# The Use of Block Chain Technology in Different Application Domains

Bachelor Project in Software Development

The IT University of Copenhagen

Jacob Stenum Czepluch - jstc@itu.dk

Nikolaj Zangenberg Lollike - nlol@itu.dk

Simon Oliver Malone - soma@itu.dk

20th May 2015

## Abstract

In this paper, the exciting possibilities that block chain technology offers in regards to decentralised trust-free systems are investigated. More specifically this includes research of how block chain technology can advantageously be utilised in different domains, from finance to more general societal applications.

On the basis of a small trust-based coffee shop, a proof of concept system has been developed as a base point for an evaluation of the strengths and weaknesses of the block chain technology. Clearly both are present, but they are much dependent on which cases the technology is applied to. In the example of a coffee shop, the low maintenance, built in security and ease of implementation are factors that speak for the utilisation. On the other hand the inconvenience of currency conversion and transaction time are drawbacks. On a more general scale the security and trust-freeness of the technology is definitely features that allow for it to be applied in a broad spectrum of applications. However, scalability, costs and fluctuating currencies are hindrances.

It is argued that block chain technology has the potential to restore trust in the banking sector by introducing a level of transparency.

The technology is still young and suffers from teething troubles, but it is argued that as it matures it will have a great impact in many areas of application.

# Contents

# List of Figures

# Acknowledgement

We would like to thank the very open minded and welcoming communities surrounding and supporting different cryptocurrencies and block chain technologies.

A special thanks to the Ethereum development community for answering all of our questions and helping us getting up and running as quickly as possible. And doing so while being very busy getting Ethereum ready for the Frontier release.

# 1 | Introduction

Even though block chain technologies, sparked by the introduction of Bitcoin in 2009, have existed for about 6 years, we are still in the dawn of their utilisation. Innovative solutions that have potential societal impact are still being researched and discovered every day. In light of recent years' decrease of consumer trust in the banking sector, block chains might be a solution in order to rectify this.

So far, block chains have mostly been used to power *cryptocurrencies*. The Ethereum Foundation has kick-started a new development in this area by implementing a generic programmable block chain that is applicable in a wide variety of applications. This paves the way of utilising the advantageous features such as its trust-free, transparent and highly secure nature, in many other application areas than just economic systems. IBM and Samsung are even experimenting with using the Ethereum block chain to power the *Internet of Things (IoT)*. Block chain technology is a promising technology we might see used more in the future, as it is currently under the radar of many entities in the financial sector, why it is interesting to explore the realm of the new possibilities they offer.

This bachelor thesis will investigate the potential of the Ethereum technology, and the overall application areas, where decentralisation of organisations and applications through block chain technology can be useful.

More specifically the focal point of this paper is to investigate how currently trust-based centralised systems in the financial sector and society in general, can benefit from becoming decentralised and trust-free. This is done through evaluating the proof of concept implementation of a digitalised punch card on the Ethereum block chain.

## 1.1 Research question

How can decentralisation through block chain technology be applied beneficially in economic systems and society in general?
Sub-questions in order to answer the main research question:

- How does the Ethereum block chain work?
- How can a decentralised application be implemented using the Ethereum block chain?
- What are the benefits and drawbacks of block chain technology in general?

## 1.2 Contents of This Paper

This section will give a brief overview of the chronological order of the topics we will cover in this paper in order to answer the research questions in section 1.1.

In order to understand how decentralised systems based on block chains work, a background of current research and a thorough account of the technology will be conducted. By implementing a conceptualisation of a decentralised application, for the student driven coffee shop at the *IT University of Copenhagen (ITU)*, we will prove the potential of decentralised applications.

On the basis of this implementation we will analyse the benefits and drawbacks that characterise decentralised systems powered by block chain technology. The results of the analysis will lay the foundations of discussing the features of Ethereum and block chains on a general scale, and an assessment of how this technology can be utilised in finance, government and society in general.

## 1.3 Cryptographic Economic Systems

In 2009 Bitcoin introduced a brand new form of currency. But for a software developer, the revolutionary technology that made this possible is much more fascinating: *the block chain*. Bitcoin is a decentralised *cryptocurrency* which runs trust-free, autonomous, publicly and anonymously. All this is possible thanks to the block chain.

Bitcoin is fascinating due to it being the first big *Decentralised Autonomous Organisation (DAO)*. An organisation that runs autonomously due to the rules of transactions being dictated by the block chain protocol, and decentralised due it being maintained by a peer-to-peer network. The fact that big organisations can run this way changes the landscape of organisations, corporations and their applications. As mentioned in the beginning, this technology is still in its dawn of being applied in other domains. Therefore this is a very interesting topic to investigate since it has the possibility of changing an entire ecosystem of transaction systems.

## 1.4 Why Ethereum?

So why not use the well-established Bitcoin in order to investigate this area?

Ethereum is specifically built to support transactions on a general scale through the use of *smart contracts*. Ethereum essentially offers "a featureless block chain", that supports a Turing-complete programming language to write smart contracts in. It is thus a completely programmable block chain that distributes logic that would normally be run on a centralised server. While the Bitcoin block chain has been used before in order to support other than monetary transactions, it is more intriguing to dive into a platform that was specifically built to do this, why Ethereum was settled upon as a focal point of this paper.

# 2 | Methodology

This chapter covers the methodologies that are used in order to investigate and answer the research questions in section 1.1.

## 2.1 Design Science Research

In order to investigate our research questions we use the *Design Science Research (DSR)* approach, more specifically the adaptation developed by V. Vaishnavi and B. Kuechler[52]. This process revolves around having a problem that can be investigated by designing an artefact in order to solve this problem. The artefact is then evaluated in order to discuss and reflect upon whether or not the problem has been solved. This methodology is very similar to the suggested guidelines of DSR by Hevner[32] and we will make use of both of these publications.

## 2.2 Phases of DSR

The DSR approach splits the process of research into multiple phases as we can see in figure 2.1.

#### Awareness of Problem

The process starts with an interesting problem that is often sparked by new developments in the industry. For example you have knowledge of a system that has issues that now can be improved upon. In this case there is a problem of trust and transparency in transaction systems. A specific case is the self-

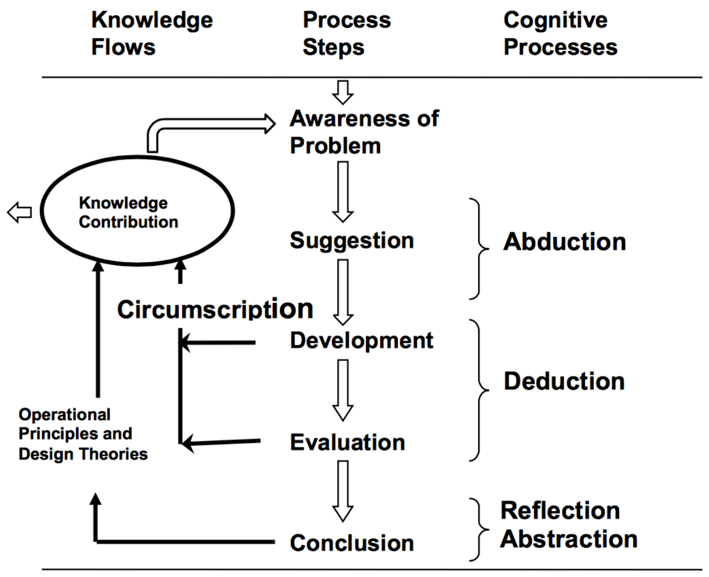


Figure 2.1: Cognitive processes used in Design Science Research as suggested by Vaishnavi and Kuechler[52].

service transaction system in our coffee shop at the university. Developments in decentralisation through block chain technology can seemingly be utilised in some way here.

### Suggestion

A proposal of change is made based upon research and theory in the field, and specifications are made for the artefact that is going to be designed. In this case, through researching the ecosystem of *cryptographic economic systems* and block chain technology the suggestion is to implement a digitalised *punch card* system built on the Ethereum block chain.

### Development

The artefact is designed and implemented. Our digitalised punch card will be realised. The back-end of smart contracts are coded and issued, and a client for users to manipulate these is implemented.

**Evaluation**

The success of the artefact in terms of solving the research problem is analysed and evaluated through different evaluation methods such as testing, experimenting and observation[32].

**Conclusion**

The end of the research cycle. The results of our research is accounted for, which can be used as basis for more research.

## 2.3 Reflections on Research

In this report we are using DSR and our account of the technology to give an educated opinion on the prospects of this technology. More specifically in which use cases decentralisation of applications can prove useful and its societal impact.

We are thus looking at a specific case to analyse the technology in detail in order to upheave this discussion to a more general level. Consequently we have this extra process layer of reflection in our methodology.

# 3 | Background

This chapter will give you some background knowledge of the research and developments in cryptocurrencies and block chain technology so far, in order to give a brief overview of the landscape of this.

## 3.1 Bitcoin: the First DAO

In the original white paper on Bitcoin from 2008, Satoshi Nakamoto has a vision of a new decentralised economic system:

> *"A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution."*[39]

This vision was realised through the implementation of Bitcoin, which also is the first example of a decentralised autonomous organisation (DAO) – an organisation that runs completely autonomously, decentralised, transparent and secure thanks to the underlying block chain[39]. Decentralised because it runs purely peer-to-peer through nodes of users on the network. Transparent because the transactions are public and the network open to everyone. Autonomously thanks to protocols dictating the rules of transactions on the network. And secure due to all nodes on the network verifying the transactions through consensus algorithms. All these perks together are the cornerstones of a *trust-free transaction system.*

## 3.2 Block Chains in Finance

The perks of the block chain are very interesting in finance where transparency, trust and security in transactions are vital[23]. The block chain has thus been used to develop many economical systems and cryptocurrencies such as Bitcoin and thousands of *altcoins*, such as Litecoin[1], Dogecoin[2] and Nxt[3]. Cryptocurrencies are especially handy because you do not need a middleman such as a bank to ensure trust and security in transactions[11, 23, 39]. This is autonomously dealt with by the block chain protocol[11, 39], which consequently leads to transaction fees being much lower than that of a bank[15].

Cryptocurrencies are also handy in microtransactions since you can pay much smaller fractions of a cryptocurrency than a *fiat currency*[15]. Additionally these digital currencies are useful in terms of the global market since your transactions go directly from peer to peer, avoiding banks. This means you can send money to anywhere in the world only waiting for a block to be generated. In Bitcoin's case this is about 10 minutes[39], which is much faster than a transaction through multiple banks, and much less costly due to the fees such an international transaction would have.

The use of block chains have been generalised to trade more than only money. Digital assets such as shares, contracts and stock options have been traded as *smart property* on the block chain. Every object that you have a right to can be emulated as a smart property. A protocol that does this is *Mastercoin*[4]. Mastercoin uses meta data through what they call an *omnilayer* that allows for the creation of digital assets to issue and trade as smart properties on the Bitcoin block chain. Thus economic systems, and trading in general, benefit from this trust-free, secure and transparent transaction system. The latest development in this area is that NASDAQ is experimenting with the block chain in order to automate transaction processes that are now handled by lawyers[44].

[1]`https://litecoin.org/da/` (visited on 16/05/2015)
[2]`http://dogecoin.com` (visited on 16/05/2015)
[3]`http://nxt.org` (visited on 16/05/2015)
[4]`http://www.omnilayer.org` (visited on 14/05/2015)

## 3.3 Re-Decentralising the Internet

In the recent year, there has been a shift in the research area of the block chain[19]. Secure and transparent transactions are applicable in many other use cases than economic systems.

In terms of today's web, most traffic go through a few gigantic nodes of corporations such as Google, Facebook, Amazon etc. The internet today is much more centralised than 10 years ago due to the commercialisation of the web, why today we have corporations essentially monopolising and owning parts of the world wide web[51]. There has been a shift in power from the users to the corporations. We trust all our data in the hands of big corporations, whose processes are very intransparent[35]. In addition government surveillance programs such as the *National Security Agency (NSA)* are tapping into this data without the users knowing[29, 51].

Ethereum envisions to decentralise applications in what they dub a *Web 3.0*, where applications and their data are run through block chain technology, thus allowing for a decentralised, transparent and secure internet. This would shift the power back to the users in a "redemocratisation" of the internet, as stated on their homepage[5]. Transparency would thus give users the power to see exactly how their data is handled, and that NSA is not secretly listening in on your conversations[51].

## 3.4 From the Internet of Things to the Economy of Things

The amount of electronic devices in the world is increasing rapidly[9]. With the introduction of smart devices in the late 2000s, computers have found themselves being implemented in virtually anything. There are computers in cars, our domestic appliances and our house systems. The next step in development is that these smart objects can communicate with each other over the internet and essentially maintain themselves. This development in

[5]`http://ethereum.org` (visited on 16/05/2015)

technology is called the *Internet of Things*[9, 41]. IBM, in collaboration with Samsung, is building a block chain powered technology dubbed *ADEPT*[33, 41] on the Ethereum protocol, to support this network of devices interacting with each other through smart contracts. They give this example of how IoT can be utilised:

> *"We demonstrate how, using ADEPT, a humble washer can become a semi-autonomous device capable of managing its own consumables supply, performing self-service and maintenance, and even negotiating with other peer devices both in the home and outside to optimize its environment."*[33]

These transactions between smart objects will develop an *Economy of Things* where all these smart devices will be "a point of transaction and economic value creation for owners and users"[41]. These devices will thus be able to engage in multiple markets, both financial and non-financial and autonomously react to changes in said markets.

## 3.5 Future Research

The research and appliance of block chain technology is still in its dawn. How this technology will affect society and where future research might be important, are what we will try to uncover in this report.

# 4 | Account for Technology

In this section we will explain what Ethereum is and why exactly this technology is extraordinarily interesting in regards to cryptocurrencies and a decentralised web.

## 4.1 Ethereum

Ethereum is one of many technologies emerged from the original block chain technology used by Bitcoin. The original Ethereum white paper[11] written by Vitalik Buterin in late 2013 describes the basic idea behind a new and improved way of using the decentralised block chain technology. In 2014 Gavin Wood formally described the technology in the Ethereum yellow paper[53]. The platform is supposed to be released in 2015, but at the time of writing[6] it is still in a development state.

Etherum takes the block chain technology to the next level by allowing stateful user-created digital smart contracts to be executed by implementing a generic programmable block chain. It is not solely a network for monetary transactions like Bitcoin, but a network where transactions can be used to execute an unlimited variety of smart contracts. This block chain is the back-end of decentralised applications, or as Ethereum dubs them: *ĐApps*.

Some examples of applications that Ethereum can be used for are: voting systems, domain name registries, financial exchanges, crowdfunding platforms, company governance, intellectual property and smart property.

[6] 16/05/2015

## 4.2 Vision

According to the Ethereum white paper the reasoning for making Ethereum is to

> " *. . . create an alternative protocol for building decentralized applications, providing a different set of tradeoffs that we believe will be very useful for a large class of decentralized applications, with particular emphasis on situations where rapid development time, security for small and rarely used applications, and the ability of different applications to very efficiently interact, are important.*"[11]

Ethereum does this by building what they call the "ultimate abstract foundational layer: a blockchain with a built-in Turing-complete programming language"[11]. This basically means that Ethereum allows everyone to write smart contracts (described in section 4.4.1) resulting in decentralised applications where it is possible to build your own arbitrary rules for ownership, transaction formats, and state transition functions. This relatively simple way of creating your own smart contracts, adding them to the block chain, and using them, creates a great environment to potentially solve some paramount issues in regards to some of the mentioned examples in section 4.1, where a centralised authority is trusted. One of the main points of Ethereum and other technologies using a block chain is that they are trust-free and decentralised.

## 4.3 Block Chain Technology

The block chain is essentially just what the name says: a chain of blocks. A block contains the data of all transactions within a period of time, and a reference to the block before it (see figure 4.1). The cryptography that goes into creating a block differs depending on what block chain protocol is used,

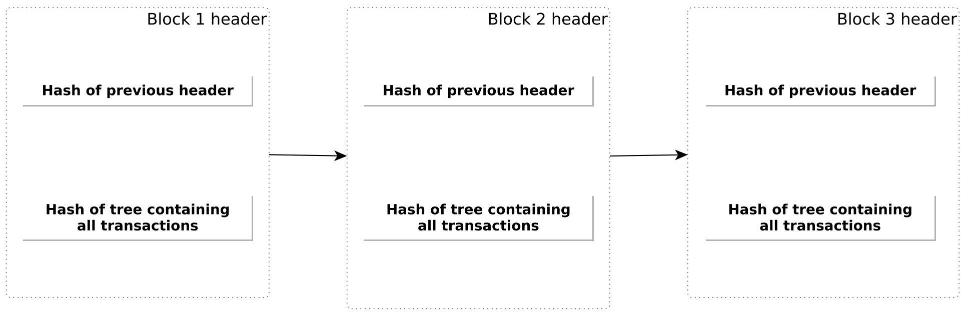

Block 1 header
Hash of previous header
Hash of tree containing all transactions
Block 2 header
Hash of previous header
Hash of tree containing all transactions
Block 3 header
Hash of previous header
Hash of tree containing all transactions

Figure 4.1: Simplified version of how Bitcoin's block chain connects blocks

but essentially you can traverse through the entire block chain and find every single transaction ever made. Hashing algorithms are used to make sure that all blocks are well formed and not tampered with, and thus the block chain keeps itself secure and virtually unbreakable.

The above is one defining feature of block chain technology, another is that it is public and decentralised. The block chain is not run from a single server, it is run on a widespread network of computers. They all hold all data in the block chain, and all work on expanding it. These computers are called miners (see section 4.3.1). Depending on the block chain protocol, these will compete to form new blocks that are then added to the block chain when selected through consensus schemes (see section 4.3.2).

The combination of the two features stated above is what makes the block chain desirable. It is secure through the immutability it gains from the hashing algorithms, when applied on the decentralised network and transparent because anyone can look through all blocks. The combination of this security and transparency is what makes the block chain a trust-free technology.

In the past block chains have been compared to bank ledgers[6, 28]. However, this definition somewhat limits the block chain artificially, as the general idea behind it can be used in a much more general sense. Transactions are not limited to monetary transferrals, but can also be used to transfer any form of data. How this data is then used, again depends on what block chain protocol it is sent on.

