Universiteit Leiden
Opleiding Informatica

Microgrid blockchain

enabled solar panel electricity trading

Name: Timon Bakker

Date: 25/01/2019

1st supervisor: Dr. A.W. Laarman

2nd supervisor: Alexandra Pitkevich MSc, Accenture

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The Energy Market is changing. Through solar panels, wind turbines and other self-sufficient renewable energy methods consumers get more power over the energy they are consuming. However, while the market is changing, consumers do currently not have the power to decide to who they want to sell their self-generated excess of renewable energy. The only buyer is the utility company, against a price they set. By using the blockchain technology to improve this situation we created a solution to enable consumers to sell and buy renewable energy through a decentralized trading platform.

With the blockchain technology, we created a physical small scale demo of a renewable energy trading platform. This implementation is created using Raspberry Pi's, an Ethereum private blockchain, INA219 current sensors, solar panels and batteries. This resulted in a efficient implementation of a trading platform which enables customers to sell their renewable energy to each other, at their own rate.

Acknowledgements

I would like to thank Accenture, Sebastiaan Raven and especially Alexandra Pitkevich for the opportunity to work on this project. As well as the opportunity to join the Blockchaingers Hackathon 2018 team, an experience I will never forget. I would also like to thank my supervisor Alfons Laarman from the Leiden Institute of Advance Computer Science (LIACS) for his help, guidance and feedback on this thesis.

Contents

1 Introduction

The energy market is changing. The time that the utility companies sold their energy to the consumers is in the past. Through solar panels and wind turbines consumers have more, and cheaper, options to generate there own renewable energy. With these developments consumers get more power in the type and amount of electricity they generate, consume and sell. Currently, the generated electricity is deposited back into the main grid and deducted from the bill. The excess of generated electricity is automatically sold back to the utility company at a lower rate then the rate the consumer is paying.

Since the beginning of the internet, there has been an trust issue: "How can I trust the information that I receive?". How can I be sure that the information I'm reading is the same as uploaded by the author. This trust issue can be partially solved by trusting a third party. For example, trusting your bank to handle your transactions and showing the amount of money in your bank account. Blockchain can tackle this trust problem without a third party, through the use of cryptography techniques the blockchain technology can make sure that the integrity of data is kept.

So what if a consumer decides he doesn't want to sell his excess generated electricity to the utility company? That's currently not an option. The utility company is the only possible buyer, as well as the party deciding the price that is paid. We are missing a trusted *decentralized* online platform for consumers to sell energy. To trade their generated electricity independent of any utility company. Such gives consumers the power to buy and sell electricity at the rate that arises on the market through free trade.

The blockchain technology could offer a solution. Through the *decentralized* property of the blockchain no third party has the power to interfere with your transactions. By using *smart contracts* consumers will be able to interact with the blockchain and be able to freely sell their electricity to anyone they want as well as buy from anyone they want. This way, consumers would be able to get a market balance price for their excess generated electricity instead of the fixed price rates of the utility companies.

In collaboration with Accenture a small scale demo is created to solve this problem. The demo simulates the electricity a household generates, consumes and trades. A *smart meter* is created to measure all incoming and outgoing electricity. Combined with a *smart contract* transactions are executed automatically on the blockchain. A dashboard is created to enable consumers to see their current flows and change the rates they are willing to pay and would like to receive for their electricity. The demo is created by using a Raspberry Pi, Ethereum blockchain and multiple current sensors.

This demo shows that it is indeed possible to enable the decentralized trading of electricity. By using the blockchain technology it is possible to efficiently implement the automated trading of electricity and give consumers the power over their price rates. This small scale demo shows potential for the future of the energy market and an efficient way to reduce the power of the utility companies. As well as providing a more competitive market for every energy user and provider. However, this demo also exposes limitations of blockchain. The technology does not seem entirely fit for the proposed solution. An implementation with less that 50 households would be vulnerable due to the low amount of *nodes* and the benefits of the blockchain technology would no longer hold between other, simpler, implementations.

In Chapter 2 of this thesis the current energy market is explained. And how renewable energy is recognized, validated and traded. Chapter 3 explains the history of the blockchain technology, the different types and a comparison of the different *consensus mechanisms*. In Chapter 4 the implementation is evaluated and certain design choices are explained. In Chapter 5 the implementation of the proposed solution is explained by design, software and physical implementation. Chapter 6 shows the results of the implementation supported by pictures of the physical demo and user dashboard. Chapter 7 evaluates the research and proposes possible subjects for future research.

2 Energy Market

In this chapter, we explain the current energy market. How renewable energy is recognized, validated and traded. As well as the arguable problems with the current market. Several solutions are being looked at from which a few are already implemented on small scale.

All electricity in the power grid is identical. Once electricity is delivered into the grid there is no way to distinguish the delivered electricity from the rest. But, some people are willing to pay more money for renewable energy than energy from an coal-fired power station or nuclear power plant. To enable the trade of renewable energy *Renewable Energy Certificates* (REC's) were invented. A renewable energy provider receives 1 REC for each 1,000 kWh renewable energy delivered to the main grid. These certificated can be traded on the market like stocks or bonds.

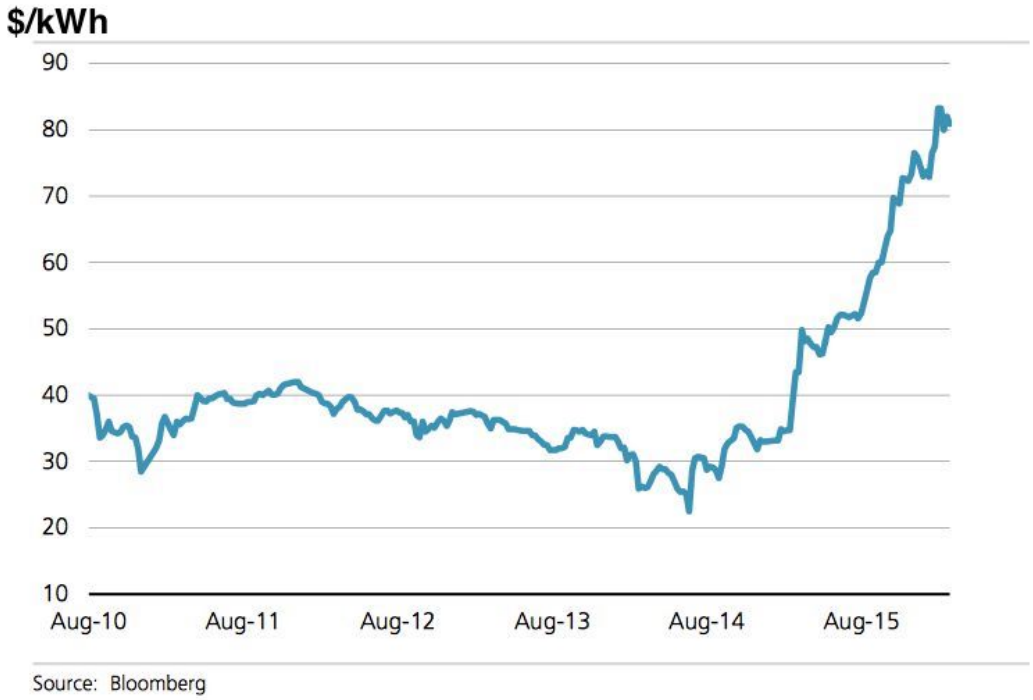


Figure 1: Trading price of REC's

By buying REC's consumers are able to buy renewable energy which would otherwise be not possible. For example, when you live in the city you can buy a REC generated with wind turbines on the other side of the country. However, that the electricity is delivered into the main grid doesn't mean that you consume this renewable energy. Because the electricity on the grid is identical, it is not possible to tell where your electricity exactly came from. The trade of REC's makes it possible to buy a certificate from energy generated in a solar park in Norway while the energy you actually consume is from the coal-fired power plant station a few blocks away. You can buy renewable energy, but it doesn't mean it is generated even remotely close to you.

On a big scale there is nothing wrong with this solution, because the renewable energy one buys is still delivered into the main grid. So somewhere you're making the world a better place. But what if some countries give more subsidy to install wind turbines and solar panels then other countries? Then companies in these countries can generate cheaper renewable energy. This way these companies can sell there REC's at a low price to other countries and therefore demotivate those countries to invest in there own renewable energy sources. The renewable energy is generated somewhere, but your street, city or even country is generally not benefiting from it.

