140366416300731.

[32] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008. URL: https://bitcoin.org/bitcoin.pdf (visited on 01/18/2018).

[33] Moni Naor. "On Cryptographic Assumptions and Challenges". In: *Advances in Cryptology - CRYPTO 2003*. Ed. by Dan Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 96–109. ISBN: 978-3-540-45146-4.

[34] Aissam Outchakoucht, Hamza Es-samaali, and Jean Philippe Leroy. "Dynamic Access Control Policy based on Blockchain and Machine Learning for the Internet of Things". In: *International Journal of Advanced Computer Science and Applications* (2017).

[35] Lawrence C. Paulson. *The Inductive Approach to Verifying Cryptographic Protocols*. Tech. rep. University of Cambridge, 2000.

[36] Alfredo Pironti and Riccardo Sisto. "Soundness Conditions for Cryptographic Algorithms and Parameters Abstractions in Formal Security Protocol Models". In: *Proceedings of International Conference on Dependability of Computer Systems, DepCoS - RELCOMEX 2008*. July 2008, pp. 31–38. ISBN: 978-0-7695-3179-3.

[37] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: http://doi.acm.org/10.1145/359340.359342.

[38] P. W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[39] National Institute of Standards and Technology. "Digital signature standard". In: *FIPS Publication 186* (1994).

[40] Douglas R. Stinson. *Cryptography, Theory and Practice, Thid eddition*. Ed. by Kenneth H. Rosen. 3rd ed. Chapman & Hall/CRC, 2006.

[41] N. Szabo. *Smart Contracts: Building Blocks for Digital Markets*. URL: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html (visited on 01/22/2018).

[42] Gavin Wood. *Ethereum: A secure decentralised generalised transactions ledger*. Tech. rep. ethcore, 2014.

[43] Yuanyau Zhang et al. “Smart Contract-Based Access Control for the Internet of Things”. In: *ArXiv e-prints* (2018).

Appendix A

PKI node scripts

For all scripts the binary code of the contract has been removed from the pkiOutput line produced by the compiler.

```
1  var PASSWORD = "";
2
3  var pkiOutput={"contracts":{"PKI.sol:PKI":{"abi":"[{\"constant\":true,\"
       inputs\":[{\"name\":\"mac\",\"type\":\"string\"}],\"name\":\"getAccount
       \",\"outputs\":[{\"name\":\"\",\"type\":\"address\"}],\"payable\":false
       ,\"stateMutability\":\"view\",\"type\":\"function\"},{\"constant\":false
       ,\"inputs\":[{\"name\":\"mac\",\"type\":\"string\"},{\"name\":\"pubKey
       \",\"type\":\"string\"}],\"name\":\"addKey\",\"outputs\":[],\"payable\":
       false,\"stateMutability\":\"nonpayable\",\"type\":\"function\"},{\"
       constant\":true,\"inputs\":[{\"name\":\"mac\",\"type\":\"string\"}],\"
       name\":\"getKey\",\"outputs\":[{\"name\":\"\",\"type\":\"string\"}],\"
       payable\":false,\"stateMutability\":\"view\",\"type\":\"function\"}]","
       bin":""}},"version":"0.4.21+commit.dfe3193c.Darwin.appleclang"};
4  var pkiContractAbi = pkiOutput.contracts['PKI.sol:PKI'].abi;
5  var pkiContract = eth.contract(JSON.parse(pkiContractAbi));
6  var pkiBinCode = "0x" + pkiOutput.contracts['PKI.sol:PKI'].bin;
7  personal.unlockAccount(eth.accounts[0], PASSWORD);
8  var deployTransactionObject = { from: eth.accounts[0], data: pkiBinCode, gas:
        1000000};
9  var pkiInstance = pkiContract.new(deployTransactionObject);
10
11 var i = 0;
12 while (!eth.getTransactionReceipt(pkiInstance.transactionHash) &&Â i < 10){
13     console.log('Transaction not mined: ' + i);
14     admin.sleep(1);
15     i++;
16 }
17
18 var pkiAddress = eth.getTransactionReceipt(pkiInstance.transactionHash).
       contractAddress;
19 console.log('Transaction mined and stored at address: ' + pkiAddress);
20
21 var pki = pkiContract.at(pkiAddress);
```

Listing A.1: PKI deploy script

```
1 var PKIADDRESS = "0xb209325f9a145acf66c2bfd0a8344196653e3d1f";
2
3 var pkiOutput={"contracts":{"PKI.sol:PKI":{"abi":"[{\"constant\":true,\"
    inputs\":[{\"name\":\"mac\",\"type\":\"string\"}],\"name\":\"getAccount
    \",\"outputs\":[{\"name\":\"\",\"type\":\"address\"}],\"payable\":false
    ,\"stateMutability\":\"view\",\"type\":\"function\"},{\"constant\":false
    ,\"inputs\":[{\"name\":\"mac\",\"type\":\"string\"},{\"name\":\"pubKey
    \",\"type\":\"string\"}],\"name\":\"addKey\",\"outputs\":[],\"payable\":
    false,\"stateMutability\":\"nonpayable\",\"type\":\"function\"},{\"
    constant\":true,\"inputs\":[{\"name\":\"mac\",\"type\":\"string\"}],\"
    name\":\"getKey\",\"outputs\":[{\"name\":\"\",\"type\":\"string\"}],\"
    payable\":false,\"stateMutability\":\"view\",\"type\":\"function\"}]","
    bin":""}},"version":"0.4.21+commit.dfe3193c.Darwin.appleclang"};
4 var pkiContractAbi = pkiOutput.contracts['PKI.sol:PKI'].abi;
5 var pkiContract = eth.contract(JSON.parse(pkiContractAbi));
6 var pki = pkiContract.at(PKIADDRESS);
```