In the remainder of this section, an account for the general features of the block chain will be made before focusing specifically on the Ethereum protocol.

### 4.3.1 Mining

To incline people to use their processing power to carry the system, the block chain protocols will reward “miners” in different ways. The more miners that run a node on their hardware, the more secure the system is.

The way you get rewarded as a miner on a block chain is by creating a block that is selected to be a part of the block chain. Therefore all miners will try to create blocks (automatically done by the mining software), so the system has to select which block gets put into the chain. But since there is no central node that can make decisions, all the nodes have to agree, and that too is done by a consensus scheme[11]. When all, or most of, the nodes can automatically agree on a block, it is easy to carry on. Any node that tries to select a different block will automatically be disregarded by the rest of the system, and it is therefore impossible to cheat the block chain, unless you can circumvent the consensus scheme (see below).

### 4.3.2 Consensus

When deciding on a block, there needs to be consensus in the block chain system. This is done by the protocol’s consensus scheme, which dictates the way all nodes can agree that a miner has a right to include its block into the block chain. On Bitcoin and the current[7] version of Ethereum, the scheme is called *proof of work*. In this particular scheme, the goal is to show that the miner has done a certain amount of work that entitles it to the block reward. In the case of Bitcoin, this is done by requiring the miner to find a string, that when concatenated with a hash of the previous block header and then hashed, returns a string with a certain amount of preceding zeroes. Ethereum’s approach is different, and will be covered in section 4.4.5.

[7] 18/05/2015

The proof of work algorithm mathematically ensures security on the block chain, as long as one single entity does not hold majority of the computing power. Only then will the illegitimate blocks, added by this entity, actually be added to the block chain, as that entity creates more than half the new blocks. The fact that the proof of work is made from the hashed header from the previous block, and that headers contain a hash of all transactions in that block (see figure 4.1), changing an old transaction requires the perpetrator to recalculate the proof of work for all subsequent blocks. It then has to convince the system to use this chain by continuously adding blocks to this chain at a higher rate than the legitimate chain.

Another consensus protocol is called *proof of stake*. This is a consensus protocol that is based on the distribution of cryptocurrency within the system. Proof of stake is the consensus algorithm that Ethereum aims to use, as it does not require the waste of resources that proof of work does[31]. It has, however, been argued that proof of stake cannot be used as a consensus scheme in its pure form[12, 43]. Unless specified, this paper assumes that proof of work is used, when discussing the block chain technology.

## 4.4 Technology of Ethereum

### 4.4.1 Smart contracts

What differentiates Ethereum from most other block chain technologies, is how it utilises smart contracts.

A smart contract is a piece of code that the *Ethereum Virtual Machine (EVM)* (described in section 4.4.6) is able to execute on the block chain. Once this piece of code has been added to the block chain, the smart contract itself cannot be altered, only the storage of the smart contract can. This means that there now exists some piece of code that acts as a contract and that is available for anyone to use. As mentioned, the execution of these smart contracts are made possible by the Turing-complete programming languages that are compiled into EVM bytecode. These languages are, as of May 2015, Solidity[8] (Java like), Serpent[9] (Python like), Mutan[10] (C like), and LLL[11] (Lisp like).

A smart contract has an *address*, just like an external account (essentially a user), but instead of acting like a wallet, it will execute code based on the data it receives. Smart contracts can call other smart contracts in almost the same way, through *messages*, which will be explained in section 4.4.4.

In order to avoid malicious and infinitely looping code or *distributed denial of service attacks* to be executed on the block chain, each execution and creation of a smart contract is powered by gas (see section 4.4.3). Powered means that the amount of gas needed to run the contract is determined by the total amount of computations and storage entries of the byte code that the EVM compiles the smart contract into.

Even though it is estimated that the total amount of computing power available directly on the block chain is equivalent to that of a 1999 smartphone[10], smart contracts are extremely powerful in the way they function as the building blocks of world wide decentralised trust-free systems.

### 4.4.2 Transactions

A transaction on the Ethereum block chain contains 6 fields; the recipient of the transaction (be it a user or a smart contract), the signature of the sender and the amount of *ether* to send with the transaction. It also has an optional data field, a start gas value and a gas price value. The first three fields are self explanatory, it is the final three that make Ethereum stand out. The data field can be used to transfer parameters to a contract. The remaining two are explained in section 4.4.3.

[8]`https://github.com/ethereum/wiki/wiki/Solidity-Tutorial` (visited on 16/05/2015). This resource states that it is JavaScript like, however, as JavaScript is not strongly typed, or object oriented, we believe that it is more Java like than JavaScript like.

[9]`https://github.com/ethereum/wiki/wiki/Serpent` (visited on 16/05/2015)

[10]`https://github.com/obscuren/mutan` (visited on 16/05/2015)

[11]`https://github.com/ethereum/cpp-ethereum/wiki/LLL` (visited on 16/05/2015). As of this date, it is written in a non-English language, but a commit from 09/04/2014 by Gavin Wood show the text in English.

### 4.4.3 Ether and gas

Ethereum uses its underlying network unit, ether, to fuel the contracts executed on the block chain. Fueling is understood as a way to pay for the execution of EVM bytecode and storage of data on the block chain. How much a specific computation will cost is defined by the complexity of the computation. The most basic computations such as addition, subtraction, and multiplication is intented to cost 1 gas. Contracts and transactions also have a start price that is fixed in order to pay the miner for his computational power. The EVM will on compilation of the code of the smart contract or transaction, decide the amount of computations and multiply them by the current gas price, thus determining the price of executing the desired transaction:

$$price_{transaction} = (computations \times price_{gas}) + price_{start}$$

Since Ethereum is still in an early testing stage, the real gas price has not yet been decided, and is therefore, as of May 2015, fixed at 10 szabo. 1 szabo is $10^{-6}$ ether, which means that for 1 ether you can get 100,000 computations[25]. The gas price will however be changed when the first non-testing block chain gets launched[12].

### 4.4.4 Messages

Messages are essentially transactions between two smart contracts. They carry almost the same fields as a regular transaction, except there is no gas price. The gas used by the sum of all computations done by all smart contracts activated by a transaction, must be covered in the gas field of said transaction[11].

[12]`http://ether.fund/tool/gas-price` (visited 19/05/2015)

### 4.4.5 Mining ether

There are some differences in the way the Ethereum block chain and the Bitcoin block chain works in terms of mining. Each block on the Ethereum block chain contains the entire *state* of the Ethereum system, stored in a "Patricia tree", which is an evolved "Merkle tree" as known from Bitcoin. It only stores the data of new transactions, whilst unchanged data is stored as pointers to the original block location[11].

Also the way a miner is selected to create a block on the block chain is different to prevent some of the centralisation found in Bitcoin's block chain. Bitcoin requires the miner to do a specific hash function on some strings, which means that it is possible to use *application-specific integrated circuits (ASICs)*, which is already being done. This makes it almost impossible for the regular user to have a chance to mine. The block validation is now done in big mining pools relying on this technology. This is even worse, since at the time of writing, the three biggest mining pools hold over 50% of the computing power and previously Ghash.io have been close to it as well[5, 40].

Ethereum aims to use a model where a miner will have to process random transactions from previous blocks and hash the result. This means that, as transactions can compute smart contracts that can hold any kind of computations, the utilisation of ASICs are not possible in the same way[11]. Secondly, a miner has to have the entire block chain available, so mining pools cannot be used in the same way. The only way of bypassing this is by "poisoning the well" by including a large number of contracts designed to cater to specific ASICs, and that is a problem that are yet to be addressed by Ethereum at the time of writing[11].

### 4.4.6 Ethereum Virtual Machine (EVM)

The Ethereum block chain can be described as a block chain with a built-in programming language or as a consensus based globally executed virtual machine[10]. What takes care of the internal state and the computations is called the Ethereum Virtual Machine (EVM). The EVM can be thought of as a large decentralised computer containing millions of objects called "accounts". Accounts have the ability to maintain an internal database, execute code and talk to other accounts. In section 4.4.1 we describe smart contracts. A smart contract is itself an account. *Externally owned accounts (EOAs)* are controlled by a private key, and if you own the key belonging to the EOA you have the ability to send ether and messages from the EOA.

Computations in the EVM are done in a stack-based bytecode language. An EVM program is a sequence of *opcodes*. Basically the EVM works in the same way as many other virtual machines. It takes some programming language and compiles it into low level code that the computer, on which it runs, understands. Due to this, developers do not have to write low level code, but can use one of the high level languages designed for writing contracts mentioned in section 4.4.1. Any code written in these languages needs to be compiled into EVM bytecode before being sent in a transaction with no recipient to create a smart contract.

### 4.4.7 Block time

The block time is the amount of time it takes for the network to assign a new block to the block chain, thus confirming pending transactions. Ethereum has a block time of 12 seconds, but there is no guarantee of the amount of time it actually takes for the network to assign a new block. It is just a value the network aims to hit, and it does this by adjusting the *block difficulty*[18].

The block difficulty is regulated each time a new block is found. If a block is found "too quickly", that being less than 12 seconds, the difficulty of finding the next block goes up. If it takes too long time to find a block, the block time consequently goes down.

The difficulty is defined by how many hashes on average it takes to create a new block. Thus if the difficulty is high, the harder it is to find a hash within the certain goal that satisfies the *proof of work*-algorithm.

The reason Ethereum has a shorter block time of 12 seconds compared to Bitcoin's 10 minutes is that contrary to the monetary transactions of Bitcoin, the Ethereum block chain is used in a wide variety of applications where transactions need to be more responsive[18].

# 5 | Proof of Concept

This chapter will cover the implementation of our proof of concept. An explanation of the current system and the implementation of a new system will be given.

## 5.1 The Concept

To test out the potential of decentralisation and the Ethereum block chain, we use our local university coffee shop *Analog* as our case. The idea is that we want to implement a digitalised punch card for coffee using smart contracts.

The reason this case is chosen is that it is currently a trust-based system with inexplicit transactions rules. The features of the block chain seem fitting in this case, due to being able to make the transaction process trust-free through dictating the transaction rules.

### 5.1.1 Current system

At this point in time, the way the "real life contract" works is that you buy a card that is worth 10 cups of coffee for 50 DKK. Each time you want a cup of coffee you deduct one clip using a scissor. Then you fill up your cup.

The punch card is an example of an artefact that we have given a certain value in the real world. The punch card in it self is worthless. The materials of the punch card are worth next to nothing. It only has a value when used in a transaction in a specific place. Thus, the idea of the punch card is that you can buy coffee without the need of a cleric. It is up to the user of the punch card to deduct the correct amount of clips when buying a cup of coffee. This way you can get coffee at all times through self-service.

### 5.1.2 The issue of trust

There are however some problems using this method. This method only works through trusting the users. And to trust the users, they have to be well informed about how the system works. For example: which kind of beverages a punch card gains access to and how long the punch card is valid and so on. They have to acknowledge the rules of the "real life contract".

In the current system there is no way of actually ensuring that the users deduct the correct amount of clips when purchasing coffee – or even pay at all, unless there is a cleric, which ruins the point of the system. The reason users deduct the wrong amount of clips can both be malicious behaviour, but also misunderstandings of the system. Using a smart contract as described in section 4.4.1 could seemingly prevent these problems.

So how would one go about implementing a decentralised system like this?

## 5.2 Implementation

This section will cover in detail the process of implementing the smart contract using the programming language Solidity and client using the Ethereum JavaScript API.

### 5.2.1 Smart contract

In order to obtain the desired functionality, a smart contract needs to be implemented to dictate the rules of the system. This section will explain how the code functions. The entire source code can be found in appendix A.1.1.

First of all functionality is needed to issue punch cards for users to purchase. In order to distribute this smart property a function called `buyClipcard` is implemented, which allows for a user to purchase a punch card for a certain price. The price can be adjusted by the issuer of the smart contract to whatever amount you like through a function called `setPrice`. When purchasing a punch card, the contract will make sure you have a sufficient balance of ether before you can commit the transaction. To keep track of users and their balances, a map of user addresses on the network and their `clipcard` property, called `clipcards`, is implemented. This way it is possible to check if people own a punch card or has sufficient clips.

After this functionality it is needed to dictate the rules of buying a cup of coffee, using the punch card token. This part of the contract is basically a "vending machine" in the sense that a product has a fixed price. A function that deducts one clip from your punch card is therefore added as `useClip`.

In order for this to be fully automated, you would have to implement a digital lock on the coffee dispensers. Otherwise you would not solve the problem of people just stealing coffee. To implement a hardware solution is out of scope for the system, since it is the aim of this paper to implement a proof of concept on how the software should be specified. However a system like this could be implemented, by having a lock on the dispensers connected to the network. A smart contract could then be implemented to turn this lock on and off, as this smart contract would be invoked by the system. Being able to invoke hardware through block chain technology is a small example of IoT (see section 3.4).

A fully fledged automated system could thus be implemented with two smart contracts in conjunction with each other.

In order to issue these smart contracts onto the network you make a transaction with the smart contract logic as described in section 4.4.6. As stated in section 4.4.1, the price of fueling this contract is determined by how many operations it includes. If we have enough ether to fuel the smart contract it will be submitted to the block chain and be accessible through an address.

### 5.2.2 JavaScript client

Now that the implementation of the back-end functionality has been covered, it is desirable to use our smart contracts in an easier way than through command line transactions. This is possible by using the Ethereum JavaScript API in order to build an HTML client.

**Ease of use**

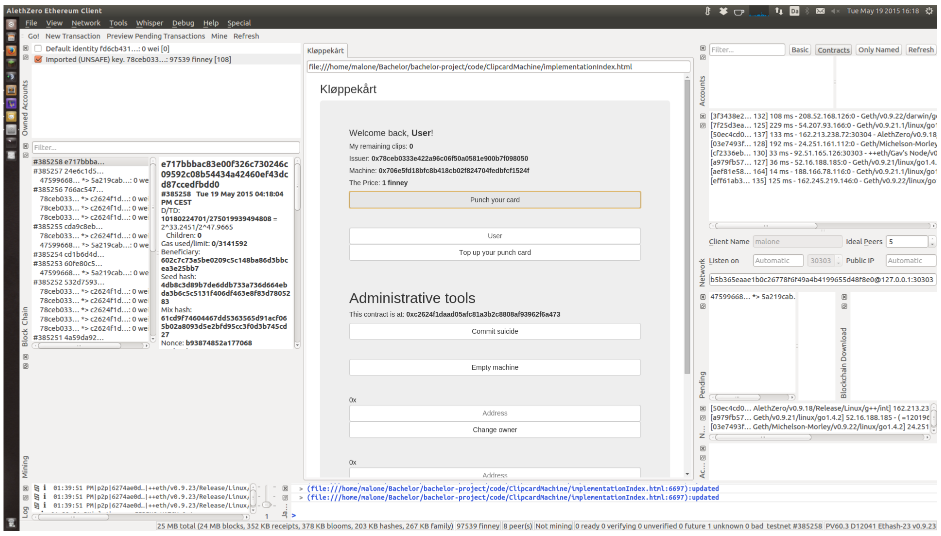
The role of the client is basically to make the smart contract easier to use. It will display stored information and make functions accessible through a *graphical user interface (GUI)* that interacts with the contract itself. Thus, instead of making specific transactions with data to the smart contract, input fields and buttons have been made to take care of the job for you, as seen in figure 5.2. This is done by using a JavaScript API provided by Ethereum that utilises *web3.js*[26]. The `web3` object makes it possible to communicate with the network because it knows your private address. Therefore you can only use decentralised applications (ÐApps) in a browser that utilises `web3`, for example *Mist* or *AlethZero*, which are both clients developed by Ethereum. The JavaScript API offers a lot of functionality that can be used to communicate with a smart contract and the Ethereum block chain. Namely we utilise the function of getting stored data out of the contract to display in our client and calling functions to manipulate this data. The source code of our client can be found in A.1.2.

## 5.3 Tour of the System

In order to access the client of our decentralised application, you need a browser that support Ethereum based applications. Currently there are two clients which are *AlethZero*[13] and *Mist*[14].

This section will take the reader on a tour through the usage of our system, while explaining what goes on under the hood. In this example the client, AlethZero as seen in figure 5.1 is used to showcase our application.

[13]`https://github.com/ethereum/cpp-ethereum/wiki` (visited on 18/05/2015)
[14]`https://github.com/ethereum/go-ethereum/wiki` (visited on 18/05/2015)



Tue May 19 2015 16:18

Default identity fd6cb431…: 0 wei [0]
Imported (UNSAFE) key. 78ceb033…: 97539 finney [108]

```
#385258 e717bbba…
#385257 24e6c1d5…
   47599668… *> 5a219cab…: 0 we
#385256 766ac547…
   78ceb033… *> c2624f1d…: 0 wei
   78ceb033… *> c2624f1d…: 0 wei
   78ceb033… *> c2624f1d…: 0 wei
#385255 cda9c8eb…
   78ceb033… *> c2624f1d…: 0 wei
   47599668… *> 5a219cab…: 0 we
#385254 cd1b6d4d…
#385253 60fe80c5…
   47599668… *> 5a219cab…: 0 we
#385252 532d7593…
   78ceb033… *> c2624f1d…: 0 wei
   78ceb033… *> c2624f1d…: 0 wei
   78ceb033… *> c2624f1d…: 0 wei
   78ceb033… *> c2624f1d…: 0 wei
   78ceb033… *> c2624f1d…: 0 wei
#385251 4a59da92…
```

```
e717bbbac83e00f326c730246c
09592c08b54434a42460ef43dc
d87ccedfbdd0
#385258   Tue 19 May 2015 04:18:04
PM CEST
D/TD:
10180224701/275019939494808 =
2^33.2451/2^47.9665
  Children: 0
Gas used/limit: 0/3141592
Beneficiary:
602c7c73a5be0209c5c148ba86d3bbc
ea3e25bb7
Seed hash:
4db8c3d89b7de6ddb733a736d664eb
da3b6c5c5131f406df463e8f83d78052
83
Mix hash:
61cd9f74604467dd5363565d91acf06
5b02a8093d5e2bfd95cc3f0d3b745cd
27
Nonce: b93874852a177068
```

file:///home/malone/Bachelor/bachelor-project/code/ClipcardMachine/implementationIndex.html

Kløppekårt

Welcome back, **User**!