In The Netherlands there is the possibility for consumers to sell electricity back to the utility company. These companies are obligated to subtract the amount of *kilowatt hour* (kWh) you generated in excess from your bought kWh in the same period, called *netting*. So they pay you as much for each kWh as you pay them. But when you generate more energy then you consume, the utility company gets to decide the price you receive for the excess generated electricity. At the moment of writing you pay €0,23 for each kWh and receive €0,07 for each kWh you return more then you consume[13]. There is also a *netting limit* which limits you to the amount of renewable energy you can return to the main grid. It is understandable that utility companies make costs by receiving you excess electricity and sending it to other consumers. But, these restrictions are not future-proof with the changing energy market.

Because of these restrictions house owners are not stimulated to invest in more solar panels or wind turbines then they need for personal use. The average payback period on solar panels with netting is 10 years. Without netting the expected payback period exceeds 25 years, more then the expected lifespan of solar panels[5]. It is expected that in 2020 netting is no longer allowed by the Dutch Government, making investments in personal renewable energy sources even more uninviting.

Several solutions have been proposed and implemented in the last years to partially solve these problems. For example, investing in solar panels on farmers barns or becoming partial owner of a industrial wind turbine. But, the problem with private owned solar panels and wind turbines remains unsolved.

In Chapter 5 a solution is proposed and implemented to solve these problems. A similar solution has been implemented in the last years: The Brooklyn MicroGrid. A solution where a community stores and trades renewable energy. The Brooklyn MicroGrid projects focuses on trade within communities and the upkeep of electricity in case of a natural disaster. Whereas the solution proposed in this thesis focuses on implementing a free market for electricity trade. The transactions in the Brooklyn MicroGrid are implemented by using the blockchain technology as well[10].

3 Blockchain

In this chapter, we give an introduction in the blockchain technology. The history, basic concept and different consensus mechanisms are explained to give a basic understanding of the technology before the software implementation of the demo described in Chapter 5.

3.1 Introduction

Blockchain is a technology to enable *decentralized* secure storage of records by using cryptography. The records are stored in *blocks* which are linked together, hench the name *blockchain*. It can be seen as an distributed ledger which keeps track on certain data mutations. Each *block* contains three key ingredients:

- cryptographic hash of the previous block
- timestamp
- data

A blockchain is typically managed using a peer-to-peer network where all users, called *nodes*, keep a complete record of the ledger of mutations. By the use of a *consensus mechanism* the nodes agree on how the blockchain is updated. Special nodes, called *miners*, are used to bundle the hashed data mutations in *blocks*. Blocks are immutable except for their nonce value. Miners manipulate the nonce value until the hashed value of the block falls in a certain pre-defined range. When the first miner gets a correct value the other miners check if it's correct. Checking is easy because of the properties of hashes. Hashes are *one-way functions* which are easy to compute on every input but hard to invert[19]. Therefore it is easy to check if a hashed value is correct given the used nonce. When a miner confirms the hash he adds the block to his version of the blockchain. When more and more miners find the same value and add the block to their chain the block is slowly 'accepted'. When 51% of the computational power agrees on the same value it is generally accepted as the truth and added to every nodes ledger. This is call the *Proof-of-Work* consensus mechanism.

The first block in a blockchain is called the *Genesis Block*. This block is created by the developer of the blockchain implementation and contains all important meta information about the blockchain. For example, the difficulty of the hash, the time between the creation of new blocks, pre-funded accounts and the maximum amount of tokens available. Pre-funded accounts and tokens are only relevant in a blockchain implementation as cryptocurrency, for example *Bitcoin*, where the tokens represent some kind of value. Through pre-funding the developer can implement a certain amount of tokens for accounts at the start of the blockchain implementation. Because the genesis block is the first block it doesn't contain the *hash value* of the previous block, because there isn't one.

Figure 2 shows a schematic view of the first blocks within a blockchain. In this figure the data inside the block consists of transactions, which are stored by using Merkle Trees. More about Merkle Trees is explained in Chapter 3.2.

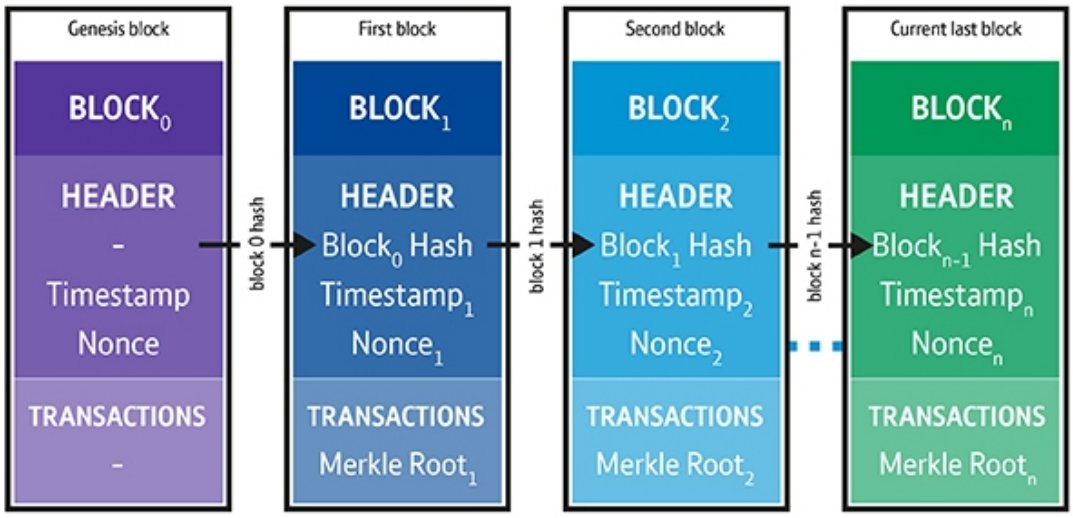


Figure 2: First blocks of blockchain implementation

To change the data inside a block, one node needs to convince other nodes his mutated version of the block is the correct one. In other words, his wrong hash value has to be accepted by all others nodes. This explains the verification power of a blockchain. The more nodes there are, the harder it is to get 51% of the computation power to alter the data inside the blocks. Because each block contains the hash value of the previous block, the more blocks there are the safer the previous blocks. Because to alter an older block the block has to be mutated and the hash of every following block has to be computed and verified again.

So why would someone spend computation power and energy on finding the correct hash? With a *Proof-of-Work* Blockchain implementation the miner who first finds the correct hash gets a reward. the reward is in tokens, which are defined in the Genesis Block. In the case of Bitcoin, the reward is around 12.5 BTC (Bitcoin) for each block. This may not sound like a lot, but with the current exchange rate this is almost $\$45.000$.

3.2 History of Blockchain

The first notice of a cryptographically secure chain of blocks was in 1991. Stuart Haber and Scott Stornetta described how to securely time-stamp a digital document. Their solution was to hash a new document with his own value as well as the hashed value of the previously hashed document. This way the documents are 'chained' together. So to manipulate one document in the chain all the upcoming documents needed to be hashed again as well. Otherwise the hashed values would be incorrect and you can deduce there was tempering with the files[7].

In 1992 Haber and Stornetta, in cooperation with Dave Bayer, improved the concept by adding Merkle Trees to the design. This made the 'chain' a lot more efficient because of the possibility to save multiple files in the same 'block'. The original idea is the same: chain the blocks together to improve security[1]. Merkle Trees, also know as hash trees, are trees where every leaf node is labeled with the hash value of its data and every non-leaf node is labeled with the hash of the labels of its children[9]. Merkle Trees can be used to verify data transferred between different computers. As today, data within blockchain blocks is still stored using Merkle Trees.

The rapid development of the blockchain technology came in 2008 when a group or person by the name of *Satoshi Nakamato* created blockchain. Followed the next year by the implementation in the Bitcoin cryptocurrency. It was the first digital currency to solve *double spending* as well as the most popular cryptocurrency to date[4]. Double spending is the possibility of falsifying or duplicating digital money. Bitcoin was the first to manage this problem by using a *confirmation mechanism* and a *universal ledger*. Because every transaction and Bitcoin is counted for, it is not possible to duplicate the coins[2].

In 2014 the term *Blockchain 2.0* was created to distinguish between the more *classical* blockchain technology and the newer ones. Blockchain 2.0 can be seen as a 'new generation' blockchain with more sophisticated and new features. One of the most relevant features is that the blockchain 2.0 allows for programmable transactions. This gives the possibility to create so-called *smart contracts* to automatically and decentralized transfer *cryptocurrency* when pre-defined conditions are met.