Listing A.2: PKI access script

Appendix B

AC node scripts

For all scripts the binary code of the contract has been removed from the acOutput line produced by the compiler.

```
1  var PASSWORD = "";
2
3  var acOutput={"contracts":{"AccessContract.sol:AccessContract":{"abi":"[{\"
       constant\":false,\"inputs\":[{\"name\":\"id\",\"type\":\"string\"}],\"
       name\":\"requestAccess\",\"outputs\":[{\"name\":\"\",\"type\":\"bool
       \"}],\"payable\":false,\"stateMutability\":\"nonpayable\",\"type\":\"
       function\"},{\"constant\":false,\"inputs\":[],\"name\":\"kill\",\"
       outputs\":[],\"payable\":false,\"stateMutability\":\"nonpayable\",\"type
       \":\"function\"},{\"constant\":true,\"inputs\":[],\"name\":\"isOccupied
       \",\"outputs\":[{\"name\":\"\",\"type\":\"bool\"}],\"payable\":false,\"
       stateMutability\":\"view\",\"type\":\"function\"},{\"constant\":true,\"
       inputs\":[],\"name\":\"getConsumerID\",\"outputs\":[{\"name\":\"\",\"
       type\":\"string\"}],\"payable\":false,\"stateMutability\":\"view\",\"
       type\":\"function\"},{\"constant\":true,\"inputs\":[],\"name\":\"
       getConsumerAddress\",\"outputs\":[{\"name\":\"\",\"type\":\"address
       \"}],\"payable\":false,\"stateMutability\":\"view\",\"type\":\"function
       \"},{\"constant\":false,\"inputs\":[],\"name\":\"reset\",\"outputs
       \":[],\"payable\":false,\"stateMutability\":\"nonpayable\",\"type\":\"
       function\"},{\"inputs\":[],\"payable\":false,\"stateMutability\":\"
       nonpayable\",\"type\":\"constructor\"},{\"anonymous\":false,\"inputs
       \":[{\"indexed\":false,\"name\":\"producerAddress\",\"type\":\"address
       \"}],\"name\":\"Alert\",\"type\":\"event\"}]","bin":""}},"version":"
       0.4.21+commit.dfe3193c.Darwin.appleclang"};
4  var acContractAbi = acOutput.contracts['AccessContract.sol:AccessContract'].
       abi;
5  var acContract = eth.contract(JSON.parse(acContractAbi));
6  var acBinCode = "0x" + acOutput.contracts['AccessContract.sol:AccessContract'
       ].bin;
7  personal.unlockAccount(eth.accounts[0], PASSWORD);
8  var deployTransactionObject = {from: eth.accounts[0], data: acBinCode, gas:
       1000000};
9  var acInstance = acContract.new(deployTransactionObject);
10
11 var i = 0;
12 while (!eth.getTransactionReceipt(acInstance.transactionHash) && i < 10){
13     console.log('Transaction not mined: ' + i);
14     admin.sleep(1);
15     i++;
16 }
17
18 var acAddress = eth.getTransactionReceipt(acInstance.transactionHash).
       contractAddress;
19 console.log('Transaction mined and stored at address: ' + acAddress);
20
21 var ac = acContract.at(acAddress);
```

Listing B.1: Access control deploy script

```
1 var ACADDRESS = "0xb209325f9a145acf66c2bfd0a8344196653e3d1f";
2
3 var acOutput={"contracts":{"AccessContract.sol:AccessContract":{"abi":"[{\"
    constant\":false,\"inputs\":[{\"name\":\"id\",\"type\":\"string\"}],\"
    name\":\"requestAccess\",\"outputs\":[{\"name\":\"\",\"type\":\"bool
    \"}],\"payable\":false,\"stateMutability\":\"nonpayable\",\"type\":\"
    function\"},{\"constant\":false,\"inputs\":[],\"name\":\"kill\",\"
    outputs\":[],\"payable\":false,\"stateMutability\":\"nonpayable\",\"type
    \":\"function\"},{\"constant\":true,\"inputs\":[],\"name\":\"isOccupied
    \",\"outputs\":[{\"name\":\"\",\"type\":\"bool\"}],\"payable\":false,\"
    stateMutability\":\"view\",\"type\":\"function\"},{\"constant\":true,\"
    inputs\":[],\"name\":\"getConsumerID\",\"outputs\":[{\"name\":\"\",\"
    type\":\"string\"}],\"payable\":false,\"stateMutability\":\"view\",\"
    type\":\"function\"},{\"constant\":true,\"inputs\":[],\"name\":\"
    getConsumerAddress\",\"outputs\":[{\"name\":\"\",\"type\":\"address
    \"}],\"payable\":false,\"stateMutability\":\"view\",\"type\":\"function
    \"},{\"constant\":false,\"inputs\":[],\"name\":\"reset\",\"outputs
    \":[],\"payable\":false,\"stateMutability\":\"nonpayable\",\"type\":\"
    function\"},{\"inputs\":[],\"payable\":false,\"stateMutability\":\"
    nonpayable\",\"type\":\"constructor\"},{\"anonymous\":false,\"inputs
    \":[{\"indexed\":false,\"name\":\"producerAddress\",\"type\":\"address
    \"}],\"name\":\"Alert\",\"type\":\"event\"}]","bin":""}},"version":"
    0.4.21+commit.dfe3193c.Darwin.appleclang"};
4 var acContractAbi = acOutput.contracts['AccessContract.sol:AccessContract'].
    abi;
5 var acContract = eth.contract(JSON.parse(acContractAbi));
6
7 var ac = acContract.at(ACADDRESS);
```

Listing B.2: Access control access script