My remaining clips: **0**

Issuer: **0x78ceb0333e422a96c06f50a0581e900b7f098050**

Machine: **0x706e5fd18bfc8b418cb02f824704fedbfcf1524f**

The Price: **1 finney**

Administrative tools

This contract is at: **0xc2624f1daad05afc81a3b2c8808af93962f6a473**

0x

0x

```
[3f3438e2… 132] 108 ms - 208.52.168.126:0 - Geth/v0.9.22/darwin/g
[7f25d3ea… 125] 229 ms - 54.207.93.166:0 - Geth/v0.9.21.1/linux/go
[50ec4cd0… 137] 133 ms - 162.213.238.72:30304 - AlethZero/v0.9.18
[03e7493f… 128] 192 ms - 24.251.161.112:0 - Geth/Michelson-Morley
[cf2336eb… 130] 33 ms - 92.51.165.126:30303 - ++eth/Gav's Node/v0
[a979fb57… 127] 36 ms - 52.16.188.185:0 - Geth/v0.9.21/linux/go1.4.
[aef81e58… 164] 14 ms - 188.166.78.116:0 - Geth/v0.9.21.1/linux/go
[eff61ab3… 135] 125 ms - 162.245.219.146:0 - Geth/v0.9.22/linux/go
```

Client Name: malone  Ideal Peers: 5
Listen on: Automatic  30303  Public IP: Automatic
b5b365eaae1b0c26778f6f49a4b4199655d48f8e0@127.0.0.1:30303

47599668… *> 5a219cab.

```
[50ec4cd0… AlethZero/v0.9.18/Release/Linux/g++/int] 162.213.23
[a979fb57… Geth/v0.9.21/linux/go1.4.2] 52.16.188.185 - ( =12019
[03e7493f… Geth/Michelson-Morley/v0.9.22/linux/go1.4.2] 24.251
```

```
01:39:51 PM|p2p|6274ae0d…|++eth/v0.9.23/Release/Linux/
01:39:51 PM|p2p|6274ae0d…|++eth/v0.9.23/Release/Linux/
01:39:51 PM|p2p|6274ae0d…|++eth/v0.9.23/Release/Linux/
```

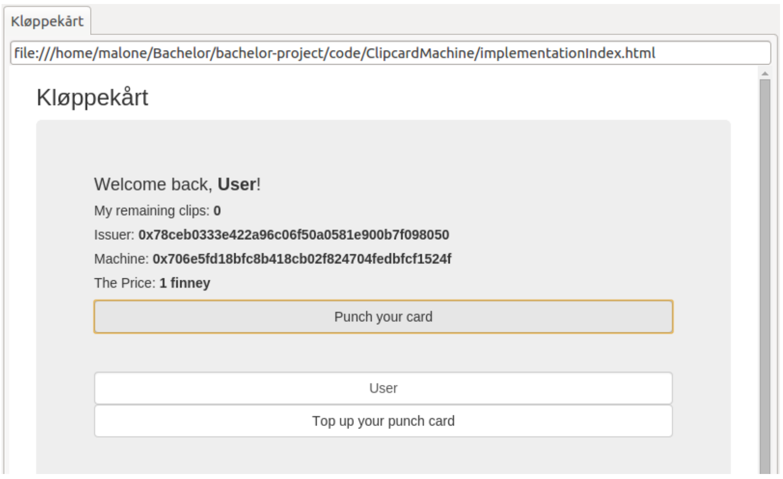
```
> (file:///home/malone/Bachelor/bachelor-project/code/ClipcardMachine/implementationIndex.html:6697):updated
> (file:///home/malone/Bachelor/bachelor-project/code/ClipcardMachine/implementationIndex.html:6697):updated
>
```

25 MB total (24 MB blocks, 352 KB receipts, 378 KB blooms, 203 KB hashes, 267 KB family) | 97539 finney | 8 peer(s) | Not mining | 0 ready 0 verifying 0 unverified 0 future 1 unknown 0 bad testnet #385258 | PV60.3 D12041 Ethash-23 v0.9.23

Figure 5.1: AlethZero – the browser used to view our client and display the state of the network.

### 5.3.1 The welcome screen

Once you are connected to the Ethereum network through a supported browser, your local block chain will be synchronised. Upon data synchronisation you can navigate to `http://malone.dk/clip` in order to access the digitalised punch card. Here you will be greeted by the landing page of our client as seen in figure 5.2



file:///home/malone/Bachelor/bachelor-project/code/ClipcardMachine/implementationIndex.html

Kløppekårt

Welcome back, **User**!

My remaining clips: **0**

Issuer: **0x78ceb0333e422a96c06f50a0581e900b7f098050**

Machine: **0x706e5fd18bfc8b418cb02f824704fedbfcf1524f**

The Price: **1 finney**

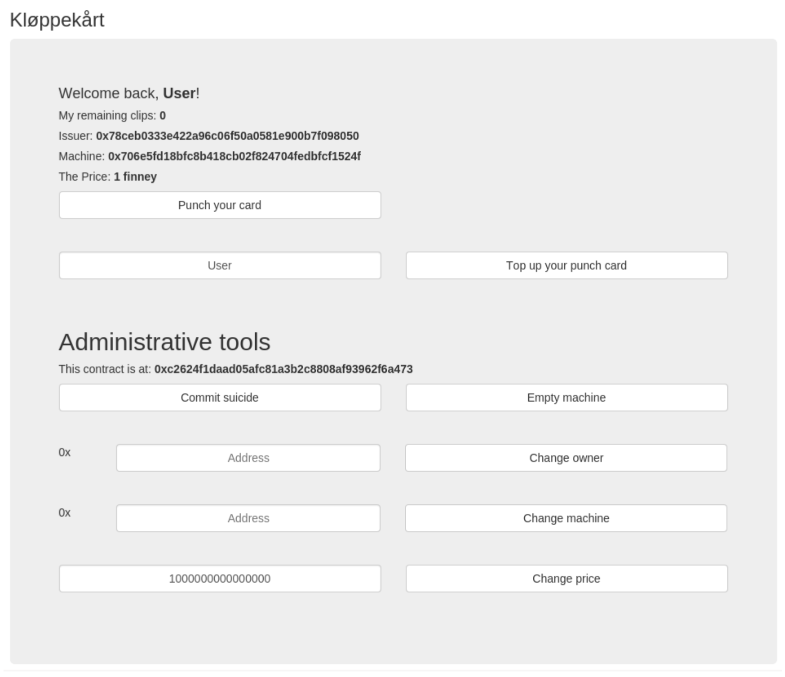
Figure 5.2: The GUI of the HTML/JavaScript client.

All the data displayed in bold are information taken directly out from the block chain, as it is contained in the storage of the smart contract of the implementation.

Since you are connected to the Ethereum network through authenticating with your private key, one of the strong suits of these kind of applications is that there is no need for logging in, since the network already knows who you are.

In this case the client does not know me yet as I have not used the system before. Therefore I am identified as “User”, and have no balance.

In addition the client displays the current price of a punch card; the network address of the issuer of the contract; and the network address of the smart contract that controls the lock on the coffee dispensers. All these values can be updated through an admin panel as seen in figure 5.3 that are immediately accessible to users with admin rights. Consequently there is no admin login either since the data of user rights too is stored on the block chain.


Kløppekårt

Welcome back, **User**!

My remaining clips: **0**

Issuer: **0x78ceb0333e422a96c06f50a0581e900b7f098050**

Machine: **0x706e5fd18bfc8b418cb02f824704fedbfcf1524f**

The Price: **1 finney**

Administrative tools

This contract is at: **0xc2624f1daad05afc81a3b2c8808af93962f6a473**

Figure 5.3: The admin view of the client.

### 5.3.2 Purchasing a punch card

In order to purchase a punch card, you need some amount of ether. In the current implementation the price is set to 1 finney which is 0.001 ether.

The price is set low for testing purposes but as stated above it can be set as you wish. In the end ether essentially has a monetary value. The price could thus be set accordingly to fit the price of the physical punch card or even be controlled by a smart contract that keeps an eye on fluctuations in the market which is discussed in section 7.4.5.

Upon the release of Ethereum, ether can be acquired through exchanges or through mining on the network. At the time of writing[15] Ethereum is still in development, thus “real ether” cannot yet be acquired.

If you have an adequate amount of ether you can purchase a punch card by typing your name in the name input field and click on the “Top up your punch card” button.

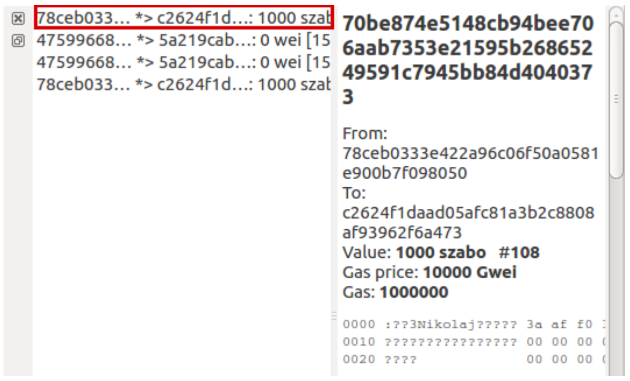
This will initiate a transaction to the smart contract, powering the system, consisting of the payment of 1 finney, your name and that you want to invoke the purchase punch card function on the smart contract. This is all done through the JavaScript of the client, thus the plumbing has been done for you. Your address on the network is the sender of this transaction why the smart contract will map this address as the owner of the punch card.

When using AlethZero you will be able to see the transaction in the *Pending* window as seen in figure 5.4. Here it is seen that the transaction is completely transparent as all its data is publicly visible. Visible in the sense that the computational operations I want carried out in addition to the data I send with it are listed. Whilst waiting for the transaction to go through, a loading screen appears.

Once a new block is discovered the transaction will be submitted to the block chain. As accounted for in section 4.4.7 the block time of Ethereum is 12 seconds. Discussions on the block time can be found in section 7.4.3.

The transaction is thus carried through once it is submitted to the block

[15] 18/05/2015



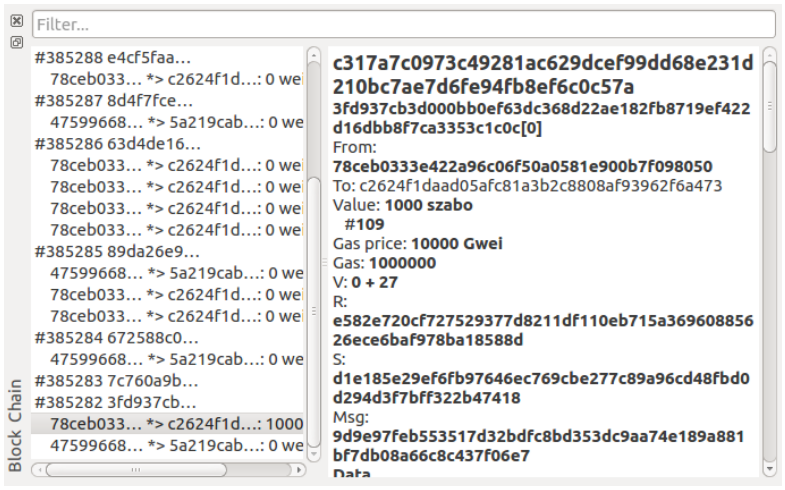
```
78ceb033... *> c2624f1d...: 1000 szab
47599668... *> 5a219cab...: 0 wei [15
47599668... *> 5a219cab...: 0 wei [15
78ceb033... *> c2624f1d...: 1000 szab
```

**70be874e5148cb94bee70
6aab7353e21595b268652
49591c7945bb84d404037
3**

From:
78ceb0333e422a96c06f50a0581
e900b7f098050
To:
c2624f1daad05afc81a3b2c8808
af93962f6a473
Value: **1000 szabo   #108**
Gas price: **10000 Gwei**
Gas: **1000000**

```
0000 :??3Nikolaj????? 3a af f0
0010 ???????????????? 00 00 00
0020 ????             00 00 00
```

Figure 5.4: A pending transaction



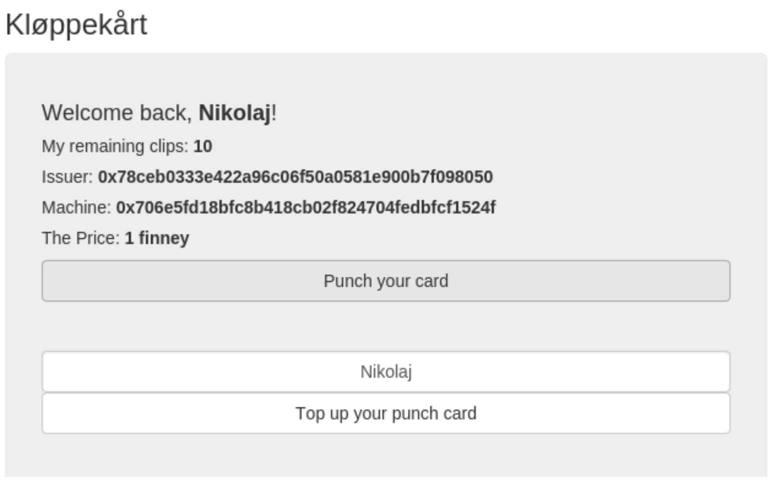
```
#385288 e4cf5faa...
   78ceb033... *> c2624f1d...: 0 wei
#385287 8d4f7fce...
   47599668... *> 5a219cab...: 0 we
#385286 63d4de16...
   78ceb033... *> c2624f1d...: 0 wei
   78ceb033... *> c2624f1d...: 0 wei
   78ceb033... *> c2624f1d...: 0 wei
   78ceb033... *> c2624f1d...: 0 wei
#385285 89da26e9...
   47599668... *> 5a219cab...: 0 we
   78ceb033... *> c2624f1d...: 0 wei
   78ceb033... *> c2624f1d...: 0 wei
#385284 672588c0...
   47599668... *> 5a219cab...: 0 we
#385283 7c760a9b...
#385282 3fd937cb...
   78ceb033... *> c2624f1d...: 1000
   47599668... *> 5a219cab...: 0 we
```

**c317a7c0973c49281ac629dcef99dd68e231d
210bc7ae7d6fe94fb8ef6c0c57a**
**3fd937cb3d000bb0ef63dc368d22ae182fb8719ef422
d16dbb8f7ca3353c1c0c[0]**
From:
**78ceb0333e422a96c06f50a0581e900b7f098050**
To: c2624f1daad05afc81a3b2c8808af93962f6a473
Value: **1000 szabo**
**#109**
Gas price: **10000 Gwei**
Gas: **1000000**
V: **0 + 27**
R:
**e582e720cf727529377d8211df110eb715a369608856
26ece6baf978ba18588d**
S:
**d1e185e29ef6fb97646ec769cbe277c89a96cd48fbd0
d294d3f7bff322b47418**
Msg:
**9d9e97feb553517d32bdfc8bd353dc9aa74e189a881
bf7db08a66c8c437f06e7**
Data

Figure 5.5: The transaction is stored on the block chain

chain. In figure 5.5 we see that the transaction and its data are now stored publicly in a block. The system will now recognise you through your address being mapped in the smart contract. Consequently your name and a balance of 10 clips will be displayed as seen in figure 5.6.

This transaction is now stored forever on the block chain for everyone to see, but unable to be tampered with.


Kløppekårt

Welcome back, **Nikolaj**!

My remaining clips: **10**

Issuer: **0x78ceb0333e422a96c06f50a0581e900b7f098050**

Machine: **0x706e5fd18bfc8b418cb02f824704fedbfcf1524f**

The Price: **1 finney**

Figure 5.6: The transaction has gone through and the system now recognises the user's name and his balance since this data is now stored in the block chain.

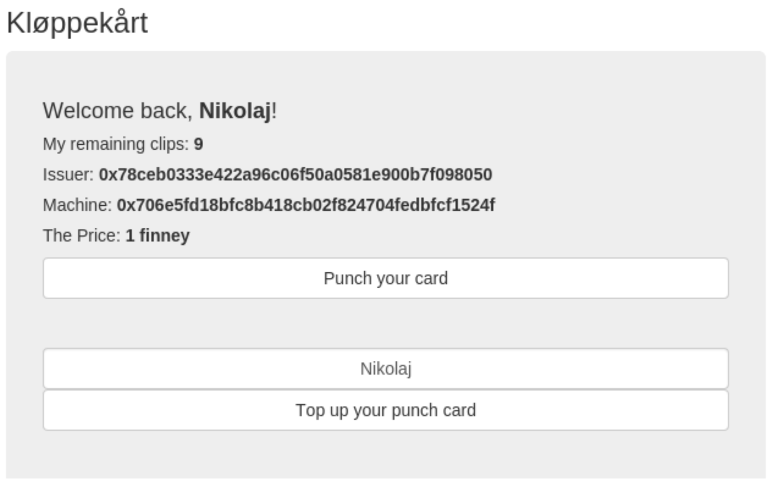
### 5.3.3 Purchasing coffee

If you have a punch card you will have a balance of an amount of clips. The smart contract thus looks in the block chain to determine if your address is mapped to a punch card entity, and checks the balance of this. If you have clips left you are granted access to buying coffee. Consequently the button "Punch your card" in the client will not be greyed out.

To purchase a cup of coffee and thus deducting a clip from your balance, you click said button. Again a pending transaction to the smart contract similar to figure 5.4 will be initiated.

This transaction only contains a method call to invoke the deduct clip function of the smart contract. This method call will deduct your balance by one, while sending a message to the smart contract of the lock on the coffee dispenser to open for a period of time. Once again you wait for a block to be found before the transaction is carried through after which your balance should be updated to be one less as seen in figure 5.7, and the coffee dispenser unlocked.

The block chain is now updated with the data that you have purchased


Kløppekårt

Welcome back, **Nikolaj**!

My remaining clips: **9**

Issuer: **0x78ceb0333e422a96c06f50a0581e900b7f098050**

Machine: **0x706e5fd18bfc8b418cb02f824704fedbfcf1524f**

The Price: **1 finney**

Figure 5.7: After buying a cup of coffee, the balance of the punch card has been updated.

a cup of coffee, just like when a punch card was purchased in figure 5.5, thus concluding the tour of the system.