For example, a blockchain could be implemented and programmed to ensure save package delivery. If each package carries a GPS-tracker, then when the package is delivered the delivery fee can be automatically transferred to the carrier. The blockchain could be programmed to make a transaction iff the GPS-tracker of the package matches the destination address and the receiver gave a signature. Through the use of smart contracts this transaction can be completed without the need of a third party.

3.3 Public, Private & Hybrid Blockchains

The blockchain technology makes a distinction between public and private domains. Both have many similarities and are often misconstrued. They are both decentralized peer-to-peer networks which use consensus methods to keep the nodes in sync and thereby guarantee immutability of the ledger[8]. However, recently a third domain has been suggested: hybrid. Which tries to combine the best of both worlds. The different types are schematically shown in figure 3.

In a *public blockchain* the network is open and anyone who wants can join as a node, for example Bitcoin. There is no controlling party that decides who can and who can't join the network. It is completely *decentralized*. A disadvantage is the amount of computational power needed to reach consensus. This makes the blockchain more secure because it is harder for one party to influence or mutate the ledger but it costs more computational power to calculate. The public ledger makes it possible for everyone to see each transaction. When you know the address of an user you can look up all transactions concerning this user. This transparency can be seen as safe, but as an user you lose your financial privacy. Your transactions and balance are public. Imagine that your money on the bank is safe, but everyone can see your bank statements. One solution to claim back your privacy is the use of a new address for each transaction. Your *wallet* can bundle all the addresses to make the implementation and use easy for the end-user.

In a *private blockchain* it is not possible to freely join the network as a node. You need either an invitation from the *network starter*. Or be invited by a set of rules put in place on creating the network. It is also a possibility that existing *nodes* can create new nodes. The set of rules can be implement using a smart contract. Because of these features a private blockchain is more suitable for implementation of Internet of Things (IoT) solutions then a public blockchain. Solutions where decentralization and transparency are not a high priority. An example of a private blockchain is Hyperledger. *Hyperledger Fabric* is a open-source private blockchain developed by IBM which can be used for developing enterprise solutions.

A *hybrid blockchain* is best described as a combination between a public and private blockchain. Trying to combine the best aspects of both worlds. A hybrid blockchain uses an public ledger to make transactions, so everyone can see the transactions. But the consensus process is confined to a pre-defined set of nodes, similiar to a private blockchain.

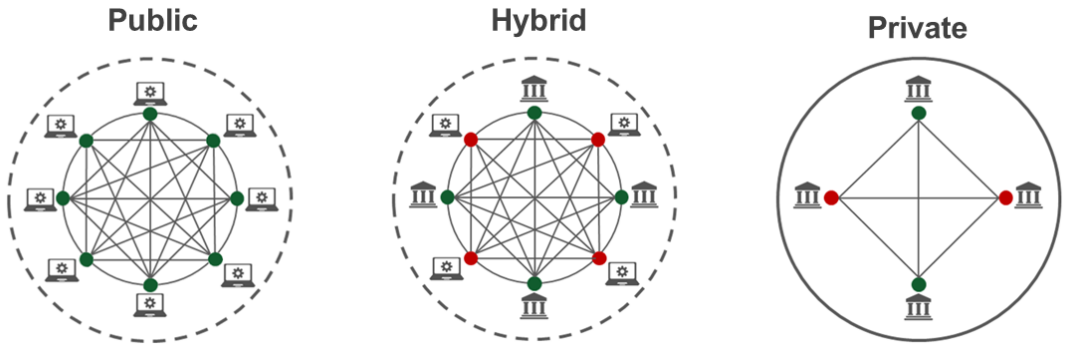


Figure 3: Schematic overview of different types of blockchain

3.4 Consensus mechanisms

To reach consensus on what block is valid to add to the chain, different *consensus mechanisms* have been developed. What mechanisms to use depends on trust, decentralization, scale and the application itself. All mechanisms can be useful for different applications because not every blockchain has the same purpose or strives for the same level of security. For this and other reasons different consensus mechanisms where developed. The three most used consensus mechanisms are:

- Proof-of-Work
- Proof-of-Stake
- Proof-of-Authority

3.4.1 Proof-of-Work

Proof-of-Work (PoW) is the consensus mechanism used in the first blockchain implementations. With the Proof-of-Work mechanism the miner gets two things. An amount of data (transactions) ready to become the next block and a pre-defined hash property. An example of a hash property is that the first five characters of the hash value have to be 00000. To change the hash-value the miner has the possibility to change a value within the block, the *nonce*. By changing the nonce the miner alters the data in the block and therefore the hash-calculation outcome. Miners try different nonce-values as fast as possible to find a hash value to match the requirement. Once a correct nonce is found it is verified by other miners. The first to find a correct nonce value receives a *mining reward* which depends on the specifications given in the genesis block. The more difficult the consensus algorithm, the rarer the hash property. The difficulty usually changes dynamically by altering the block-mining rate. The downside of the Proof-of-Work mechanism is the need of computational power. Because multiple miners try to find a correct nonce as fast as possible a lot of *double work* is done. Figure 4 shows the properties of a Proof-of-Work block.

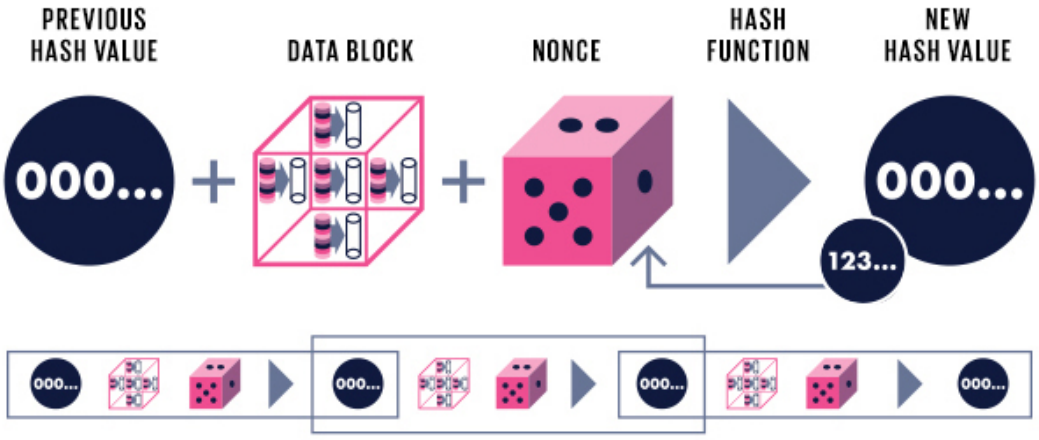


Figure 4: Proof-of-Work consensus mechanism

3.4.2 Proof-of-Stake

Proof-of-Stake (PoS) was created as an answer to the problems and inefficiency of the Proof-of-Work consensus mechanism. Instead of *brute-force* mining to find the solution of a block, the Proof-of-Stake mechanism works with validating. Not every node has the option to create the next block, something that is the case with Proof-of-Work. Instead the creator of the next block is chosen by the Blockchain based on several features. Usually the wealth (*stake*) and *age* of the node. The block is then *minted* instead of mined. The nodes who create the new blocks are called *forgers*. To make sure the node validates the block correctly his coins are held in an *escrow account* until the block is validated. Because there is no computational power needed, forgers are not rewarded for there creation of the new block. Instead they receive the transaction fees paid for the transactions within the block. The concept is based on the assumption that an user who is a big stakeholder will have an incentive to be honest and therefore validate the block correctly. It also solves the inefficient use of computation power, and therefore energy, with the Proof-of-Work mechanism. One possible problem with this consensus mechanism is that the rich are getting richer. Because only the users with high stakes get to validate new blocks and receive transaction fees they are the only ones to receive free coins. Figure 5 shows the properties of a Proof-of-Stake block.