## 5.4 Modular Implementation

Another way of implementing the proof-of-concept system is to use a modular smart contract system. An implementation of this can be found in appendix A.2. However, it has not been possible to run this successfully on the *testnet*, despite the fact that the code should be working[16].

The modular implementation is build up around modules that each have their own responsibility. The proposed modular implementation is made up of seven smart contracts, with each its own responsibility, and each being interchangeable without it affecting the others.

[16]An administrator on the Ethereum Forum confirmed that the code does not have any obvious problems: `https://forum.ethereum.org/discussion/2136/help-with-contracts-interacting-with-each-other` (visited on 15/05/2015)

### 5.4.1 The contracts

There are seven smart contracts in the modular implementation (and one super smart contract that is not actually pushed to the block chain).

**Issuer** (appendix A.2 line 23) is the smart contract that communicates with the GUI. It is responsible for making the needed information visible to the user (`getBalance`, `getPrice`, `getName`), and handling the input from the user (`buyCard`, `punchCard`). It also handles input from the owner (`setPrice`, `setAddAmount` and `setPunchAmount`).

**AssetHolder** (appendix A.2 line 218) is the smart contract that handles the system's assets (ether and punches/clips). It works as the controlling layer between the *Issuer* and the *PunchDb*. It is responsible for making the *PunchDb* add and deduct the right amount of punches to the punch card, transferring ether to the owner or the sender depending on the event. It can also "unpunch" a punch card, should the *Machine* fail.

**Settings** (appendix A.2 line 326) is the database for storing settings. The settings are stored in pairs of keys of type `bytes32` (a string) and values of type `uint`. Functions for setting and getting settings are also available. This way, the settings module can store an arbitrary amount of settings, though those used in this system are "price", "addAmount" and "punchAmount".

**NameDb** (appendix A.2 line 399) is the database for storing names. It stores names in pairs of addresses and strings, and functions for setting and getting names are available.

**PunchDb** (appendix A.2 line 350) is the database responsible for storing the amount of punches available to the users. It stores the information in pairs of addresses and integers (as `uint`). Functions are available to add and deduct a number of punches as well as getting the balance of an address.

**Machine** (appendix A.2 line 185) is the artificial representation of the machine that the system runs. It simply keeps an account of when it should be unlocked (in terms of block number) and how many cups it has poured since it was last filled. It is set to fail every 10th time to simulate it running out of coffee, after which it is immediately refilled (the counter is reset). It also provides a *getter* that tells whether it is currently open.

**Manager** (appendix A.2 line 434) is the smart contract that manages the other smart contracts. All the other smart contracts extend a super contract that is called `IsManaged` (appendix A.2 line 2) that provides a field for storing the address of the manager and a function for setting this. The manager stores the smart contracts in pairs of strings and addresses that makes it possible to store an arbitrary number of smart contracts in the manager. All the smart contracts then use the manager to contact each other by getting the right component using the `getComponent` function.

# 6 | Analysis of Implementation

In this chapter we will analyse our simple implementation by comparison with the old system in terms of trust, security, scalability and ease of use. We will at the end reflect on the proposed modular system compared to the simple system. The reason for analysing the simple implementation and not the modular is that the latter never ran on the testnet and therefore could not be tested.

## 6.1 Testing

In order to do a meaningful analysis of our implementation it must first be documented that it is in fact working as intended.

### 6.1.1 Solidity contract

To test the proof of concept smart contract implementation, you would normally do extensive unit testing on it to make sure that every method works correctly, especially in boundary cases. This poses a problem in the current state of Ethereum development, as there is not at present time[17] a unit testing library available for Ethereum development. There is one under development, but as it states on its README.md on GitHub it is not ready for use[18].

Due to this lack of an automated test environment, another path had to be taken. With the simple functionality of the punch card system, black box testing was deemed acceptable.

[17]13/05/2015
[18]`https://github.com/androlo/sUnit` (visited on 13/05/2015)

### 6.1.2 Black-box testing

As unit testing is not available, black box testing was chosen as an alternative. The results can be found in appendix B on page 101. The tests are designed to test boundary cases for all the functions used by the client, except for the `setIssuer` function because that would mean to hand over the client to someone else, and the functionality is equal to that of `setMachine`. `commitSuicide` and `emptyMachine` are not tested either, as they are trivial in their functionality.

As it can be seen from the test results, all tests but one passed. The one test that failed is in relation to the price of a punch card. The test is designed to test that negative numbers cannot be pushed, as it would result in a negative price. However, the function has been implemented using unsigned integers that does not allow for negative numbers, instead converting it to a very high number instead. Thus, the test fails. It is however chosen that this need not be fixed. Firstly, it is an administrative function, why only people with knowledge of the system will access it. Secondly, the result of the action is very clear, and can at worst only mean that customers cannot buy a punch card, not that they can get them for free. When this is upheld against the costs of issuing an updated contract, leaving it be is the better solution.

### 6.1.3 JavaScript client

The testing of the client would normally entail some form of usability tests as well as tests of the functionality. There are a few different functionalities present in the client, however they all link up directly to functionality in the contract, and due to the pending window in AlethZero it is possible to check that the correct transactions are sent at each click. The remainder of the functionality links up to showing information on the screen, which is easily tested by manual checks in AlethZero as both the original issuer and a customer and a check of the contract in a non-Ethereum client (to check that it states the correct information).

Usability testing was in this situation deemed a waste of time. The reasoning behind this is that the client is not meant for production in its current state, it is, as the name tells, only a proof of concept.

## 6.2 Trust and Security

As the implementation of the smart contract dictates the rules of the transaction processes, it removes the issue of trust from these. You are only able to buy a punch card if you have sufficient funds. It is able to identify if a user has the right amount of clips for a cup of coffee. It will deduct the right amount of clips and update a users balance automatically.

Consequently, the users do not need to know the transaction rules as they are instead dictated by the smart contract. This way malicious behaviour and misunderstandings are ruled out of the system since they are carried out automatically according to the defined protocol. Additionally due to the transparency of the operations conducted on the smart contract, users will also be able to verify that the actions are in fact carried out correctly, and that there are no invisible bias or malicious computing happening during transactions.

Therefore the system operates trust-free as we circumvent the problem of trusting the users to carry out the transaction correctly.

## 6.3 Limitations

There are some limitations and drawbacks to this implementation that will be analysed in the following.

### 6.3.1 Block time

One of the downsides of our implementation is that transactions can take up to approximately 12 seconds, due to the block time of the block chain. When purchasing a punch card or a cup of coffee the transaction has to go through and be submitted to the block chain before any balances are updated. This means waiting for a new block to be discovered. In general this would average out to around 6 seconds of waiting for your transaction to go through.

The block time issue is hard to circumvent due to the nature of the Ethereum block chain. This would mean to introduce a centralised service of some sorts in conjunction with the block chain in order to bring down transaction time. Using the block chain would then seem pointless since you are no longer having a purely decentralised system and thus compromising transparency, trust and security.

### 6.3.2 Extendability

Our system relies on two smart contracts where one of them has all the transaction logic. If we are to change inner workings of this specific smart contract we have to issue the entire thing again which is costly in gas, and all current punch cards would be lost, meaning that ether would have to be manually transferred back to the customers, which is costly too. Also the client needs to be updated with a new smart contract address for connectivity.

To address this problem getters and *setters* have been implemented in order to give the issuer the opportunity to change a lot of the data stored in the smart contract without having to issue an entirely new one. You are for example capable of changing the address of the coffee machine smart contract and the address of the issuer (essentially the administrator of the contract). The only reason to issue a new smart contract is if the logic of some of the transaction functions needs to be updated. These functions in its current form are however simple so it would be fair to assume that such updates would happen rarely.

If the system is to be extended you would have to either make another smart contract that manipulated the old one, essentially a module on top of the other, or reissue a new smart contract with the new logic.

### 6.3.3 Improvements

If we were to further develop the system, the nature of smart contracts would make it easy to extend the system with new features automating other processes of the coffee shop.

In transaction systems in general, the use of a block chain can prove useful in regards to get statistics of sales and revenue as you will have a complete dataset of transactions in one place. Consequently additional smart contracts can be implemented in order to handle the purchase of supplies through interacting with the record of sales. In this way, by automating processes, through smart contracts in conjunction with each other, a shop can be turned into a DAO.

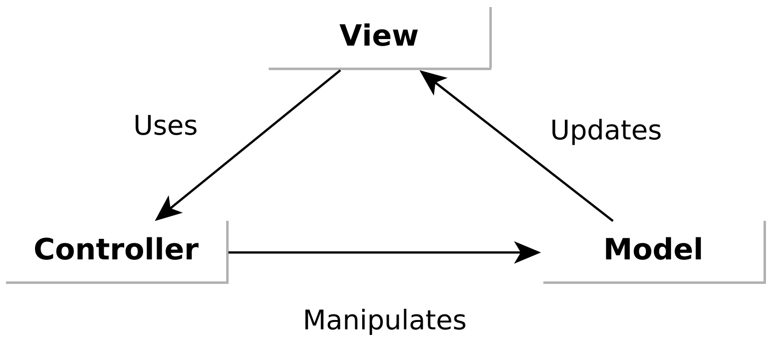
As stated in section 5.3.2 the system could be extended to handle pricing of the coffee automatically. A smart contract could set the price in ether in accordance to a fiat currency. In addition a type of exchange could also be implemented in order to accept different kind of cryptocurrencies or even fiat currencies. Such smart contracts might already be implemented in different systems on the block chain, but due to the public network you could utilise the same smart contracts in a different system. Due to this, by developing one transaction system in a clever way you have the generic building blocks of the smart contracts to easily build a transaction system that handles completely different entities. This is again one of the strong suits of smart contracts on the block chain.

The scope of this development is however only to implement the transaction process of the punch card.

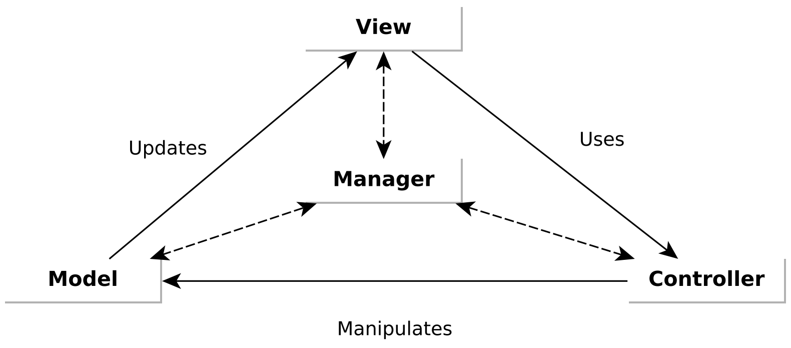
## 6.4 Modular Implementation

In order to make the smart contract less costly to update, it could be split up into more smart contracts as modules of logic as shown in the modular implementation detailed in section 5.4.

This implementation is related to the *Model-View-Controller (MVC)*[1] pattern with some alterations to make it fit better on the block chain. The alteration is that another component has been added to the pattern that keeps track of the different modules. This is needed, since the nature of the system does not allow for a restart and creation of new objects with the new components. Consequently the components have to be exchanged on



(a) Classic MVC



(b) MVC with a manager

Figure 6.1: Representations of the classic MVC (a) and the modified MVC (b). In the modified version all addresses of other components are found using the Manager, represented by dashed arrows.

a running environment or the system has to be discarded. In figure 6.1, a representation of the differences between the classic MVC and the MVC with a manager can be seen.

If this pattern is applied on the proposed modular implementation, it is obvious that the `Manager` is the manager in this case. The `Issuer` is then the view of the model, while the `AssetHandler` is the controller for the `PunchDb` which are one of the models. The other two models in the system are the `NameDb` and the `Settings`. All three models simply store data without really "thinking" about it[19]. The `Machine`, which is an artificial representation of

[19]The `PunchDb` does however check that it cannot go into negative features – it can be

a physical artefact stands out of this model in its current form.

The main reason for using this system over the simple implementation is extendability as mentioned in section 6.3.2. With the modules being interchangeable, it is possible to extend the system without it affecting other modules.

As an example, imagine that it is needed that the birthday of customers are shown and stored. In this case only the issuer needs to be updated, and a module for storing birthdays added. The fact that this can be done not only saves the issuer money, as the whole system does not need to be replaced, it also makes sure that all the customers still have their punch cards up to date without the need to transfer anything, as needed in the non-modular solution (see subsection 6.3.2).

The extra checks for message senders (needed for security reasons) and getting the addresses of other smart contracts do increase costs of running the modular implementation, as compared to the simpler solution. It is thus a matter of how likely a future extension of the system is, whether the one or the other solution should be chosen. This choice is harder to make for a decentralised system, as computing power is a scarce resource. In the case of the coffee shop at ITU, the simple implementation is most likely adequate, because of the limited user base and the low risk of needed extension later on.

argued that this belongs in the controller, however to limit the amount of unnecessary intercontract calls, this has been moved as it makes no sense to allow negative accounts in the system

# 7 | Discussion

In this chapter we will discuss our implementation and on basis of this some of the features of Ethereum and decentralisation through block chain technology in general.

## 7.1 Why Decentralise?

There are many other ways to implement a digital punch card, than using block chain technology. Digital assets are commonly known – so why not just implement a system where your punch card balance is maintained by a database running on a web server?

When implementing a system like this you will introduce a different aspect of trust, in the user having to trust that the system works correctly. Thus the system will not be verified by anyone but the issuer. A system like this is completely opaque and the user will not be able to know the inner workings of the transactions. The user has no way of verifying that the transaction is run correctly.

The fact that this solution is centralised introduces other problems as well. When the system runs on a centralised unit, it can be tampered with by someone hacking the system. Some level of security is needed. Also you introduce reliability on hardware. One server crash can have catastrophic consequences if proper backups are not implemented. And no matter what, a crash will mean downtime and lost profit.

These problems are inherently solved by the block chain technology in Ethereum. The lack of a central web server means that there is no server that can crash and bring downtime. Even if a node crashes, the system will still be up. As you cannot tamper with the data stored on the block chain, the system is much more resilient to hackers. Since the system is run transparently, you need not trust the issuer either.

Therefore block chain technology can be effectively used in transaction systems due its perks in the areas of security, transparency and trust – aspects that are all of utmost importance when conducting a transaction.

## 7.2 The Greater Picture

It is easy to make the specific implementation more generic. It is basically a punch card, which in turn is basically a parallel currency. Relatively easily, our smart contract can be rolled out in many different instances, each having its own GUI and offering a secure way to utilise the beneficial factors of the punch card method.

The fact that smart contracts can call other smart contracts makes series of smart contracts possible. In our example a punch will call a smart contract releasing a coffee machine to brew a cup of coffee. This could be extended to a series of contracts, performing different actions, to provide additional features. Due to this you can even use the smart contract as a single ticket system, while still implementing security for the handling of these tickets at the time they are used.

Perhaps you would not even need to specify all the smart contracts yourself due to all smart contracts being public and able to interact with each other. You could utilise a smart contract someone else has built in conjunction with your own system. For instance, our modular implementation could make use of the Ethereum *namereg* smart contract[20]. This is one of the powers of Ethereum: a smart contract can stand alone, but the same smart contract can be part of a much bigger system if it is implemented cleverly.

[20] `https://github.com/ethereum/dapp-bin/tree/master/namereg` (visited on 16/05/2015)

## 7.3 Economic Systems

As mentioned, our proof of concept system can be viewed as a parallel currency – one that in our example can be used to buy coffee, but as argued for could be used for basically anything. In our example, you use ether to buy the currency, but if you want to, you could set up a system that exchanges physical currency to ether before sending it to our system. Because of this our system could be the cornerstone in many situations where some sort of physical artefact can be exchanged for a digital one, and where security and reliability is important factors.

By having the entire block chain in the back of the smart contract, it becomes impossible to forge fake punch cards without having 51% of the total computing power of the network[11]. Furthermore it becomes impossible to (maybe illegally[38]) resell your clips for profit. Both factors are very appealing to the issuer. They can both secure that the issuer gets the money it is entitled to, and that no one will be able to buy large quantities with the sole objective of selling them off for profit, thus possibly buyout the issuer. The customers on the other hand know that they get what they pay for. There is no way for the issuer to artificially alter the balances of the customers without a publicly exposed method that does so.

## 7.4 Challenging the Block Chain

Naturally, everything is not just well and good in block chain country. As with all new technologies, there are some issues that have not yet been solved and an implementation phase is often more complicated than desired when a lot of parties, radical changes, and trust are involved.

### 7.4.1 The size factor

One of the first problems that springs to mind is that of scalability. Vitalik Buterin is very aware of this, and has been addressing the problem early on[15–17]. As he writes, the problem is that for the whole theory of the security of the block chain to apply, a large number of full nodes are required. Otherwise you might end up with a less decentralised system, like Bitcoin has experienced[5, 40]. The problem here is that this nullifies the security measure that decentralisation is a big part of (see section 4.3). It is unlikely that anyone with the computing power to carry such a big node would care about a small coffee shop such as the one at ITU, but if the technology is to be applied in a wider selection of application domains, as discussed in chapter 8, it is a very important aspect to discuss.

From the start, Ethereum has taken steps toward managing scalability better than Bitcoin. For instance, by using accounts rather than being based on *unspent transaction outputs (UTXO)*, which according to Buterin causes a significant amount of space being used, when a lot of transactions are being performed from one address.

Buterin has already presented several ideas on how to deal with the scalability problem[13, 15–17], though common for them all is that they slightly lower security and heighten complexity a great deal. Part of his concern is usability due to transaction fees, meaning how small values can feasibly be transferred on the Ethereum network. The major concern, however, is that of the sheer data size. As mentioned, the larger the block chain gets, the less secure it becomes as only a few nodes will be able to contain all the data. Buterin even mentions that in a use case like Ethereum, where the uses for the block chain are so many, it might be conceivable that the block chain will rise to a size where no single data center can contain all of it[16]. Therefore it seems like partitioning the block chain into smaller digestible pieces may be the only way to go forth.

The problem is that when you subdivide the block chain into smaller *substates*, you also divide the amount of hashing power needed to push illegitimate blocks, as miners only mine blocks within their own substate. Therefore, if there are 200 substates, one will only need 4% of the total hashing power to take over a substate and potentially invalidate the entire system[16]. Several possible solutions to this are covered, but all of them need to be proven effective before they can be implemented safely. Buterin goes on to explain a model he calls “The Twelve Dimensional Hypercube”.

In this model, the block chain is also subdivided, but in a different way, so that there are edges between the states that messages can move along. This solution also includes a move from the current consensus scheme of proof of work to proof of stake (see section 4.3.2). This should make sure that only if one controls more than 33% of the total sum of ether in the system, they can make illegitimate transactions.

The final solution presented to the scalability problem is to have many block chains[17]: some for one specific purpose, some more generalised purposes, like Ethereum. The idea behind this is that the block chains can use each other to provide security for one another, no matter what the separate block chains are used for. This way, a miner can mine on a block chain of a suitable size, and yet security would still be very high, albeit not as high as if only one block chain existed. This solution also relies on a proof of stake model, like the previous one, but because of the interconnectivity of the different chains, a high percentage of the combined stakes of the block chain ecosystem would be needed.

These solutions are all more technical than the scope of this paper and only visions at the time of writing[21]. Therefore we will not go into more detail about the technical aspects of them.

### 7.4.2 Coping with stress

Another obstacle that is important in the more widespread use of block chain technology, but not so much in the specific example presented in chapter 5, is how the system copes with stress. A payment technology such as VISA handles 4,000 transaction per second in average and has been stress tested in 2013 to handle 47,000 transactions per second[50]. Bitcoin can in comparison only handle 7 transactions per second, due to the fact that block sizes are restricted to have a maximum size of 1 MB[4]. This is not so much a problem if the block chain is only used to buy coffee in a small coffee shop. However, if Ethereum is to be utilised on a broad spectrum of applications, more are needed. If for instance the customers in the coffee shop cannot buy coffee

[21] 17/05/2015

because an election is being held in Nigeria, it is going to be hard to sell the idea of the system to the coffee shop.

It is obvious that this is not a viable amount of transactions, should this technology be implemented on a global basis. Changes to the way different companies use the block chain are, however, being made, in order to assure that a transactions per second level alike or even higher than VISA's can be handled. Some of the theories in section 7.4.1 will also help Ethereum being able to handle such levels of transactions per second, thus eliminating one of the huge advantages that established technologies like VISA have over decentralised block chain technologies[16].

### 7.4.3 Waiting disrupts the flow

Another disadvantage of running applications on the block chain is, as mentioned in section 6.3.1, the time factor. Ethereum has a block time of 12 seconds, meaning this is the potential waiting time for an action to be carried out. This is very unlike the norm of today, where processing and internet speeds are at a level, where it is expected that any request is processed almost immediately.

This will take some getting used to by regular users, as convenience is very important in a world where consumers are used to one-click shopping[22]. Therefore it might be an obstacle in regards to the universal acceptance of the technology as an alternative to centralised services. For instance, in our proof of concept, you will need to wait up to 12 seconds from "punching" your punch card till you can get coffee, where as today it is an instant process.

And this is not even the whole disadvantage. If it was an equal waiting time, for instance 12 seconds for everyone, then the flow of purchase would continue, just with the added wait. However as it is, all transactions made within a timespan of 12 seconds will be carried out simultaneously. This means that if the customer power peaks at around 8 customers in 12 seconds[23], 8 people will get the coffee machine opened at the same time,

[22] `http://www.amazon.com/gp/help/customer/display.html?nodeId=468482` (visited on 08/05/2015)

[23] This number is purely fictional, and based on: if there is a queue; how many customers

which can become a bottleneck. This added disruptiveness in the flow of process will be present in all application domains where block chain technology is implemented.

There are use cases for block chain technology, where the block time wait is of little matter. For instance, if the technology was to be used as a permanent record of property rights, a twelve second wait for the transfer of property such as land or a house is nothing. This will be covered in more detail in section 8.2.

Ideally you would want to lower the block time even further. However it has to be taken into account that the internet itself is not capable of instant transfer of information. Whenever a miner successfully mines a block, it does take some time before everybody on the network are informed of this. This leads, eventually, to lowered security, as Buterin describes in a blog post from July last year[18]. As this shows, there is a strict balance between the security of the protocol and *block propagation*, which needs to be addressed when lowering the block time. Until proper methods have been developed and tested, going lower than 12 seconds is too risky. The improvements already implemented are significantly better than the 10 minutes of Bitcoin, almost without lowering security. Sure, 12 seconds will reduce realistic use cases, but it is a good place to start. When improvements are introduced, more use cases will come.

### 7.4.4 Fees: hidden or blatant

Costs is one of the more obvious drawbacks of decentralisation and block chain technology. In the centralised world of today, someone pays for a server to run their application, or the user downloads a program that then runs on their computer. Most likely, the cost of running a server will somehow be paid by consumers since companies need to have profit, and the costs of running a server is an expense that needs to be covered. However this cost

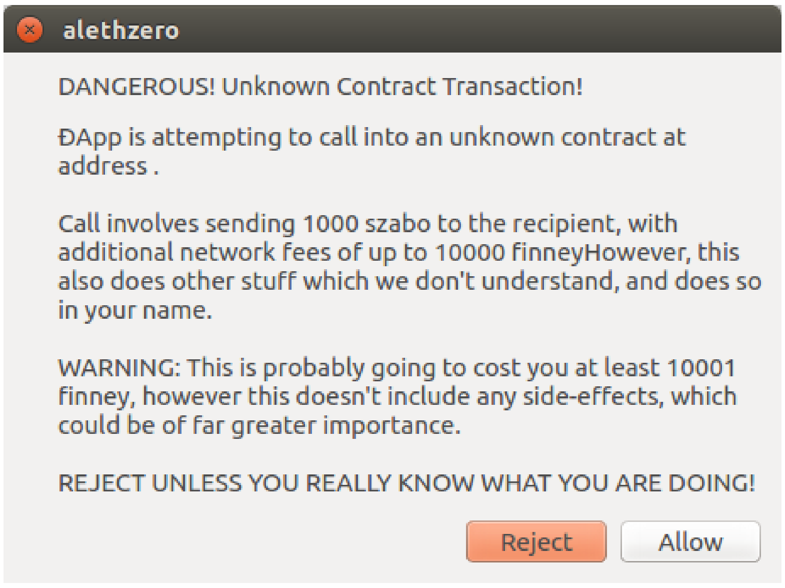
could feasibly deduct a clip from their current physical punch card.

is often hidden from the user. On the block chain it is very blatant that transactions and computational power come at a price.

This is definitely going to be one of the challenges of Web 3.0, as users are not used to pay in this way for services online, and that fact might hinder the spread of the technology. The fact that users will be constantly reminded that an action has a fee, will certainly make some users choose centralised solutions where the prices are more hidden. For instance, when looking at the proof of concept presented earlier in this paper (see chapter 5), there will be fees involved when buying a punch card, and every time it is punched. These fees do not exist in the current transaction process. Another fitting example is within the realm of tickets. It is easy to see the same logic that is present in the punch card smart contract utilised for the sale of tickets e.g. sports events (as mentioned in section 7.2). Here the customer would be faced with very blatant fees upon purchase and utilisation of the ticket. In a centralised system these fees would most likely be included in a “handling fee” and possibly in the printing of the ticket, both not as visible as a dialogue popping up on a screen.

Furthermore centralised services have an advantage: They can control fees as they please. If it comes to outright competition with, for instance, an Ethereum-based solution, the centralised actor can just remove the fees, and take the slight reduction in profit. The decentralised will have to reduce the price instead, but the blatant fees will still be present. Therefore a decentralised solution will have to be very comprehensive in describing the total price, including what fees there are and why.

Otherwise it is conceivable that users will see the first message about fees and consequently use a centralised service instead. Because of the way ÐApps are accessed (through an Ethereum client), there will most likely be added a very clear message about the possible fees included in the action, possibly like it is shown in the AlethZero client in figure 7.1.


alethzero

DANGEROUS! Unknown Contract Transaction!

ĐApp is attempting to call into an unknown contract at address .

Call involves sending 1000 szabo to the recipient, with additional network fees of up to 10000 finneyHowever, this also does other stuff which we don't understand, and does so in your name.

WARNING: This is probably going to cost you at least 10001 finney, however this doesn't include any side-effects, which could be of far greater importance.

REJECT UNLESS YOU REALLY KNOW WHAT YOU ARE DOING!

Figure 7.1: The transaction dialogue of AlethZero as of 12/05/2015 when a user tries to buy a punch card in the proof of concept presented earlier.

### 7.4.5 Fluctuating currencies

There are other things making it difficult to adapt a block chain technology for common use. One of them is the ever changing value of the different cryptocurrencies. Over the last two years the value of 1 BTC has decreased from approximately \$1150 to \$200[24]. It is common that the value of 1 BTC varies with more than \$2 a day[25], which makes it an extremely unstable currency. This means that people potentially can lose a big amount of money in a very short period of time. When applying the technology in concrete situations like presented in chapter 5 this is a problem. It would require the price to be updated several times a day, or to implement a contract that would do this automatically. This does, however, not remove the problem completely. Even if the price in ether is kept up to date constantly, it would still require that the coffee shop transferred this into fiat currency continuously, otherwise they could potentially lose a significant amount of money. In the same way, it would be required that the customer bought the ether

[24]`https://goo.gl/sbi47u` (visited 18/05/2015)
[25]`http://www.coindesk.com/price/` (visited on 16/05/2015)

just before spending it. These are significant drawbacks in any system using block chain technology.

Therefore some kind of assurance mechanism has to be developed in order to ensure that the value of the consumers' assets remain the same. In order to do this some derivation of *futures contracts* could be used. Some initiatives have already been made in regard to this[26]. As the name, futures contracts imply, it is a contract between parties. Functionality like this is very easy to implement in a smart contract using Ethereum.

### 7.4.6 Keeping everything safe

In the current punch card system at the coffee shop at ITU there is one way to lose all your "currency": to actually lose your punch card. This is similar in the proof-of-concept implementation where losing the private key to your address will cause the same situation, albeit in a much more serious way. Such a loss will make your account unavailable for good – there is no way of retaining it, as there is no central authority that keeps track of this.

In comparison, Denmark has a central authentication system today, called NemID[27]. Here, your private key is stored on a central server, thus not trust-free nor decentralised. A similar solution for your block chain address can easily be developed, but with the same drawbacks: introducing trust.

### 7.4.7 Converting currencies

As mentioned in section 7.4.5, it is necessary to convert your fiat currency to cryptocurrency for you to use a block chain based system. This can be a hindrance, as it is another step in the purchase process that is not present in the current system. If given the choice of either having to convert fiat currency to ether to buy a punch card or put down a coin at the counter, the latter is far the easier. Therefore, it will possibly be necessary for the coffee shop to work with a currency exchange of some sort, when dealing with the purchase process[3].

[26] `http://opture.co` (visited 03/03/2015)
[27] `http://en.wikipedia.org/wiki/NemID` (visited on 16/05/2015)

Digital Currency Exchangers work in the same way as stock exchanges or most other exchanges. It allows for people to sell and/or buy a given cryptocurrency for conventional fiat currency or other cryptocurrencies[28]. As the list in the Wikipedia article shows, very few of the most common used exchanges are decentralised. This means that if hackers are able to gain access to the centralised servers that run the exchange, situations like the MtGox hacking might occur[37]. As the article states, the security was very bad in this particular case. MtGox was considered a monopolist in the bitcoin exchange market at the time being. The aftermath of them losing $460 million worth of BTC, has however resulted in a vast amount of new rising exchanges with better security[3].

The fact that people have actually lost huge amounts of money due to exchanges being hacked, combined with the problems discussed above in regards to securing your private key, might be one of the main reasons why the layman will find it difficult to make the switch to using some kind of cryptocurrency and block chain technology to handle day to day financial matters.

If the coffee shop, or any other shop with a system set up on block chain technology, does not want to rely on an exchange, there are alternatives. The Ripple technology presented in section 8.3 also allows for seamless currency conversions between fiat currency and cryptocurrency. This technology, however, has its own drawbacks that will be discussed later on.

[28]`http://en.wikipedia.org/wiki/Digital_currency_exchanger` (visited 15/05/2015)

# 8 | Reflections

In this chapter we will reflect upon our discussion and implementation and discuss how the more generic block chain of Ethereum can be used in the domains of digitalised voting; land rights; and currency conversion and transfer in a broader perspective than cryptocurrencies. Finally, we will discuss what this means for the banking sector as it is today.

## 8.1 Electronic Voting

There is one area where the implementation of computer technology has yet to get widespread approval: elections. Parliamentary and presidential elections are of such importance that the security needs are much greater than in most other applications and have therefore haltered the implementation of computer technology[30].

There are a lot of unsafe areas of electronic voting that have to be dealt with before it can be implemented, and the fact that hackers continue to break into government databases, like the CSC hacking of 2012[49], is not helping convince the public.

Problematic areas within the process of voting exist when it is done on an electronic network like the internet. Malicious software can be installed on the computer of the voter to alter their vote before it gets recorded.

There is also the fact that the votes have to be stored. If they are stored on a centralised server, it would be a playground for hackers. Some might have political agendas, some economical and some might just find it challenging to break in and alter an election result[30]. If the votes are solely stored on the server, it will not be possible to check the validity of the result, and even if a successful attack is discovered, the only way to go would be a time consuming and costly re-election. There have been made measures to counter this in the form of systems that print a physical ballot as a backup[7], but it is still not optimal.

These storage insecurities could largely be nullified if a block chain approach was used. All transactions are timestamped so it cannot be changed, except via a 51% attack[11]. It would be virtually impossible for a single man or a network of activists to manipulate such a block chain. But perhaps a state would have the resources to do so. Super powers like the United States have the resources to tamper block chains through sheer computing power. Thus, the block chain would not offer complete security, but it would be a much harder task to influence an election. Furthermore it would be quite evident that someone has been tampering with the network if it was done.

Now this only deals with the storage of the votes. There are still the unsolved troubles of *man in the middle attacks* on the computer the voter uses. If we keep the domain of this discussion to parliamentary elections in Denmark, it would be a solution to still do voting the old fashioned way, where you get your voting card in the mail and go to the nearest voting place on election day. But instead of getting a long sheet of paper, you get something like a QR code that you scan in the voting box in order to cast your vote. The QR code could represent a sending address on the Ethereum block chain, and your vote would then be sent to a smart contract that validates the address and registers your vote. That transaction will then be on the block chain as a permanent record of the anonymous vote. However some precautions must be taken in order to ensure the security on the used computer.

The benefits of using a system like this are many – some more clear than others. Of course the counting of votes are easily automated, greatly reducing the work and time needed before a result is clear, while at the same time removing human errors. Furthermore it will simplify the process of voting. In addition this system also ensures the anonymity of the voters, like today, due to the voter being given a random address. This is better than for instance Estonia's maligned system[29]. Here voters can only vote by registering their ID card at the computer they are voting at, potentially losing their anonymity.

To sum up, if block chain technology is used, some of the significant risks of electronic elections can be dealt with, with relative ease. If a platform like Ethereum is used, it will not be too difficult to create a smart contract that handles the election process securely.

## 8.2 Digitalised Rights

One of the application areas where the block chain technology, in its decentralised immutable nature, really comes to its right, is within the realm of transferring property rights. Peruvian economist Hernando de Soto claimed back in 2000 that roughly $9 trillion is locked up in "dead capiltal" in impoverished countries due to a lack of a trustworthy way of keeping track of land rights[47]. This is an excellent example of the problem of trust in the financial sector that one might not immediately think of. And it is one, where the strength of the block chain really comes to show.

Block chain technology offers an elegant solution to this problem that has its roots in the lack of a trustworthy authority, due to high corruption[20]. When no single entity can falsify a transaction, and that transaction represents the handling of ownership of a piece of land, it is not possible for anyone to steal land by bribing officials.

It is easy to envision the overall structure of how such a system would work. On Ethereum, an implementation would look somewhat like the proof of concept presented in chapter 5. Of course, a deed to a piece of land is much more complex than a punch card for coffee, and further documentation might need to be stored, for instance on *Swarm*, the decentralised storage unit of Ethereum that is yet to be developed[30]. A hash of the documents can be stored in the transaction to verify that the documents remain unchanged.

Another way of using block chain technology to tackle the same problem

[29]`https://estoniaevoting.org/` (visited on 16/05/2015)
[30]`https://github.com/ethereum/cpp-ethereum/wiki/Swarm` (visited on 16/05/2015)

is presented by Factom.org[31]. They utilise the Bitcoin block chain to store a hash of the documentation and keep the actual files on what they call the "Factom Data Layer"[32]. This solution is very similar to what could be provided on Ethereum in the future, except it is more specifically targeted this kind of transactions. This has both benefits and drawbacks. The more generalised approach of Ethereum will most likely make sure that there are more miners on all the data. Factom on the other hand will have a strong block chain in Bitcoin for a small part of the data, whilst a more specialised entity is responsible for the storage of the rest. Some of the features of Ethereum are superfluous on a system like this. For instance the 12 second block time, as mentioned in section 7.4.3, has little positive effects in a system like this. There is also a difference in how the data is audited. Factom relies on client side auditing, while Ethereum allows for auditing on the block chain for the price of gas fees. It is not meant for this paper to take a standpoint on which is best, but it goes to show that the same general technology, can be utilised in different ways to solve the same problem.

## 8.3 Ripple

Ripple is not a block chain technology, but it still belongs in this paper as a reference point and to serve as a possible use case for block chain technology.