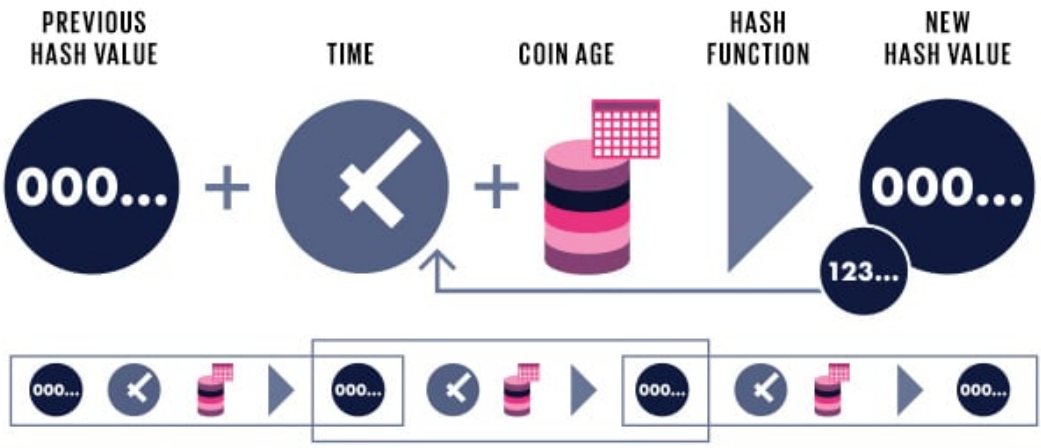


Figure 5: Proof-of-Stake consensus mechanism

3.4.3 Proof-of-Authority

Proof-of-Authority (PoA) is the most recent development in blockchain consensus mechanisms and proposed as an optimized version of the Proof-of-Stake mechanism[16]. It is an compromise between complete decentralizated and more efficient, centralized consensus mechanisms. Proof-of-Authority pre-approves certain nodes as trusted nodes called *validators*. These nodes have the authority to verify transactions and creating new blocks. The group of validators is supposed to be fairly small, less then 25 nodes. This ensures efficiency and manageable security of the network[16]. It solves an important problem of the Proof-of-Stake mechanism: that wealth as stake is different for every user. 20 tokens might be a lot of money for one user, but negligible for another one.

Therefore there incentive is different, but the mechanism thinks they are the same. Proof-of-Authority solves this by using your identity as stake instead of your invested wealth. The application for this consensus mechanism is most effective for permission private Blockchain implementations. For example, corporate solutions. The popularity of this consensus mechanisms has risen quickly because of the implementation in *Ethereum*, once of the most popular Blockchain implementations today.

3.5 Ethereum

Ethereum is an open-source, public blockchain and the most well known Blockchain 2.0 implementation[20]. Ethereum is developed to create an alternative protocol for building decentralized applications[15]. Ethereum differs from more traditional blockchain implementations like Bitcoin by the built-in Turing-complete programming language: *Solidity*. Which enables developers to create and deploy applications on the blockchain. Solidity makes it possible to create arbitrary rules for automated transactions called *smart contracts*. The cryptocurrency on the Ethereum blockchain is called *Ether*. Which can be traded between accounts. Ethereum enables developers to create private blockchain implementations using the Ethereum protocol.

Ethereum uses a proof-of-work consensus mechanism that, unlike Bitcoin, is ASIC-resistant. ASIC stands for Application Specific Integrated Circuits. By being ASIC-resistant, Ethereum makes mining with expensive special purpose hardware less useful and focuses on general purpose hardware. This design choice keeps up a more decentralized network because every PC owner can mine without needing specialized hardware. Each Ethereum account has a 20-byte address which is unique for each account. Enabling transactions between accounts. Each account keeps track of a *nonce* to make sure each transaction is only processed once, as well as a coin balance called the *ether balance*. If an smart contract is deployed then this is stored in the concract field of the account. Otherwise this field is empty.

4 Design Decisions

In Chapter 1 an introduction is given on the problem where Chapter 2 introduces the current energy market. So, is the blockchain technology suitable for changing the energy market? In this chapter the different design choices made for the demo are discussed. The implementation of the demo is explained in Chapter 5.

For the physical implementation the Raspberry Pi is chosen because of the possibilities with the GPIO connectors and the availability of many software libraries. It is also possible to implement the demo using an Arduino or other suitable micro-computer. However, with the convenience of the Raspberry Pi to support Linux it is possible to implement the physical and software part of the demo on one device.

The INA219 current sensors are chosen because of the low price and the compability with the Raspberry Pi, especially through the use of the *pi-ina219* library written by Chris Borril[3]. However, the INA219 sensors are not the first choice. The newer INA3221 current sensor offered the same functionality with the additional benefit of measuring three channels at the same time. So instead of using multiple INA219 sensors it is be possible to implement the demo with one INA3221 sensor. However, due to the lack of availability in The Netherlands the INA219 is chosen.

The solar panel and charge controller are both derived from an power bank designed for charging mobile phones. Because of the low price and bad building quality (easy to take apart) these power banks are suitable for implementing the physical part of the demo. Because of the low capacity of the power bank battery (140mAh, 3.7v) a slightly larger battery is used in the final implementation.

All parts are connected by using 10cm Dupont breadboard cables. These are chosen because of the compatibility with the Raspberry Pi GPIO connectors as well as the INA219 sensors. For connecting with the solar panel, charge controller and battery pins similiar to the GPIO connectors where soldered to the connecting points of those parts.

The dashboard is developed using HTML5, CSS3, PHP, AJAX and a mySQL database. All programming languages are chosen because of earlier experience working with these languages as well as compatibility with each other. The mySQL database is chosen because it is light and doesn't burden the Raspberry Pi microSD card to much. This because the microSD card has a limit on the reading and writing of the card. So for long-term implementation of the smart meter it is better to use a light database.

The blockchain is developed as a Private Ethereum Blockchain using Geth. The Go implementation of Ethereum[6]. Ethereum is chosen because of the support of smart contracts, the documentation and the possibility to implement a private blockchain of the protocol. As Ethereum is used, the smart contracts are written in Solidity. Currently the only programming language supported by Ethereum.

5 A Digital Energy Market Based on Blockchain

In this chapter we explain the implementation of the demo. The demo is based on the design choices made in the previous chapter. The design for the realization of this demo is made from the perspective of the house owner. Where the generated, consumed and surplus amount of electricity should be measured. And where the automated transaction should be made on the blockchain following the calculated excess or surplus in electricity.

5.1 Design

Each household is designed to track there own electricity generated and consumed. It should also calculate the excess or surplus in electricity, keep track of the wallet and interact with the blockchain. Therefore each household should contain two essential parts:

- Smart Meter
- Blockchain node

The *Smart Meter* can be seen as an upgrade on the *tradition electricity meter* which is upgraded to be compatible with current flowing two directions and is directly accessible from the internet. In The Netherlands over 3 million households have a smart meter installed with the goal to have one installed in every household by 2020[12].

The *blockchain node* is necessary in each household to interact with the blockchain. Enabling transactions to another household and keeping the integrity of the ledger. These transactions will be made automatically through interaction with the smart contract.

The Smart Meter and blockchain node are combined into one physical device, a micro-computer. Household owners need the possibility to look into there current usage and history. Therefore a dashboard is developed that communicates with the data in the smart meter and is accessible on the households local network. A schematic of the physical implementation can be seen in figure 6.

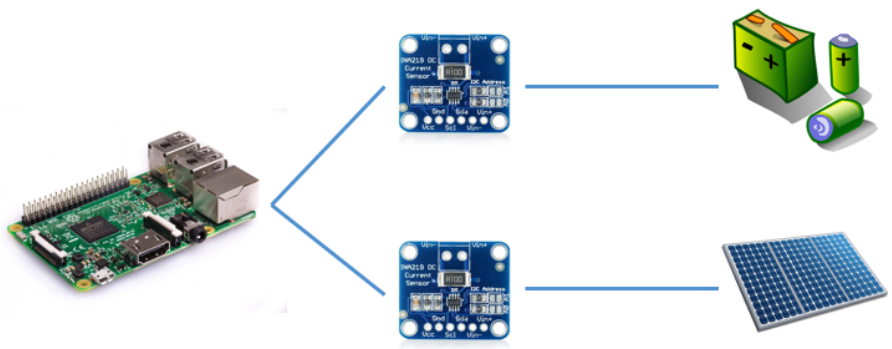


Figure 6: Simplified schematic of household implementation

5.2 Physical Implementation

For the physical implementation of the demo a Raspberry Pi, solar panel, battery and two INA219 sensors are used to simulate each house. The connection between the parts is shown through PCB in figure 7 and pictures in figure 11 and 12. The parts are more thoroughly explained below.

5.2.1 Raspberry Pi

The Raspberry Pi used is a model 2 B. Containing 1GB RAM and a 900MHz quad-core ARM Cortex-A7 CPU. Because of the low computational requirements of the Proof-of-Authority consensus mechanism every model of the Raspberry Pi could have been used. The Raspberry Pi represents one household and works as a smart meter, blockchain node and web server.