### 8.3.1 What is Ripple?

Above, it is stated that Ripple is not a block chain technology, and that is true, when we define that as a technology that via cryptographic hashing ensures security and trust-freeness in a network built up by many decentralised nodes that all verify all transactions. Ripple uses a different strategy that they call a consensus algorithm, which tries to reach a consensus between nodes that are mostly Ripple's own servers. The software has since 2013 been open source[14], but the clients are still using Ripple's servers as per

[31] `http://factom.org` (visited on 12/05/2015)

[32] Showed in this video `https://www.youtube.com/watch?v=uYQ5icxGvmA` (visited on 12/05/2015)

default for verification. In this way, Ripple is a centralised service that therefore is not vulnerable to 51% attacks. However, the users put their trust in Ripple Labs, who has in the past changed features of the network without prior notification[14]. Up until the release of the source code, the network was 100% reliant on Ripple Lab's continued existence and dedication to the project, though that has changed in the current environment, as the project could theoretically be continued by the community.

The consensus algorithm is used to gain mathematical certainty for every transaction (that 80% of the nodes verify it), and any transaction that are not verified this way are discarded. Those that are verified are made into a "last closed ledger", which is then stored, sort of like the blocks on a block chain. The very functionality of this consensus algorithm has been a concern of some parties within the cryptocurrency world. Especially since a ledger fork happened in the Stellar Network, which uses the same consensus protocol as Ripple. This fork, which eventually resulted in loss of transactions, led to concerns over the consensus protocol to a degree where the Stellar Network went over to a centralised approach until a new white paper could be written and implemented[34].

Ripple is meant as a way to transfer assets from one person to another based on trust. While the network does have a built in cryptocurrency, this is more meant as a way to do trust-free transactions where trust cannot be obtained, and as a security measure, than an actual currency. The system has a path finding algorithm that tries to find a path of peers connected by trust between a sender and a recipient of a transaction. This way the system manages trust instead of making it obsolete. This is of course in it self a drawback, however, it does allow for the transfer of assets other than cryptocurrencies, and for the seamless transfer of different currencies, including cryptographic variants. Finally it needs to be mentioned that Ripple relies on gateways to transfer money in and out of Ripple, and these are too entities that the users need to trust[14, 27].

### 8.3.2 Ripple and block chain technology

Since the consensus algorithm of Ripple has been under scrutiny, it is feasible to look into a similar implementation being made on block chain technology. Some of Ripple's selling points are that it consumes less electricity to run than block chain technology does, and that it is not vulnerable to 51% attacks[14]. Both of these are true in some manner, though it can be argued that with the release of the source code for the verification nodes, someone could gain a majority and cause disruptions and forks like described earlier. If a similar system was introduced in a block chain environment this security flaw would not be present, while that of the 51% attack is something that is under continuous research. The features of the Ripple solution, such as the path finding algorithm, would have to be built into the protocol which would undoubtedly be a task for a research team, but as seen with the generalised approach of Ethereum, this should be possible.

## 8.4 The Opaque Banking Sector

So far in this paper, it has been shown how block chain technology can handle payments, accounts and even currency conversion securely and automatically. This leaves a problem for a large group of businesses: banks. If banks are not to be made obsolete by the progression of technologies such as Ethereum, Bitcoin and Ripple, they have to adapt.

In the finance world as we know it today, almost all relations are built on trust. A trust that, in the light of events throughout the last ten years, starting with the bankruptcy of Lehman Brothers, has only decreased among consumers of the finance and banking world[2, 48].

According to a PwC report only 32% out of 2000 people across the UK trust retail banks and only 15% trust investment banks[45]. These numbers are alarmingly low and call for some severe action to be taken in order to restore some acceptable level of trust among consumers and their banks. It is only 10% of the asked consumers from the report whose trust in retail banks have increased. For investment banks this number is even lower at only 6%.

Banks are naturally working on regaining as much of their consumers' trust as possible, as solutions are being suggested and tested throughout the global financial markets[22]. Denning points out that the number one problem is the lack of transparency:

> "*The root cause of the meltdown was opaqueness. After Lehman's collapse, no one could understand any particular bank's risks from derivative trading and so no bank would deal with any other bank or shadow-bank.*"[22]

This ultimately lead to the decrease of the consumers' trust in banks. Edelman's Global Trust Survey of 2014 shows that banks are the least trusted of all industries at only 51% and technology the most trusted industry at 79%[24]. Once again the numbers are not in favour of the banks, and since trust has always been the most paramount factor in banking, it seems something is being done in a very wrong way. Denning continues his argument with:

> "*Because most of the big banks and many of the shadow banks had been involved to an unknown degree in risky derivative trading, no one could tell whether any particular financial institution might suddenly implode.*"[22]

This enhances the fact that the lack of transparency is a large problem. If not even banks are able to trust other banks, it is very difficult to justify how consumers should be able to do so.

Denning has a series of suggestions on how to re-establish the trust. These suggestions include changing the focus of the banking sector to continuously add value to their costumers; making money as the result instead of the goal of activities; being totally transparent about all its activities because it is proud of how it conducts its business and more[22].

These suggestions all seem like viable solutions, but let us consider an entirely different approach to solving these problems. Instead of basing banking on trust, why not remove the factor that is the biggest problem in banking: trust.

This can be done with a decentralised, trust-free, consensus based technology. A technology like this, is exactly what a block chain is, as described in section 4.3.

Due to the way it works and the completely transparent ledger that comes with it, block chain technology makes it possible to trace money and transactions across big banks and *shadow banks*, avoiding unknown degrees of risky derivative trading.

As we show in our proof of concept in chapter 5, it is far from difficult to take a trust-based financial concept and apply some kind of block chain technology to make it into a trust-free concept.

Further more, according to an article in The Economist, Head of Innovation at VISA, Jim McCarthy, has said that if a network like VISA were to be build today it would almost certainly be decentralised[23].

The above points in favour of the banks at least considering if they are doing things the right way and to take a serious look at how they can use block chain technology to strengthen their business and regain the trust of their costumers. It is also a sign that the financial sector is beginning to open its eyes and acknowledge some of the possibilities that block chain technology offers.

### 8.4.1 Adaptation

In 2003, Bradley and Stewart, wrote about diffusion of online banking[8]. At that time, online banking was a very new thing. When they wrote the paper only few papers had investigated the diffusion of the online retail banking sector. Bradley and Stewart state that "[at] a fundamental level, the industry adoption level is governed by supply and demand side factors – that is, consumer demand for online banking and the available supply of new technology, particularly where it can reduce reliance of the branch network"[8].

Fast forward a little more than a decade, and the situation is arguably the same. A new technology has arisen, giving the banks the opportunity of getting rid of one of the huge issues that the banking sector is subject to, as described in 8.4: trust. Banks are researching the block chain technology and Fidor Bank has already adopted the Ripple (see 8.3) technology[36, 46]. This is indeed a step towards a decentralised banking sector or at the very least a transparent banking sector. According to Roger's bell curve from the Technology Adoption Lifecycle model[33], block chain technologies might be considered to be in the "early adopters" phase.

In 2003, online banking was reducing reliance of the branch network[8]. Today, the block chain makes it possible to remove reliance on any kind of centralised opaque trust-based network.

Large corporations have, to some extend, started accepting bitcoins as payment for their services[21]. It should be understood that they are not actually accepting bitcoins themselves, but have deals with bitcoin exchanges that instantly converts the bitcoins to a fiat currency and transfers them on to their bank accounts. It cannot be ignored that Bitcoin is the first cryptocurrency being globally accepted. In the light of technologies like Ethereum, it can be argued that Bitcoin might not be the last.

The Swiss investment bank, UBS, has, as one of the first huge banks, taken a decision towards adapting block chain technology and are opening a block chain research lab in London[42].

If history repeats itself the same way it did with the adoption of online banking, chances are that we, in the next couple of years, will see big changes in the way banks conduct their business.

[33]`http://en.wikipedia.org/wiki/Technology_adoption_lifecycle` (visited on 15/05/2015)

# 9 | Conclusion

The purpose of this paper has been to explore drawbacks and benefits of using block chain technology in application areas where transparency and trust-freeness advantageously can be used.

A working proof of concept of a digitalised punch card transaction system was designed and implemented successfully on the Ethereum block chain in order to evaluate this technology. This was done through extensive research and accounting for the technology.

The analysis of the implementation proved how the features of Ethereum and block chains in general can be utilised beneficially in economic systems.

The currently trust-based solution of the self-service system was automatised through the use of smart contracts and consequently turned into a trust-free system. Accordingly users can no longer misunderstand or misuse the transaction system as the transaction rules are dictated by the block chain. In addition the transparency of the block chain enables users to verify that transactions are carried through as predicted. Furthermore due to having the consensus checking of the entire block chain network behind it, the system cannot be hacked or otherwise tampered with, making the system very secure.

These are all features that are desirable in trust-based systems, why we argue that block chain technology can be beneficially utilised in such.

The analysis also uncovered some limiting factors of the implementation that were argued to be issues of the block chain technology in general. The issue of block time especially is a limitation of our punch card system.

Issues of scalability, cost and block time were thus heavily discussed and it was found that these aspects of block chain technology in general are not quite on par with the solutions offered in centralised systems.

Fluctuation, transaction frequency and security were also discussed, but these factors are all areas where suggestions to viable solutions were found.

Right now there is no good solution to the issue of the block chain becoming too big for ordinary users to carry a node on the network. There are suggestions and theories on this subject, but they remain untested. Therefore at this point in time, the bigger a block chain gets, the more centralised the network becomes.

Similarly a lower block time is desirable, however it would make the network unstable. Consequently the responsiveness of the applications suffer.

The issue of every action on the network essentially costing money, is also an obstacle to overcome in order for users to accept this technology. We argue that it will require a certain adoption time as users are used to fees being more hidden.

These are all issues that need to be addressed in order for block chains to become widespread in applications. Therefore innovation and research are still necessary in this field.

Even though these hurdles exist it can be argued that block chain technology potentially can have significant impact on society in general.

We argue that block chain technology is one of the most promising technologies so far if you were to implement digital elections. There are however aspects of security that are not solvable through block chains in this use case that needs to be taken into account.

In terms of solving the problems of securing digital rights we argue that the features of the block chain are very beneficial. The problem has its roots in the lack of trustworthy authorities, why creating a trust-free system would rectify this. In this specific use case it is hard to find obvious drawbacks to using the block chain, as factors like block time and costs are of little relevance here, why we recommend block chain technology to be utilised.

In the comparison of Ripple to block chain technology there are some drawbacks of the method Ripple uses that has been publicly exposed. These include centralisation and severe problems in consensus, both problems that are inherently fixed with block chain technology. Ripple, however, provide a solution to one of the earlier mentioned problems of block chain technology: the secure conversion of fiat currency to cryptocurrency.

Discussions on how block chains can be used in financial areas made it clear that the banking sector as we know it, has to seriously consider the way it conduct its business if it wants to regain the trust of its consumers. It is suggested that one possible way of doing so is by removing the element of trust entirely by using block chain technology to build transparent and trust-free systems.

If we were to continue researching this topic, it would be interesting to actually implement our system in collaboration with the coffee shop in order to get some actual feedback and results on the system. An analysis of the data could determine how a trust-based system may benefit or suffer from converting to a trust-free system

Furthermore it would be relevant to implement the suggested improvements of the system in order to develop a complete DAO. This would mean extending the system by automating more of the processes of the coffee shop through extensive use of smart contracts communicating with each other. It would be interesting to see which possible unforeseen problems this can bring in regards to handling the challenges of the block chain discussed in section 7.4.

In addition the implementation of a hardware solution on the actual coffee dispensers would be a good way to investigate an actual case study of the Internet of Things.

To conclude this thesis, block chain technology definitely has immense potential due to its revolutionising features. The technology is already applicable in a vast amount of application domains. If the continued development can overcome the challenges block chains face, we predict that they will have a fundamental impact on the financial sector as well as society.

# References

[1] Jeff Atwood. *Understanding Model-View-Controller.* 5th May 2008. URL: http://blog.codinghorror.com/understanding-model-view-controller/ (visited on 15/05/2015).

[2] Vikas Bajaj. *Financial Crisis Enters New Phase.* http://www.nytimes.com/2008/09/18/business/18markets.html?pagewanted=all. 17th Sept. 2008.

[3] Prableen Bajpai. *A Look At The Most Popular Bitcoin Exchanges.* 19th Nov. 2014. URL: http://www.investopedia.com/articles/investing/111914/look-most-popular-bitcoin-exchanges.asp (visited on 15/05/2015).

[4] Bitcoin. *Scalability.* 10th Jan. 2015. URL: https://en.bitcoin.it/wiki/Scalability (visited on 13/05/2015).

[5] *Bitcoin Hashrate Distribution.* URL: https://blockchain.info/pools (visited on 16/05/2015).

[6] *Block Chain Documentation.* URL: https://bitcoin.org/en/developer-guide#block-chain (visited on 04/05/2015).

[7] Danny Bradbury. *How Block Chain Technology Could Usher in Digital Democracy.* 16th June 2014. URL: http://www.coindesk.com/block-chain-technology-digital-democracy/ (visited on 12/05/2015).

[8] Laura Bradley and Kate Stewart. "The Diffusion of Online Banking". In: *Journal of Marketing Management* (2003).

[9] Paul Brody and Veena Pureswaran. *Device democracy: Saving the future of the Internet of Things*. 2015. URL: http://public.dhe.ibm.com/common/ssi/ecm/gb/en/gbe03620usen/GBE03620USEN.PDF (visited on 15/05/2015).

[10] Vitalik Buterin. *Ethereum Development Tutorial*. 2nd July 2014. URL: https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial (visited on 29/04/2015).

[11] Vitalik Buterin. *Ethereum White Paper*. 2013. URL: https://github.com/ethereum/wiki/wiki/White-Paper (visited on 16/05/2015).

[12] Vitalik Buterin. *Hard Problems in Cryptocurrency*. 21st Mar. 2014. URL: http://vitalik.ca/files/problems.pdf (visited on 18/05/2015).

[13] Vitalik Buterin. *Notes on Scalable Blockchain Protocols (verson 0.3)*. 14th Apr. 2015. URL: https://github.com/vbuterin/scalability_paper/blob/master/scalability.pdf (visited on 06/05/2015).

[14] Vitalik Buterin. *Ripple is Officially Open Source*. 26th Sept. 2013. URL: https://bitcoinmagazine.com/7275/ripple-is-officially-open-source/ (visited on 13/05/2015).

[15] Vitalik Buterin. *Scalability, Part 1: Building on Top*. 17th Sept. 2014. URL: https://blog.ethereum.org/2014/09/17/scalability-part-1-building-top/ (visited on 29/04/2015).

[16] Vitalik Buterin. *Scalability, Part 2: Hypercubes*. 21st Oct. 2014. URL: https://blog.ethereum.org/2014/10/21/scalability-part-2-hypercubes/ (visited on 29/04/2015).

[17] Vitalik Buterin. *Scalability, Part 3: On Metacoin History and Multichain*. 13th Nov. 2014. URL: https://blog.ethereum.org/2014/11/13/scalability-part-3-metacoin-history-multichain/ (visited on 29/04/2015).

[18] Vitalik Buterin. *Toward a 12-second Block Time*. 11th July 2014. URL: https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/ (visited on 08/05/2015).

[19] Vitalk Buterin. *Visions, Part 1: The Value of Blockchain Technology.* 13th Apr. 2015. URL: https://blog.ethereum.org/2015/04/13/visions-part-1-the-value-of-blockchain-technology/ (visited on 12/05/2015).

[20] *Corruption Perceptions Index 2014: Results.* URL: www.transparency.org/cpi2014/results (visited on 10/05/2015).

[21] Jacob Davidson. *No, Big Companies Aren't Really Accepting Bitcoin.* 9th Jan. 2015. URL: http://time.com/money/3658361/dell-microsoft-expedia-bitcoin/ (visited on 16/05/2015).

[22] Steve Denning. *How Can Bankers Recover Our Trust?* 6th Feb. 2015. URL: http://www.forbes.com/sites/stevedenning/2013/02/06/will-we-ever-trust-bankers-again/ (visited on 12/05/2015).

[23] The Economist. *Blockchain - The next big thing.* 9th May 2015. URL: http://www.economist.com/news/special-report/21650295-or-it-next-big-thing (visited on 09/05/2015).

[24] Edelman. *2014 Edelman Trust Barometer.* 1st Jan. 2015. URL: http://www.edelman.com/insights/intellectual-property/2014-edelman-trust-barometer/about-trust/global-results/ (visited on 12/05/2015).

[25] Ethereum Foundation. *Ether Unit Converter.* URL: http://ether.fund/tool/converter (visited on 06/05/2015).

[26] Ethereum Foundation. *JavaScript API.* 2014. URL: https://github.com/ethereum/wiki/wiki/JavaScript-API (visited on 12/05/2015).

[27] *Gateways.* URL: https://ripple.com/knowledge_categories/gateways-2/ (visited on 13/05/2015).

[28] Robert Grahan. *BitCoin is a public ledger.* 30th May 2013. URL: http://blog.erratasec.com/2013/05/bitcoin-is-public-ledger.html#.VUfQX-SlilM (visited on 04/05/2015).

[29] Glen Greenwald and Ewen McAskill. *NSA Prism program taps in to user data of Apple, Google and others.* 7th June 2013. URL: http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data (visited on 07/05/2015).

[30] Doug Gross. *Why can't Americans vote online?* 8th Nov. 2011. URL: http://edition.cnn.com/2011/11/08/tech/web/online-voting/ (visited on 12/05/2015).

[31] Vinay Gupta. *The Ethereum Launch Process.* 3rd Mar. 2015. URL: https://blog.ethereum.org/2015/03/03/ethereum-launch-process/ (visited on 18/05/2015).

[32] Alan R. Hevner, Salvatore T. March and Jinsoo Park. *Design Science in Information Systems Research.* 2004. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.1725&rep=rep1&type=pdf (visited on 13/05/2015).

[33] Stan Higgins. *IBM Reveals Proof of Concept for Blockchain-Powered Internet of Things.* 17th Jan. 2015. URL: http://public.dhe.ibm.com/common/ssi/ecm/gb/en/gbe03620usen/GBE03620USEN.PDF (visited on 15/05/2015).

[34] Stan Higgins. *Stellar Network Fork Prompts Concerns Over Ripple Consensus Protocol.* 9th Dec. 2014. URL: http://www.coindesk.com/stability-questions-dog-ripple-protocol-stellar-fork/ (visited on 13/05/2015).

[35] Laura Hood. *Google's terms and conditions are less readable than Beowulf.* 17th Oct. 2013. URL: http://theconversation.com/googles-terms-and-conditions-are-less-readable-than-beowulf-19215 (visited on 07/05/2015).

[36] Anna Irrera. *UBS to Open Blockchain Research Lab in London.* 2nd Apr. 2015. URL: http://blogs.wsj.com/digits/2015/04/02/ubs-to-open-blockchain-research-lab-in-london/ (visited on 15/05/2015).

[37] Robert McMillan. *The Inside Story of Mt. Gox, Bitcoin's $460 Million Disaster*. 3rd Mar. 2014. URL: `http://www.wired.com/2014/03/bitcoin-exchange/` (visited on 15/05/2015).

[38] Brian Mikkelsen. *Lov om videresalg af billetter til kultur- og idrætsarrangementer*. 23rd May 2007. URL: `https://www.retsinformation.dk/forms/R0710.aspx?id=12058` (visited on 07/05/2015).

[39] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: `https://bitcoin.org/bitcoin.pdf` (visited on 14/05/2015).

[40] Dave Neal. *Ghash.io promises not to break the Bitcoin market*. 13th Jan. 2014. URL: `http://www.theinquirer.net/inquirer/news/2322534/ghashio-promises-not-to-break-the-bitcoin-market` (visited on 29/04/2015).

[41] Sanjay Panikkar et al. *ADEPT: An IoT Practitioner Perspective*. 7th Jan. 2015. URL: `http://www.scribd.com/doc/252917347/IBM-ADEPT-Practictioner-Perspective-Pre-Publication-Draft-7-Jan-2015` (visited on 15/05/2015).

[42] Yessi Bello Perez. *UBS to Research Blockchain Technology in London Lab*. 2nd Apr. 2015. URL: `http://www.coindesk.com/ubs-to-research-blockchain-technology-in-london-lab/` (visited on 15/05/2015).

[43] Andrew Poelstra. *On Stake and Consensus*. 22nd Mar. 2015. URL: `https://download.wpsoftware.net/bitcoin/pos.pdf` (visited on 18/05/2015).

[44] Rob Price. *Nasdaq is experimenting with the revolutionary technology behind bitcoin*. 11th May 2015. URL: `http://uk.businessinsider.com/nasdaq-private-market-blockchain-bitcoin-experiment-currency-ledger-2015-5?r=US` (visited on 15/05/2015).

[45] PricewaterhouseCoopers. *How financial services lost its mojo – and how to regain it*. 2nd Oct. 2014.

[46] Pete Rizzo. *Fidor Becomes First Bank to Use Ripple Payment Protocol.* 5th May 2014. URL: `http://www.coindesk.com/fidor-becomes-first-bank-to-use-ripple-payment-protocol/` (visited on 15/05/2015).

[47] Hernando de Soto. "This Land Is Your Land: A Conversation with Hernando de Soto". In: *World Policy Journal* (2011), pp. 35–40. URL: `http://www.worldpolicy.org/journal/summer2011/this-land-is-your-land` (visited on 07/05/2015).

[48] Investopedia Staff. *Case Study: The Collapse of Lehman Brothers.* URL: `http://www.investopedia.com/articles/economics/09/lehman-brothers-collapse.asp` (visited on 08/05/2015).

[49] Ian Thomson. *Danish court finds Pirate Bay cofounder guilty of hacking CSC servers.* 30th Oct. 2014. URL: `http://www.theregister.co.uk/2014/10/30/danish_court_finds_pirate_bay_cofounder_guilty_of_hacking_csc_servers/` (visited on 12/05/2015).

[50] Manny Trillo. *Stress Test Prepares VisaNet for the Most Wonderful Time of the Year.* 10th Oct. 2013. URL: `http://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html` (visited on 13/05/2015).

[51] Stephen Tual. *The case for (re)decentralizing the Internet.* 29th Apr. 2015. URL: `https://medium.com/ursium-blog/the-case-for-re-decentralizing-the-internet-724013c0622` (visited on 16/05/2015).

[52] V. Vaishnavi and B. Kuechler. *Design Science Research in Information Systems.* 20th Jan. 2004. URL: `http://www.desrist.org/design-research-in-information-systems/` (visited on 13/05/2015).

[53] Gavin Wood. *Ethereum Yellow Paper.* 2014. URL: `http://gavwood.com/paper.pdf`.

# List of Terms

**address** A unique identifier that in block chain technology identifies a user or other entity that can hold the related cryptocurrency. In Ethereum addresses can be either EOAs or smart contracts. IV, 16, 22, 23, 25–28, 30, 31, 35, 37, 38, 42, 48, 51, 70

**altcoins** Alternatives to Bitcoin. Cryptocurrencies that are very similar to Bitcoin. 8

**API** application programming interface. 21, 23

**ASIC** application-specific integrated circuit. 18

**block difficulty** The approximate number of tries it takes to hit an acceptable proof of work. 19

**block propagation** The amount of miners working on the wrong data due to them not being notified of the newest block yet. 45

**cryptocurrency** A currency that is based on cryptographic security. V, 1, 3, 7, 8, 11, 15, 36, 47–50, 54, 58, 61, 68, 69

**cryptographic economic system** An economic system that relies heavily on cryptocurrencies. 5

**DAO** Decentralised Autonomous Organisation. 3, 7, 36, 61

**distributed denial of service attack** A way of denying access to a service by bombarding it with mundane requests from a distributed network of machines. The amount of requests will result in real requests being pushed back in queue until they time out. 16

**DSR** Design Science Research. 4, 6

**EOA** externally owned account. 19, 68

**ether** The cryptocurrency used by Ethereum as both a currency and to fuel the computation of cryptocurrencies. 16, 17, 22, 26, 30, 35, 41, 43, 47, 48, 70

**EVM** Ethereum Virtual Machine. 15–19, 70

**fiat currency** Fiat currency is a currency which derives its value from government regulation or law. 8, 47–49, 58, 61

**futures contract** "In finance, a futures contract (more colloquially, futures) is a contract between two parties to buy or sell an asset for a price agreed upon today (the futures price) with delivery and payment occurring at a future point, the delivery date" `http://en.wikipedia.org/wiki/Futures_contract` (visited on 16/05/2015). 48

**getter** A function that does nothing but return specific data. 31, 35

**GUI** graphical user interface. 23, 30, 40

**IoT** Internet of Things. 1, 10, 22, 61

**ITU** IT University of Copenhagen. 2, 38, 42, 48

**man in the middle attack** When malicious software is used to catch and view or alter messages leaving a computer before they reach their destination. 51

**MVC** Model-View-Controller. 36, 37

**NSA** National Security Agency. 9

**opcode** An opcode is an instruction that specifies the operation to be performed. It is abbreviated from operation code. 19

**punch card** A physical card with a number of somehow removable parts (punches or clips) that each has a value in some specific product. In this case one punch equals one cup of coffee. IV, 5, 20–22, 24–26, 28–30, 32–36, 38–41, 44, 46, 48, 52, 59

**setter** A function that does nothing but change a specific field. 35

**shadow bank** A shadow bank is a non-bank financial intermediaries that provide services similar to traditional commercial banks. 57

**smart contract** An Ethereum address that holds EVM code that can be run through the use of transactions. 3, 5, 11, 12, 15–23, 25–32, 34–36, 38, 40, 41, 46, 48, 52, 59, 61, 68

**smart property** Property which ownership is controlled by a block chain protocol. 8, 11, 21

**state** In terms of block chain technology, a state is the complete availability of funds to the different addresses. This means that on the Ethereum protocol, a state describes how much ether each address has. 18, 43, 70

**substate** When a block chain is divided into smaller pieces, the state is divided as well. A substate is one of these divided pieces of the state. 42

**testnet** The block chain run by Ethereum prior to the release of the first commercially usable block chain. The testnet is unstable and unfinished. 29, 32

**UTXO** unspent transaction outputs. 42

**Web 3.0** Term used by Ethereum to describe a decentralised internet. 9, 46

**ĐApp** decentralised application. 11, 23, 46

# A | Implementation code

## A.1 Non-modular implementation

### A.1.1 Smart Contracts

Listing A.1: Smart contract code/contract.sol

```
contract ClipcardMachine {

  struct Clipcard {
    uint amountOfClips;
    bytes32 name;
  }

  mapping (address => Clipcard) clipcards;

  address issuer;
  address machine;
  uint price;
  uint newClips;

  function ClipcardMachine(address machineAddress) {
    issuer = msg.sender;
    machine = machineAddress;
    price = 1000000000000000;
    newClips = 10;
  }
```

```

    function deductClip(address beneficiary) {
       clipcards[beneficiary].amountOfClips -= 1;
    }

    function addClip(address beneficiary) {
       clipcards[beneficiary].amountOfClips += 1;
    }

    function buyClipcard(bytes32 name) returns (bool success) {

       if(msg.value < price || name == ""){
          return false;
       }

       Clipcard c = clipcards[msg.sender]; // assigns reference
        c.name = name;
        c.amountOfClips += newClips;

       return true;
    }

    function useClip() returns (bool success){
       if(clipcards[msg.sender].name == "" || clipcards[msg.sender
           ↪ ].amountOfClips < 1) {
          return false;
       }
       deductClip(msg.sender);

       Machine mach = Machine(machine);
       bool open = mach.open();
       if(!open) {addClip(msg.sender);}

       return true;
    }

    function setPrice(uint p) {
        if(msg.sender == issuer && p >= 0)
            price = p;
    }

    function setMachine(address a) {
        if(msg.sender == issuer)
            machine = a;
    }

    function setIssuer(address a) {
        if(msg.sender == issuer)
            issuer = a;
    }

    function emptyMachine(){
        if(msg.sender == issuer)
            issuer.send(this.balance);
    }

    function commitSuicide(){
        if(msg.sender == issuer)
            suicide(issuer);
    }

    function checkBalance() returns (uint balance) {
        Clipcard c = clipcards[msg.sender];
        balance = c.amountOfClips;
        return balance;
    }

    function getName() returns (bytes32 name){
        Clipcard c = clipcards[msg.sender];
        name = c.name;
        return name;
    }
}

contract Machine {
    /**~\label{line:modular_machine}~
      * An artificial representation of a Machine that gets opened
         ↪ when a punch is registered
      *
      */
    address issuer;
    address owner;
    uint public openTill;
    uint public cupsPoured;

    // Empty constructor
    function Machine(){
        owner = msg.sender;
    }

    function setIssuer(address addr){
        if(msg.sender == owner) {issuer = addr;}
    }

    function open() returns (bool result) {
        if(msg.sender != issuer) {return false;}
        if(cupsPoured == 10) {
            cupsPoured = 0;
            return false;
        }
        if(block.number > openTill) {openTill = block.number + 2;}
        else {openTill += 2;}
        return true;
    }

    function isOpen() returns (bool open) {
        return (block.number < openTill);
    }
}
```

### A.1.2 HTML JavaScript Client

Listing A.2: Client source code from code/index.html

```
<html>
<head>
    <meta charset="utf-8">
    <title>Kl&oslash;ppek&aring;rt</title>
    <link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/
       ↪ bootstrap/3.2.0/css/bootstrap.min.css">
    <link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/
       ↪ bootstrap/3.2.0/js/bootstrap.min.js">
    <style>
        .form-control {
            width: 100% !important;
            text-align: center;
        }
        .btn {
            width: 100% !important;
            margin-bottom:40px;
        }
        .admin {
            display: none;
        }

        .loading {
            position: absolute;
            display: none;
            top:0px;
            bottom: 0px;
```

```
      left: 0px;
      right: 0px;
      background: url(1x1.png);
    }
    .loading img {
      position: absolute;
      margin-top: 200px;
      width: 240px;
      background: #FFF;
      border-radius: 5px;
    }
  </style>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery
     ↪ /2.1.3/jquery.min.js"></script>

  <script type="text/javascript">

    $(function(){
      if (typeof web3 !== 'undefined') {
        var width = $(window).width();
        margin = width/2;
        $("#loading").css('margin-left', margin-120 + "px");

        var contract = "0
            ↪ xc2624f1daad05afc81a3b2c8808af93962f6a473";
        var checkBalance = {
          "to": contract,
          "data": "0xc71daccb"
        }
        var checkName = {
          "to": contract,
          "data": "0x17d7de7c"
        }
        var ClipcardMachine = web3.eth.contract([{"constant":
            ↪ false,"inputs":[],"name":"getName","outputs":[{"
```

```
             ↪ name":"name","type":"bytes32"}],"type":"function
             ↪ "},{"constant":false,"inputs":[],"name":"useClip
             ↪ ","outputs":[{"name":"success","type":"bool"}],"
             ↪ type":"function"},{"constant":false,"inputs":[{"
             ↪ name":"name","type":"bytes32"}],"name":"
             ↪ buyClipcard","outputs":[{"name":"success","type"
             ↪ :"bool"}],"type":"function"},{"constant":false,"
             ↪ inputs":[{"name":"beneficiary","type":"address"
             ↪ }],"name":"addClip","outputs":[],"type":"
             ↪ function"},{"constant":false,"inputs":[{"name":"
             ↪ a","type":"address"}],"name":"setIssuer","
             ↪ outputs":[],"type":"function"},{"constant":false
             ↪ ,"inputs":[{"name":"p","type":"uint256"}],"name"
             ↪ :"setPrice","outputs":[],"type":"function"},{"
             ↪ constant":false,"inputs":[],"name":"emptyMachine
             ↪ ","outputs":[],"type":"function"},{"constant":
             ↪ false,"inputs":[],"name":"commitSuicide","
             ↪ outputs":[],"type":"function"},{"constant":false
             ↪ ,"inputs":[],"name":"checkBalance","outputs":[{"
             ↪ name":"balance","type":"uint256"}],"type":"
             ↪ function"},{"constant":false,"inputs":[{"name":"
             ↪ beneficiary","type":"address"}],"name":"
             ↪ deductClip","outputs":[],"type":"function"},{"
             ↪ constant":false,"inputs":[{"name":"a","type":"
             ↪ address"}],"name":"setMachine","outputs":[],"
             ↪ type":"function"},{"inputs":[{"name":"
             ↪ machineAddress","type":"address"}],"type":"
             ↪ constructor"}]);

         var machineInstance = ClipcardMachine.at(contract);

         var balance = web3.eth.call(checkBalance);
         var name = web3.eth.call(checkName);
         name = web3.toAscii(name);
         if(name) $("#cardname").val(name);
```

```
           if(!name) {
              name = "user";
              $("#back").hide();
           }
           var issuer = web3.eth.getStorageAt(contract,1);
           var machine = web3.eth.getStorageAt(contract,2);
           var price = web3.eth.getStorageAt(contract,3);
           var former_balance = balance;
           $("#map").html(web3.toDecimal(balance));
           $("#username").html(name);
           $("#contract").html(contract);
           $("#issuer").html(issuer);
           $("#machine").html(machine);
           $("#price").html(web3.fromWei(web3.toDecimal(price), " finney") + " finney");
           $("#newprice").val(web3.toDecimal(price));
           if(web3.eth.coinbase === issuer) $(".admin").show();
           /*if(web3.toDecimal(balance) <= 0) $("#useClip").prop('disabled', true);
           else                  $("#useClip").prop('disabled', false);*/


/*         var filter = web3.eth.filter('latest');
           console.log("2");

           filter.watch(function(error,result){
              console.log("3");
              $(".loading").hide();
              var balance = web3.eth.call(checkBalance);
              if(former_balance < balance) {
                 var name = web3.eth.call(checkName);
                 name = web3.toAscii(name);
                 $("#cardname").val(name);
                 $("#username").html(name);
```

```
            former_balance = balance;
         }
         var issuer = web3.eth.getStorageAt(contract,1);
         var machine = web3.eth.getStorageAt(contract,2);
         var price = web3.eth.getStorageAt(contract,3);
         $("#map").html(web3.toDecimal(balance));
         $("#issuer").html(issuer);
         $("#machine").html(machine);
         $("#price").html(web3.fromWei(web3.toDecimal(price),
            ↪  "finney") + " finney");
         if(web3.toDecimal(balance) <= 0) $("#useClip").prop
            ↪ ('disabled', true);
         else                  $("#useClip").prop('disabled
            ↪ ', false);
      });*/

      // This is due to web3.eth.filter not working as
         ↪ described in API


      setInterval(function update(){
         console.log("updated");
         $(".loading").hide();
         var balance = web3.eth.call(checkBalance);
         if(former_balance < balance) {
            var name = web3.eth.call(checkName);
            name = web3.toAscii(name);
            $("#cardname").val(name);
            $("#username").html(name);
            former_balance = balance;
         }
         var issuer = web3.eth.getStorageAt(contract,1);
         var machine = web3.eth.getStorageAt(contract,2);
         var price = web3.eth.getStorageAt(contract,3);
         $("#map").html(web3.toDecimal(balance));
```

```
          $("#issuer").html(issuer);
          $("#machine").html(machine);
          $("#price").html(web3.fromWei(web3.toDecimal(price), "finney") + " finney");
          $("#newprice").val(web3.toDecimal(price));
          /*if(web3.toDecimal(balance) <= 0) $("#useClip").prop('disabled', true);
          else                  $("#useClip").prop('disabled', false);*/
        }, 15000);



        $("#buycard").on('click', function(){
          if($("#cardname").val() !== ""){
            machineInstance.buyClipcard.sendTransaction($("#cardname").val(), {from: web3.eth.coinbase, value: price, gas: 1000000});
            $(".loading").show();
          }
          else
            alert("Input a name");
        });
        $("#useClip").on('click', function(){
          machineInstance.useClip({gas: 1000000});
          $(".loading").show();
        });
        $("#suicide").on('click', function(){
          machineInstance.commitSuicide({gas: 1000000});
          $(".loading").show();
        });
        $("#empty").on('click', function(){
          machineInstance.emptyMachine({gas: 1000000});
          $(".loading").show();
        });
```

```
            $("#newissuerbtn").on('click', function(){
              machineInstance.setIssuer.sendTransaction("0x" + $("
                  ↪ #newissuer").val(), {gas: 1000000});
              $(".loading").show();
            });
            $("#newmachinebtn").on('click', function(){
              machineInstance.setMachine.sendTransaction("0x" + $(
                  ↪ "#newmachine").val(), {gas: 1000000});
              $(".loading").show();
            });
            $("#newpricebtn").on('click', function(){
              machineInstance.setPrice.sendTransaction($("#
                  ↪ newprice").val(), {gas: 1000000});
              $(".loading").show();
            });
          }
          else {
            $(".container").html("<h1>This is a DApp - please open
                ↪ in compatible Ethereum client");
          }
        })


    </script>

</head>

<body class="container">
  <div class="header">
    <h3>Kl&oslash;ppek&aring;rt</h3>
  </div>
  <div class="jumbotron">
   <div class="row">
     <div class="col-md-12">
      <h4>Welcome<span id="back"> back</span>, <strong id="
```

```
            username"></strong>!</h4>
       <h5>My remaining clips: <strong id="map"></strong></h5>
       <h5>Issuer: <strong id="issuer"></strong></h5>
       <h5>Machine: <strong id="machine"></strong></h5>
       <h5>The Price: <strong id="price"></strong></h5>
      </div>
    </div>
    <div class="row">
       <div class="col-md-6">
          <button id="useClip" class="btn btn-default" type="submit">Punch your card</button>
       </div>
    </div>
    <div class="row">
       <div class="col-md-6">
          <input type="text" id="cardname" placeholder="Input name on clipcard" class="form-control" />
       </div>
       <div class="col-md-6">
          <button id="buycard" class="btn btn-default" type="submit">Top up your punch card</button>
       </div>
    </div>
    <div class="row admin">
       <div class="col-md-12">
          <h2>Administrative tools</h2>
          <h5>This contract is at: <strong id="contract"></strong></h5>
       </div>
       <div class="col-md-6">
          <button id="suicide" class="btn btn-default" type="submit">Commit suicide</button>
       </div>
       <div class="col-md-6">
          <button id="empty" class="btn btn-default" type="submit">
```

```html
            ↪ Empty machine</button>
        </div>
        <div class="col-md-1">
          0x
        </div>
        <div class="col-md-5">
          <input type="text" id="newissuer" placeholder="Address"
            ↪ class="form-control" />
        </div>
        <div class="col-md-6">
          <button id="newissuerbtn" class="btn btn-default" type="
            ↪ submit">Change owner</button>
        </div>
        <div class="col-md-1">
          0x
        </div>
        <div class="col-md-5">
          <input type="text" id="newmachine" placeholder="Address"
            ↪ class="form-control" />
        </div>
        <div class="col-md-6">
          <button id="newmachinebtn" class="btn btn-default" type="
            ↪ submit">Change machine</button>
        </div>
        <div class="col-md-6">
          <input type="text" id="newprice" class="form-control" />
        </div>
        <div class="col-md-6">
          <button id="newpricebtn" class="btn btn-default" type="
            ↪ submit">Change price</button>
        </div>
    </div>
  </div>