5.2.2 Solar Panel and Battery

The used solar panels are developed for the use on power banks for mobile devices. They can generate a power supply of 10 watt. The solar panel is connected to a loading meter to regulate the power supply to the battery. The sensors are connected between the load meter and battery. The battery is an 140mAh 3.7v Lithium-ion battery. Because of the low capacity of the battery a LED-lamp is used to demo the consumption of the household[11].

5.2.3 Current sensors

The current sensors used are INA219 sensors. These sensors are developed by *Texas Instruments* and are able to monitor voltage, current and power from one source up to *26 volts*. It has an 0.1 ohm shunt resistor. The maximum *voltage drop* that can be measured is 0.32 volts. By using the formula: $I = V/R$. Where I is the currency measured in *amperes*, V the voltage measured in *volts* and R is resistance measured in *ohms*. We can calculate a maximum currency of $0.32/0.1 = 3.2$ ampere. The sensor is connected to the I2C bus on the Raspberry Pi[14][17][18].

The Raspberry Pi only has only one I2C bus. Therefore the slave address of one sensor has to be adjusted. By changing the unique slave address of the sensors multiple can be connected to the Raspberry Pi. The standard address of the sensor is 1000000 or hex 40. By soldering the bridge A0 on the second sensor we can change the address to 1000001 or hex 41. Because we need two sensors no other adjustments where needed. Because there is an A0 bridge as well as an A1 bridge, it is possible to create up to four unique slave addresses on one I2C bus.

To measure the power usage, the voltage and the current have to be measured. Because we use the formula: $P = V \times I$. Where P is the power measured in *watts*. Because voltage is measured in a *series connection* and currency is measured in a *parallel connection* the sensor has to be connected both ways. Figure 7 shows the PCB overview of the physical implementation.

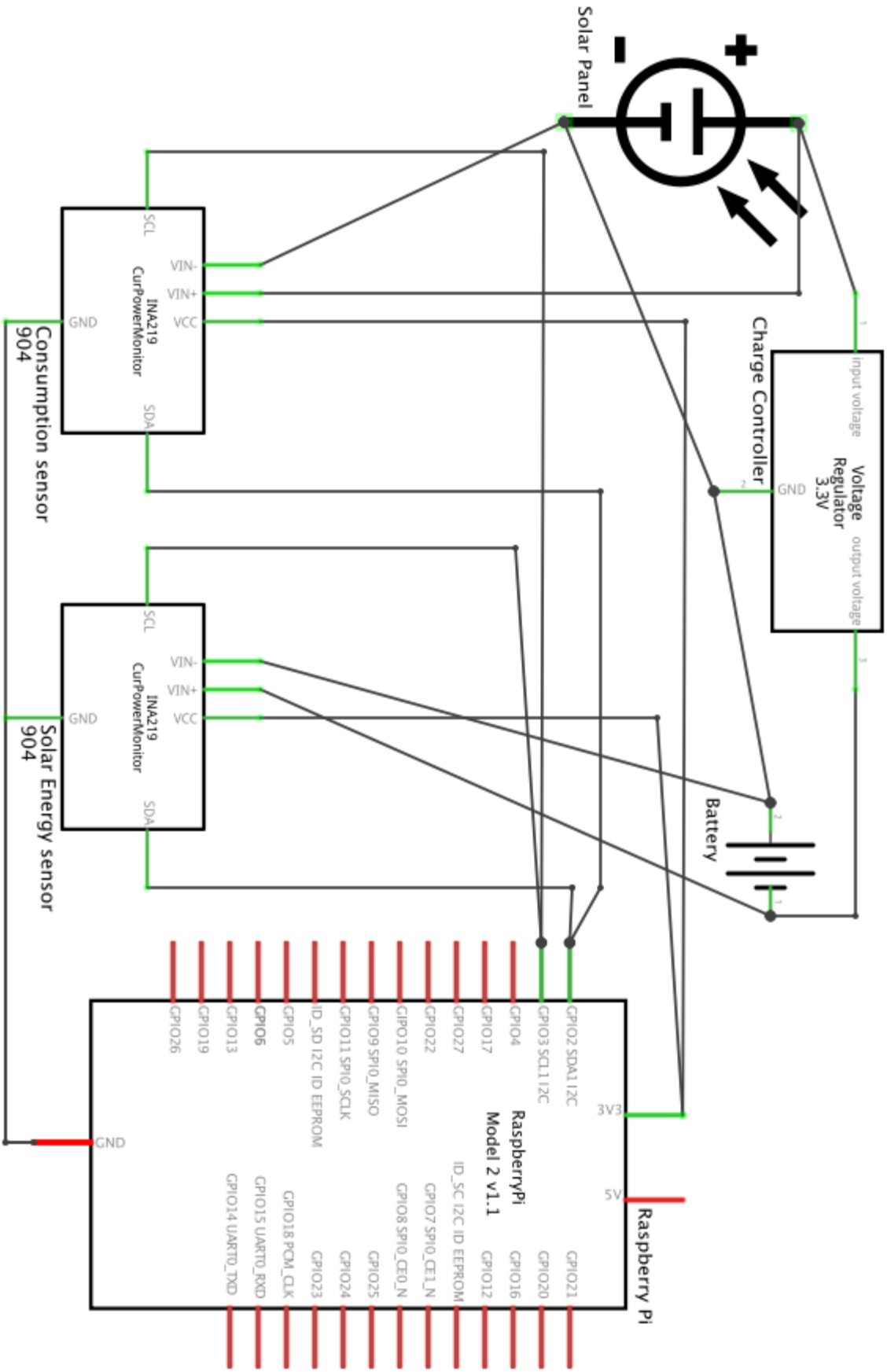


Figure 7: PCB Schematic design of household

5.3 Software Implementation

All software is developed to work on the Raspberry Pi. This way it is easier to implement the demo on a bigger scale by copying the image file to another Raspberry Pi and configure a few unique settings. The interaction between the software and the user is shown schematically in figure 9.

5.3.1 Rasbian

The web server, database and blockchain are deployed on Raspbian Stretch. This is a Linux based operating system developed for the Raspberry Pi. Raspbian was chosen because it is free, open-source and a lot of libraries are available.

5.3.2 Database

The database is an MySQL database running locally on the Raspberry Pi. To communicate with the database through python the *cursor* class is used. To improve flexibility PHPmyadmin is installed to enable the interaction with the databases from a graphical user interface.

5.3.3 Dashboard

The household dashboard is developed in PHP, HTML5 and CSS3 and built on the open-source Bootstrap Framework. AJAX is used to automatically update the current flows without the necessity to reload the dashboard. The dashboard gives a household the possibility to look into their electricity usage and change the amount of money they are willing to pay for there renewable electricity. The chosen rate is saved on the blockchain.

The dashboard is separated in three parts. An overview for users to see the current electricity consumed and generated and the tokens they have in there wallet. The next part of the dashboard enables users to see the blocks of the blockchain and this way enable transparency of the transactions. The settings part enables users to change their rates and general user settings like password and username. The dashboard is shown in figure 9 and 10.

5.3.4 Smart Meter

The *Smart Meter* software is developed using *Python3* and the open-source *pi-ina219* library developed by Chris Borril[3]. The current sensor uses the I2C communication protocol to communicate with the Pi. This enables the sensor to return digital values for the voltage, current and power. An algorithm is created to measure the two current flows each 100ms. Variance and errors are calculated and checked before the measurements are uploaded to the local database and communicated with the blockchain.

Every 5000ms the software interacts with the blockchain by using the open-source web3.js library. The web3.js library makes it possible to interact with our blockchain by using JavaScript-like commands. When necessary, a transaction is made. This is done by calling a function which interacts with the blockchain through the smart Contract. The Python code can be found in listing 1.