```

```html
<div class="loading"><img id="loading" src="loading.gif" /></div>
</body>

</html>
```

## A.2 Modular implementation

Listing A.3: Smart contracts for modular implementation

```
contract IsManaged {
  /**
    * Supercontract for all contracts that are managed by the
       ↪ Manager
    */
  address manager;

  // Set the manager for this contract
  function setManager(address managerAddr) returns (bool result){
    // Set a manager if none is set, or if the sender is the
       ↪ current manager
    if(manager != 0x0 && msg.sender != manager) {return false;}
    // Set the manager address
    manager = managerAddr;
  }

  // Kill the contract and send the funds to the manager
  function commitSuicide(){
    // Kill the contract and send funds to manager
    if(msg.sender == manager) {suicide(manager);}
  }
}

contract Issuer is IsManaged{
  /**
```

```
      * The Issuer is the main access point for this DApp.
      * It utilises the AssetHolder, Settings, Machine and NameDb
         ↪ directly and the PunchDb indirectly.
      * It is the only contract that should have a GUI.
      */
    address owner;

    // Constructor that sets the owner
    function Issuer() {
       owner = msg.sender;
    }

    // Function for buying a punch card. Accesses the AssetHolder
        ↪ and the NameDb
    function buyCard(bytes32 name) {
       if(manager != 0x0){
          // Get address of the assetHolder
          address assetHolder = Manager(manager).getComponent("
             ↪ assetHolder");

          // Try to restock card
          bool goon = AssetHolder(assetHolder).restockCard(msg.
             ↪ sender);
          // If restock failed, return
          if(!goon) {return;}

          // Get address of nameDb
          address nameDb = Manager(manager).getComponent("nameDb");
          // Set the name in the nameDb
          NameDb(nameDb).setName(msg.sender, name);
       }
    }

    // Function for punching a card. Accesses the AssetHolder and
        ↪ the Machine
```

```
  function punchCard() {
    if(manager != 0x0) {
      // Get address of assetHolder
      address assetHolder = Manager(manager).getComponent("
          ↪ assetHolder");

      // Try to punch card
      bool punched = AssetHolder(assetHolder).punchCard(msg.
          ↪ sender);
      // If punch failed, return
      if(!punched) {return;}

      // Get address of machine
      address machine = Manager(manager).getComponent("machine"
          ↪ );
      bool opened = Machine(machine).open();
      if(!opened) {
        bool unpunched = AssetHolder(assetHolder).unpunchCard(
            ↪ msg.sender);
      }
    }
  }

  // Function for setting the price. Accesses Settings
  function setPrice(uint value) {
    // Make sure sender is owner
    if(msg.sender != owner) {return;}
    // Make sure theres a manager
    if(manager != 0x0) {
      // Get the settings
      address settings = Manager(manager).getComponent("
          ↪ settings");
      Settings(settings).addSetting("price", value);
    }
  }

  // Function for getting the price. Accesses Settings
  function getPrice() returns (uint value) {
    // Make sure theres a manager
    if(manager != 0x0) {
      // Get the settings
      address settings = Manager(manager).getComponent("
         ↪ settings");
      return Settings(settings).getSetting("price");
    }
    return 0;
  }

  // Function for setting the add amount of punches. Accesses
     ↪ Settings
  function setAddAmount(uint value) {
    // Make sure sender is owner
    if(msg.sender != owner) {return;}
    // Make sure theres a manager
    if(manager != 0x0) {
      // Get the settings
      address settings = Manager(manager).getComponent("
         ↪ settings");
      Settings(settings).addSetting("addAmount", value);
    }
  }

  // Function for getting the add amount of punches. Accesses
     ↪ Settings
  function getAddAmount() returns (uint value) {
    // Make sure theres a manager
    if(manager != 0x0) {
      // Get the settings
      address settings = Manager(manager).getComponent("
         ↪ settings");
```

```
            return Settings(settings).getSetting("addAmount");
        }
        return 0;
    }

    // Function for setting the punch amount of punches. Accesses
        ↪ Settings
    function setPunchAmount(uint value) {
        // Make sure sender is owner
        if(msg.sender != owner) {return;}
        // Make sure theres a manager
        if(manager != 0x0) {
            // Get the settings
            address settings = Manager(manager).getComponent("
                ↪ settings");
            Settings(settings).addSetting("punchAmount", value);
        }
    }

    // Function for getting the punch amount of punches. Accesses
        ↪ Settings
    function getPunchAmount() returns (uint value) {
        // Make sure theres a manager
        if(manager != 0x0) {
            // Get the settings
            address settings = Manager(manager).getComponent("
                ↪ settings");
            return Settings(settings).getSetting("punchAmount");
        }
        return 0;
    }

    // Function for getting the punch balance. Accesses the
        ↪ AssetHolder
    function getBalance() returns (uint value) {
```

```
    // Make sure there's a manager
    if(manager != 0x0) {
      // Get the assetHolder
      address assetHolder = Manager(manager).getComponent("
        ↪ assetHolder");

      return AssetHolder(assetHolder).getBalance(msg.sender);
    }
    return 0;
  }

  // Function for getting the name. Accesses the NameDb
  function getName() returns (bytes32 name) {
    // Make sure there's a manager
    if(manager != 0x0) {
      // Get the assetHolder
      address nameDb = Manager(manager).getComponent("nameDb");

      return NameDb(nameDb).getName(msg.sender);
    }
    return "This issuer has no manager";
  }

  // Fucntion to set the owner
  function setOwner(address addr){
    if(msg.sender == owner) {owner = addr;}
  }

  // Function to empty the AssetHolder. Accesses the AssetHolder
  function empty(){
    if(msg.sender != owner) {return;}
    // Make sure there's a manager
    if(manager != 0x0) {
      // Get the address of the Asset Holder
      address assetHolder = Manager(manager).getComponent("
            ↪ assetHolder");

        AssetHolder(assetHolder).empty(msg.sender);
      }
    }
}

contract Machine is IsManaged {
    /**
      * An artificial representation of a Machine that gets opened
          ↪ when a punch is registered
      *
      */

    uint public openTill;
    uint public cupsPoured;

    // Empty constructor
    function Machine(){}

    function open() returns (bool result) {
      if(manager != 0x0) {
        address issuer = Manager(manager).getComponent("issuer");
        if(msg.sender != issuer) {return false;}
        if(cupsPoured == 10) {
          cupsPoured = 0;
          return false;
        }
        if(block.number > openTill) {openTill = block.number +
            ↪ 2;}
        else {openTill += 2;}
        return true;
      }
    }

    function isOpen() returns (bool open) {
        return (block.number < openTill);
    }
}



contract AssetHolder is IsManaged {
    /**
      * The AssetHolder handles the logic related to the punch
          ↪ cards and holds the value of the system
      * Can only be modified by a specified Issuer
      */
    // Empty constructor
    function AssetHolder(){}

    // Restock a card
    function restockCard(address addr) returns (bool result) {
        if(manager != 0x0) {
            // Check that the sender is the issuer
            address issuer = Manager(manager).getComponent("issuer");
            if(msg.sender != issuer) {return false;}

            // Get the settings
            address settings = Manager(manager).getComponent("
                ↪ settings");
            // Get the price
            uint price = Settings(settings).getSetting("price");
            // Get the add amount
            uint amount = Settings(settings).getSetting("addAmount");
            // Return if a setting is not set
            if(price == 0 || amount == 0) {return false;}

            // If the value of the message is too low, return the
                ↪ value and fail
        if(msg.value < price) {
          addr.send(msg.value);
          return false;
        }

        // Get the punch db
        address punchDb = Manager(manager).getComponent("punchDb");

        // Add the amount to the db.
        bool add = PunchDb(punchDb).add(addr, amount);
        return add;
      }
    }

    // Punch a card
    function punchCard(address addr) returns (bool result) {
      if(manager != 0x0) {
        // Check that the sender is the issuer
        address issuer = Manager(manager).getComponent("issuer");
        if(msg.sender != issuer) {return false;}

        // Get the settings
        address settings = Manager(manager).getComponent("settings");
        // Get the punch amount
        uint amount = Settings(settings).getSetting("punchAmount");
        // Return if the setting is not set
        if(amount == 0) {return false;}

        // Get the punch db
        address punchDb = Manager(manager).getComponent("punchDb");

        // Punch the card
        bool pun = PunchDb(punchDb).punch(addr, amount);
        return pun;
      }
    }

    // Unpunch a card
    function unpunchCard(address addr) returns (bool result) {
      if(manager != 0x0) {
        // Check that the sender is the issuer
        address issuer = Manager(manager).getComponent("issuer");
        if(msg.sender != issuer) {return false;}

        // Get the settings
        address settings = Manager(manager).getComponent("
            ↪ settings");
        // Get the punch amount
        uint amount = Settings(settings).getSetting("punchAmount"
            ↪ );
        // Return if the setting is not set
        if(amount == 0) {return false;}

        // Get the punch db
        address punchDb = Manager(manager).getComponent("punchDb"
            ↪ );

        // Unpunch the card
        bool add = PunchDb(punchDb).add(addr, amount);
        return add;
      }
    }

    // Get the balance
    function getBalance(address addr) returns (uint balance) {
      if(manager != 0x0) {
            // Get the punch db
            address punchDb = Manager(manager).getComponent("punchDb"
                ↪ );
            uint bal = PunchDb(punchDb).getBalance(addr);
            return bal;
        }
        return 0;
    }

    // Empty the Asset Holder
    function empty(address addr){
        if(manager != 0x0) {
            // Check that the sender is the issuer
            address issuer = Manager(manager).getComponent("issuer");
            if(msg.sender != issuer) {return;}

            addr.send(this.balance);
        }
    }
}

contract Settings is IsManaged {
    /**
     * Settings database
     * Can only be modified by a specified Issuer
     */
    mapping (bytes32 => uint) settings;

    function Settings() {}

    function addSetting(bytes32 name, uint setting){
        // Make sure there's a manager
        if(manager != 0x0){
            // Get the issuer
            address controller = Manager(manager).getComponent("
            ↪ issuer");
        // If the issuer sent the message, update the setting
        if(msg.sender == controller) {settings[name] = setting;}
      }
    }

    function getSetting(bytes32 name) returns (uint setting){
      return settings[name];
    }
}

contract PunchDb is IsManaged {
    /**
      * Database to handle punches on punch cards
      * The PunchDb can only be modified from a specified
          ↪ AssetHolder
      */
    mapping (address => uint) punchDb;

    // Empty constructor
    function PunchDb(){}

    // Adds punches to a punch card
    function add(address addr, uint newClips) returns (bool result)
        ↪  {
      // Make sure there's a manager
      if(manager != 0x0){
        // Get the controller of the PunchDb
        address controller = Manager(manager).getComponent("
            ↪ assetHolder");
        // If the controller sent the message, update the punch
            ↪ card
        if(msg.sender == controller) {
          punchDb[addr] += newClips;
          return true;
        }
      }
      return false;
    }

    // Deduct punches from a punch card if possible
    function punch(address addr, uint punchClips) returns (bool
       ↪ result) {
      // If the balance is too low, return
      if(punchDb[addr] < punchClips) {return false;}
      // Make sure there's a manager
      if(manager != 0x0){
        // Get the controller of the PunchDb
        address controller = Manager(manager).getComponent("
           ↪ assetHolder");
        // If the controller sent the message, update the punch
           ↪ card
        if(msg.sender == controller) {
          punchDb[addr] -= punchClips;
          return true;
        }
      }

      return false;
    }

    // Gets the balance of a punch card
    function getBalance(address addr) returns (uint balance) {
      return punchDb[addr];
    }
}

contract NameDb is IsManaged {
    /**
     * Database to handle names in the system
     * The NameDb can only be modified by a specified Issuer
     */
    mapping (address => bytes32) nameDb;

    // Empty constructor
    function NameDb(){}

    // Set a name in the database
    function setName(address addr, bytes32 name){
      // Make sure there's a manager
      if(manager != 0x0){
        // Get the controller of the NameDb
        address controller = Manager(manager).getComponent("
           ↪ issuer");
        // If the controller sent the message, update the name
        if(msg.sender == controller) {nameDb[addr] = name;}
      }
    }

    // Get a name from the database
    function getName(address addr) returns (bytes32 name) {
      // Make sure there's a manager
      if(manager != 0x0) {
        // Get the controller og the NameDb
        address controller = Manager(manager).getComponent("
           ↪ issuer");
        // If the controller sent the message, return the name
        if(msg.sender == controller) {return nameDb[addr];}
      }

      return "";
    }
}

contract Manager {
```

```
/**
  * The manager contract is responsible for handling all the
      ↪ components of the Punch Card Issuer system
  */
address owner;
mapping (bytes32 => address) contracts;

// Constructor for the Manager
function Manager(){
   owner = msg.sender;
}

// Adds a component to the system
function addComponent(bytes32 name, address comp) returns (bool
    ↪  result){
   // If sender is not owner, fail
   if(msg.sender != owner) {return false;}

   // Try to set this as manager in contract
   IsManaged managed = IsManaged(comp);
   bool sm = managed.setManager(address(this));

   // If this fails, fail
   if(!sm) {return false;}

   // Store in mapping
   contracts[name] = comp;
   return true;
}

// Removes a component from the system
function removeComponent(bytes32 name) returns (bool result){
   // If the component is not set, fail
   if(contracts[name] == 0x0) {return false;}
   // If the sender is not the owner, fail
```

```
        if(msg.sender != owner) {return false;}

        // Set the contract to nothing
        contracts[name] = 0x0;
    }

    // Kills a component by making it commit suicide
    function killComponent(bytes32 name) {
        if(owner != msg.sender) {return;}
        IsManaged(contracts[name]).commitSuicide();
        owner.send(this.balance);
        contracts[name] = 0x0;
    }

    // Get a component
    function getComponent(bytes32 name) returns (address addr){
        // Method to check that all parts have been set
        return contracts[name];
    }

    // Change the owner of the Manager
    function setOwner(address addr){
        if(msg.sender == owner) {owner = addr;}
    }

}
```

# B | Test Results

In figure B.2b on page 102 the results of the black box testing of the proof-of-concept system.

| Action | Case | Expected | Result | ✓ |
|---|---|---|---|---|
| Buying Punch Card | No funds | No change | No change | ✓ |
| | Just under price | No change | No change | ✓ |
| | Exactly enough funds | No change[(a)] | No change | ✓ |
| | Exactly enough + fees | + 10 clips | + 10 clips | ✓ |
| | Plenty of funds | + 10 clips | + 10 clips | ✓ |
| | No name | No change | No change | ✓ |
| | With name | + 10 clips | + 10 clips | ✓ |
| | With other name | + 10 clips + name | + 10 clips + name | ✓ |
| Punch card | No clips | No change | No change | ✓ |
| | 1 clip | - 1 clip | - 1 clip | ✓ |
| | Plenty of clips | - 1 clip | - 1 clip | ✓ |
| Setting price | Negative no. | No change[(b)] | $1.15792e^{+87}$ | |
| | 0 | 0 price | 0 price | ✓ |
| | Positive no. | Price = no. | Price = no. | ✓ |
| Setting machine | No address | 0x0 address | 0x0 address | ✓ |
| | An address | Updated address | Updated address | ✓ |

Table B.1: Table of black box test results

(a) No change is expected as you are also responsible for covering transaction fees.

(b) This test fails as an unsigned integer is sent as parameter. These does not allow negative numbers, and is thus changed to a very high positive number instead.