First we import the necessary libraries to read the values of the sensors and make a connection to the database. Then we define the two sensors as instances of the *INA219* class where we give the slave address as an parameter. The sensors are configured to measure in a range of 16 volts. Then the *smartMeterID* is hard-coded and a connection to the database is made. A function is created to read the measurements and upload them to the database. Feedback is printed to the terminal to inform the developer. Every second the function is called through the while loop.

```
#!/usr/bin/env python

from ina219 import INA219, DeviceRangeError
from time import sleep
import mysql.connector

SHUNT_OHMS = 0.1
MAX_EXPECTED_AMPS = 2.0
ina1 = INA219(SHUNT_OHMS, MAX_EXPECTED_AMPS, address=int('0x40',16))
#ina1 measures the generated electricity
ina2 = INA219(SHUNT_OHMS, MAX_EXPECTED_AMPS, address=int('0x41',16))
#ina2 measures the consumed electricity
ina1.configure(ina1.RANGE_16V)
ina2.configure(ina2.RANGE_16V)

#smartMeterID for each household
smartMeterID = "test1234"

cnx = mysql.connector.connect(user='root', password='raspberry',
                              host='localhost',
                              database='MicroGrid')
cursor = cnx.cursor()

def read_sensors():
  try:
    generated = str(round((ina1.voltage()*ina1.current()),2))
    consumption = str(round((ina2.voltage()*ina3.current()),2))
    update_measurement = "INSERT INTO SmartMeter(smartMeterID,
       consumption, generated) VALUES ('%s', '%s', '%s' )" % (
       smartMeterID, consumption, generated)
    cursor.execute(update_measurement)
    cnx.commit()
    print ("Measurements succesfully updated...")
  except:
    print ("something went wrong!")
  print (" ")
while 1:
    read_sensors()
    sleep(1)

cnx.close()
```

Listing 1: Smart Meter Python script to read and save sensor values

5.3.5 Blockchain

The blockchain implementation used for this demo is a private Ethereum blockchain. The blockchain is developed using the open-source Go Ethereum implementation *Geth*. As described in Chapter 3, each blockchain starts with a *Genesis Block*. This block is created through a *Genesis File*. This file contains all information about the implementation what is stored in the genesis block. The genesis file is created using *Puppeth*. Puppeth is a CLI wizard tool included in *Geth* which makes it easier to create and manipulate genesis files. By using Puppeth it is no longer necessary to write the genesis file manually. Through configuring Puppeth the *Proof-Of-Authority* consensus mechanism is configured for the implementation.

To enable different nodes to find each other a *bootnode* is used. A bootnode can best be compared with a gathering spot for nodes. Instead of all nodes looking for each other they 'gather' at the bootnode to communicate. The blockchain is deployed using *Truffle*. A tool that makes it possible to deploy and test a blockchain implementation locally. It also helps simplifying the compiling and deployment of smart contracts.

There are two ways to communicatie with a node on an Ethereum blockchain. Through Inter-Process Communication (IPC) and Remote Procedure Call (RPC). RPC makes it possible to communicate over the internet through HTPP requests whereas IPC only works locally. Because we want to communicate with the Blockchain through our dashboard the RPC-protocol is used. IPC and RPC are both built-in with *Geth*. To connect to our node (and indirectly to the blockchain) through IPC we can use:

```
1 geth attach node1/geth.ipc
```

The file *geth.ipc* exists while a node is running, in this case node1. As long as the node is running we can connect with it through the IPC file. For connecting with a node through RPC we can use:

```
1 geth attach 'http://192.168.2.3:8501'
```

Where the address is the IP-address of the node as well as the port where we can find the node. In this case the IP-address is *192.168.2.3* and the port where we can find the node is *8501*.

With geth we can run the node with different parameters. The parameters can be used to tell geth where to find the necessary files to boot up the node as well as where to find the bootnode and which privileges are allowed when communicating with the node. The code used for implementation of the demo can be found in listing 2. The code is explained briefly below the listing.

```
geth
--datadir node1/ --syncmode 'full' --port 30311 --rpc
--rpcaddr '192.168.2.3' --rpcport 8501
--rpcapi 'personal,db,eth,net,web3,txpool,miner'
--bootnodes 'enode://<account of bootnode>@192.168.2.3:30310'
--networkid 1515 --gasprice '1'
--unlock '34da6c488df38140ae4a111e428aceba80c99dc9'
--password node1/password.txt --mine
```

Listing 2: Bash script to start blockchain node

First we tell geth where to find the existing files of the node by using *–datadir*. And use *–syncmode 'full'* to synchronize the complete blockchain. The *–port* parameter is used to define the port on which the node can communicate with other nodes. *–rpc* tells geth to allow communication with the node through the RPC protocol. Using *–rpcaddr* we define the IP-address on which we can communicate with the node, this is combined with *–rpcport* to define the port to use. With *–rpcapi* we tell geth what libraries are allowed to be used when communicating with the node using RPC. In our example the *web3* library is important because it allows us to communicate with the node through a web server. *–bootnodes* is used to located the bootnode of our demo. The account, ip-address and port of the node are needed. *–gasprice* defines the transaction fee for each transaction. *–unlock* is used to unlock one of the accounts found in the earlier defined folder. Combined with the *–password* parameter the account can be unlocked. Through *–mine* the node is commanded to look for new transactions and create new blocks.

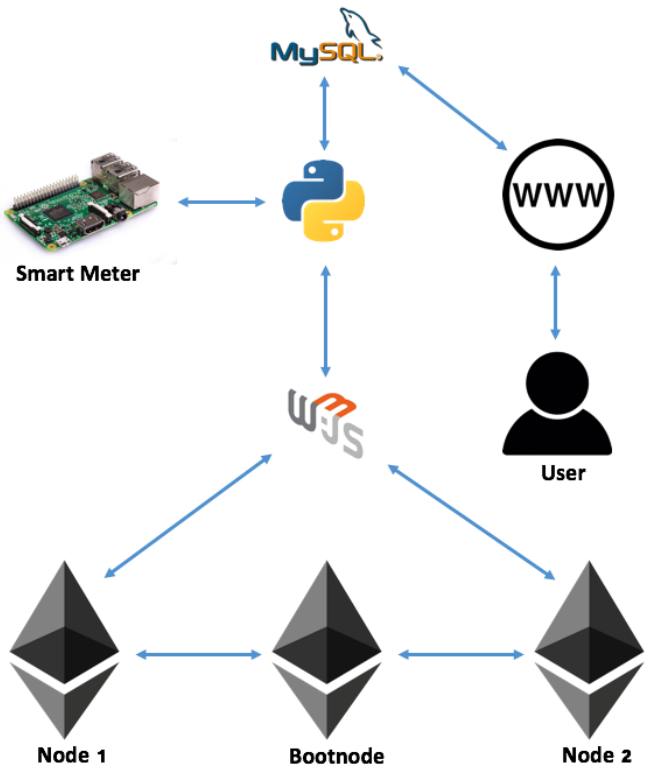


Figure 8: Simplified schematic of software implementation

The smart contracts for an Ethereum blockchain are written in *Solidity*. Which is an *contract-oriented* programming language based on Javascript. The smart contracts are compiled using *solc*, a tool installed together with Truffle. After compiling the contract is deployed on the blockchain, an action which costs ether. When succesfully deployed the network returns an address and an Application Binary Interface (ABI).

Both are necessary for other users to be able to locate the contract on the blockchain. The address let's the user know where to find the contract on the blockchain. The ABI says something about the structure of the contract and makes it possible to communicate between two program modules. It is used to encode contract call on the blockchain as well as to read data from transactions. The Solidity code for the smart contract is shown in listing 3 and is explained briefly below.

Every Solidity code starts with stating the used version. Then a *contract* is created, which can be compared to a class in an object-oriented programming language. The variables *kWh_rate*, *energyAccount*, *coinAccount* and *owner* are declared by giving their type and name. The *constructor* function works similar to other programming languages and is called when an instance of the class is created. The constructor assigns the *owner* variable to the *msg.sender* value, which is the value of the account calling the contract. The *modifier* keyword can create a restriction. In this case the *onlyOwner* modifier requires that the person communicating with the contract is the owner of the contract. The modifier can be called in a function to apply the create restriction. This is done in the *setRate()* function to ensure only the owner of the contract can assign a new value to the kWh_rate variable. The *sellEnergy()* and *buyEnergy()* functions are called through web3.js to make the automated transactions. The *getEnergyAccount()* and *getCoinAccount()* functions can be called to receive information about the accounts balances. For example for the dashboard overview.

```
pragma solidity ^0.4.7;

contract MicroChain {
    uint public kWh_rate = 15;
    mapping (address => uint) energyAccount;
    mapping (address => uint) coinAccount;
    address public owner;

    function MicroChain() {
        owner = msg.sender;
    }

    modifier onlyOwner {
         require(msg.sender != owner);
        _;
    }

    function setRate(uint rate) onlyOwner {
        kWh_rate = rate;
    }

    function sellEnergy(uint kwh) public {
        coinAccount[msg.sender] += (kwh * kWh_rate);
    }

    function buyEnergy(uint coin) {
        if (coinAccount[msg.sender] > coin) {
            coinAccount[msg.sender] -= coin;
            energyAccount[msg.sender] += (coin / kWh_rate);
        }
    }

    function getEnergyAccount() returns (uint kwh) {
        return energyAccount[msg.sender];
    }

    function getCoinAccount() returns (uint coin) {
        return coinAccount[msg.sender];
    }
}
```

Listing 3: Smart Contract Solidity script

6 Results

In this chapter the implementation results are shown. The physical implementation is shown by several pictures showing the implementation of the Raspberry Pi with the INA219 sensors as well as the solar panel and battery. The software implementation is shown by screen shots of the user dashboard. The underlying code is already discussed in Chapter 4.

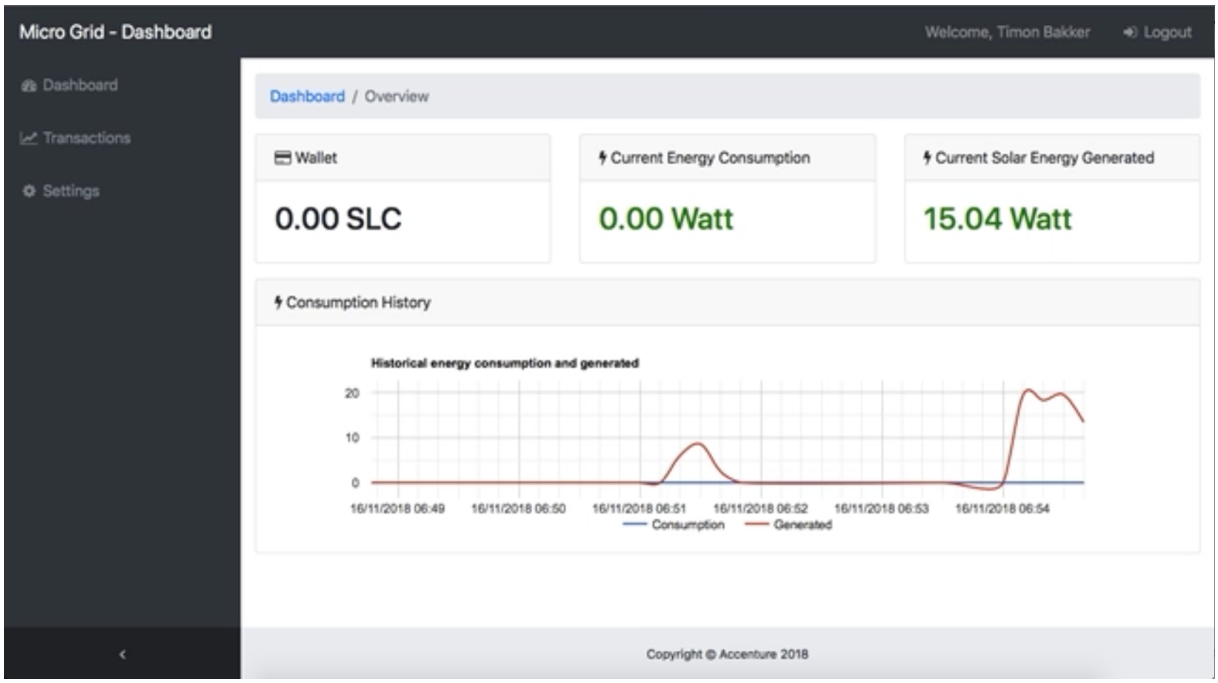


Figure 9: Dashboard: overview page

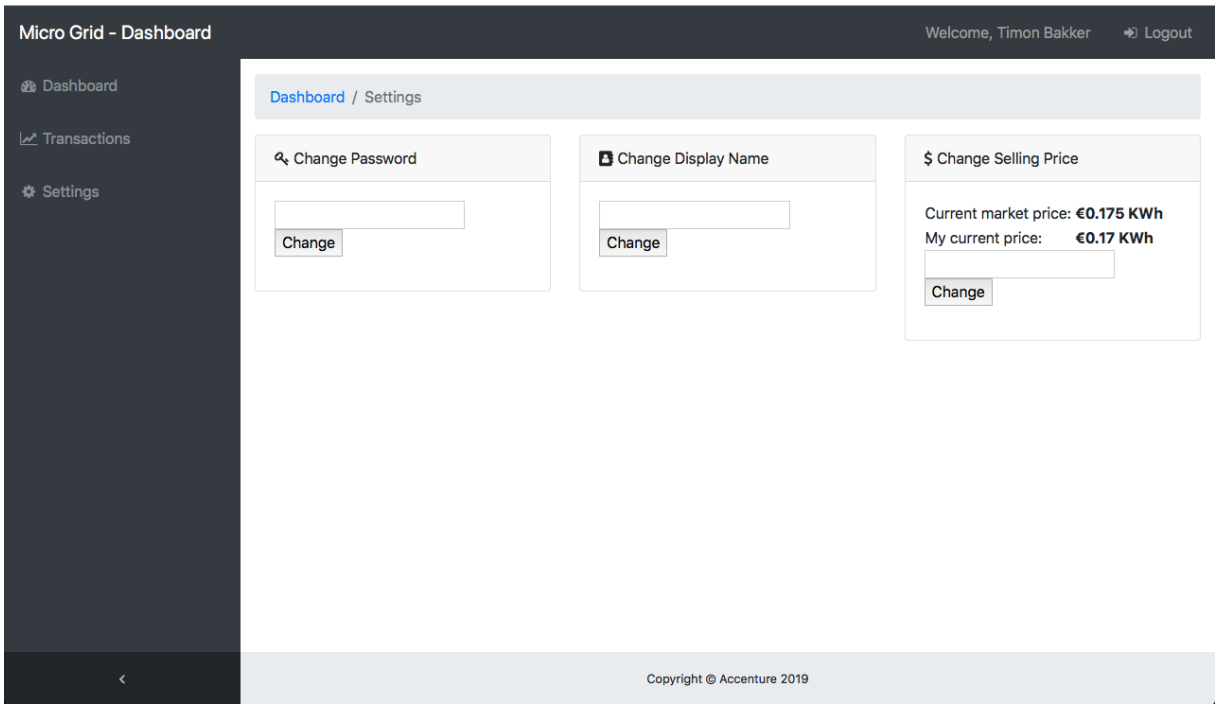


Figure 10: Dashboard: settings page

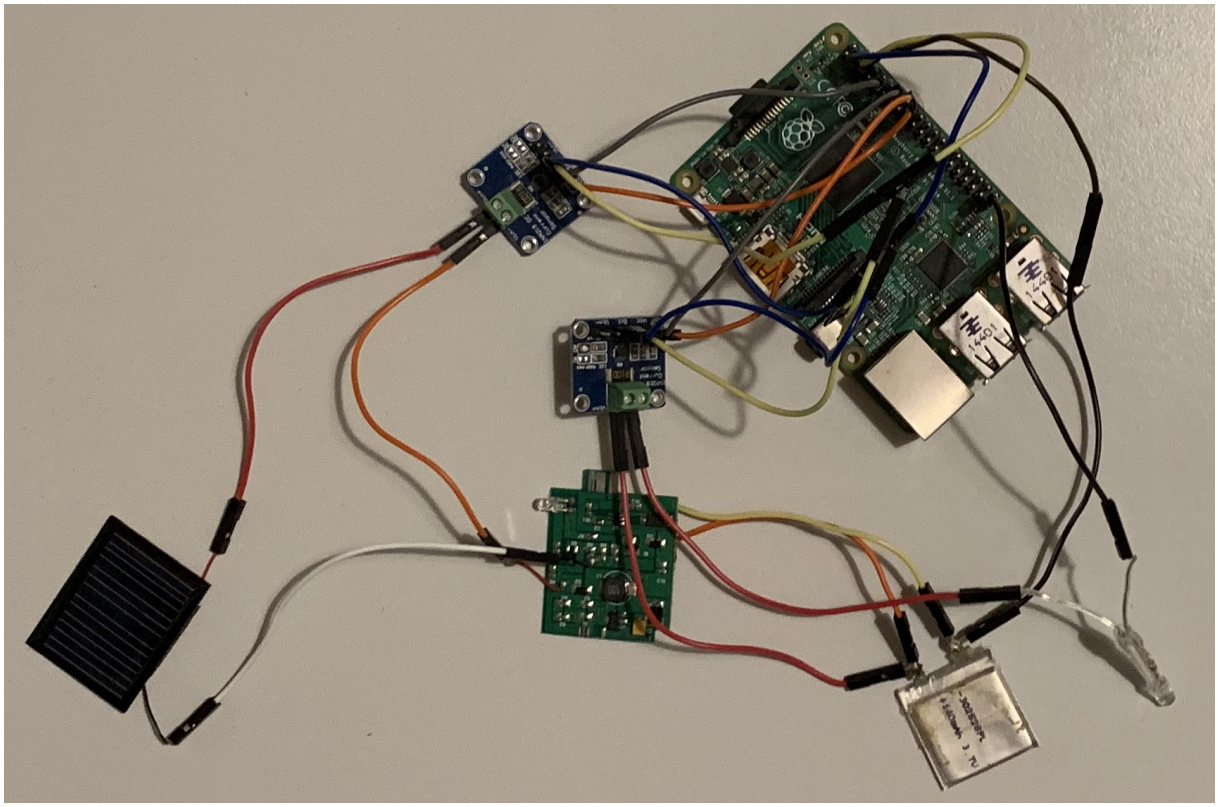
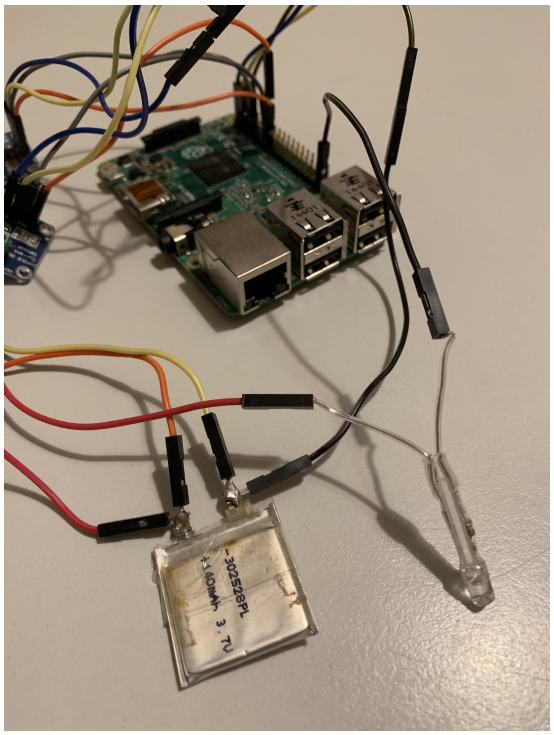
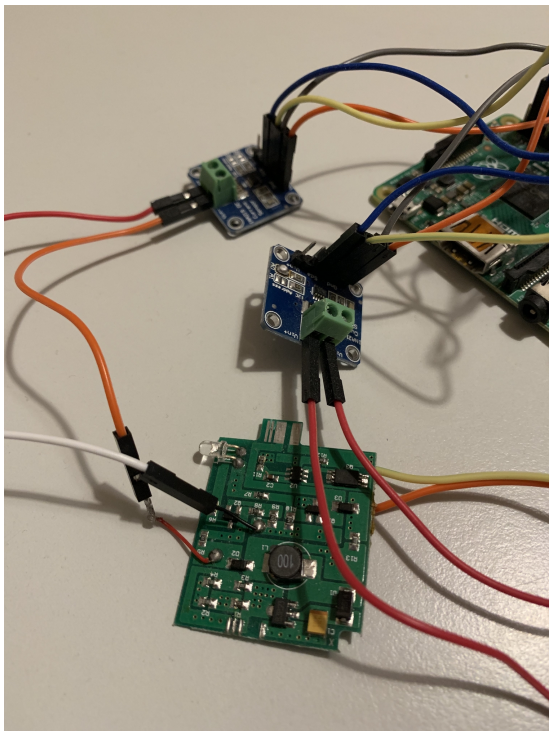


Figure 11: Physical implementation: overview



(a) Battery and LED



(b) Controller and Sensors

Figure 12: Physical implementation: different parts highlighted

7 Conclusion

In this chapter we look back on the project and discuss on the research. We look at the advantages and disadvantages of the use of blockchain for the implementation of the proposed solution. And look at the options for future work.

In this project we implemented a solution for the problem where households where unable to sell and buy private generated renewable energy. And are forced to sell there excess renewable energy to an utility company for a lower selling price than the buying price. By implementation a small scale demo using a Raspberry Pi, current sensors and the blockchain technology we have shown that the problem can be efficiently solved using the blockchain technology. Through the decentralized property of the blockchain the demo households are able to securely trade renewable energy and determine the selling and buying price without intervention of an utility company.

The research goal was to research if it was possible to efficiently implement the trading of solar panel electricity by using blockchain technology. The implemented demo shows that it is possible to efficiently implement a solution. However, the small scale implementation makes the blockchain vulnerable and the registration of new households requires some kind of centralized party to organize the implementation. A requirement that undermines the decentralized property of the blockchain.

We should realize that the blockchain technology has possibilities to make a difference in the future of digitization, but it's not the solution for every problem. Especially on small scale we should question the added value of using the blockchain technology instead of *tradition technologies*. And be aware that we choose the best option instead of the today more popular ones. The average household owner might not be interested in complicated blockchain technologies and just wants renewable, and preferable cheap, energy. The small scale and necessity of a middle man to add households tackles the decentralized concept and power of the Blockchain technology and makes it vulnerable for social hacking. By implementing an easy-to-use platform to create an account, register your house and sell or buy energy it would be a realistic solution for the future. And by the, already large, implementation of smart meters in The Netherlands scalability should be easier then men would expect.

Future work could focus on the implementation of the solution on larger scale an by using the already existing smart meters. Through the use of already existing smart meters one would be able to research a similar implementation on much larger scale to create a more secure decentralized trading platform. Another possible subject for future research could be the implementation of the proposed solution using a centralized party. Because the issue in the research problem is not a trust issue. It's the non-existent of a free energy market.

References

[1] Dave Bayer, Stuart Haber, and W. Scott Stornetta. "Improving the Efficiency and Reliability of Digital Time-Stamping". In: *Sequences II*. Ed. by Renato Capocelli, Alfredo De Santis, and Ugo Vaccaro. New York, NY: Springer New York, 1993, pp. 329–334. ISBN: 978-1-4613-9323-8.

[2] Usman W. Chohan. "The Double Spending Problem and Cryptocurrencies". In: (Dec. 2017). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3090174.

[3] chrisb2. *pi_ina219*. 2018. URL: https://github.com/chrisb2/pi_ina219 (visited on 01/25/2019).

[4] CoinMarketCap. *Cryptocurrency Market Capitalizations*. URL: https://coinmarketcap.com (visited on 01/25/2019).

[5] Groene Courant. *Zonder salderen leveren zonnepanelen financieel niks meer op*. 2016. URL: https://groenecourant.nl/zonne-energie/zonder-salderen-leveren-zonnepanelen-financieel-niks-meer-op/ (visited on 01/25/2019).

[6] GO Ethereum. *Official Go implementation of the Ethereum protocol*. URL: https://geth.ethereum.org (visited on 01/25/2019).

[7] Stuart Haber and W. Scott Stornetta. "How to time-stamp a digital document". In: *Journal of Cryptology* 3.2 (Jan. 1991), pp. 99–111. ISSN: 1432-1378. DOI: 10.1007/BF00196791. URL: https://doi.org/10.1007/BF00196791.

[8] Paveen Jayachdran. *The difference between public and private blockchain*. 2017. URL: https://www.ibm.com/blogs/blockchain/2017/05/the-difference-between-public-and-private-blockchain/ (visited on 01/25/2019).

[9] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *Advances in Cryptology — CRYPTO '87*. Ed. by Carl Pomerance. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378. ISBN: 978-3-540-48184-3.

[10] Brooklyn MicroGrid. *The future of energy is local*. URL: http://brooklynmicrogrid.com (visited on 01/25/2019).

[11] MpptSolar. *How to Connect Two or More Solar Panels in Parallel*. URL: https://www.mpptsolar.com/en/solar-panels-in-parallel.html (visited on 01/25/2019).

[12] Netbeheer Nederland. *Slimme Meter*. URL: https://www.netbeheernederland.nl/consumenteninformatie/slimmemeter (visited on 01/25/2019).

[13] Nuon. *Salderen en teruglevering energie*. URL: https://www.nuon.nl/producten/zonnepanelen/salderen/ (visited on 01/25/2019).

[14] Pi4J Project. *Pin Numbering - Raspberry Pi 2 Model B*. URL: http://pi4j.com/pins/model-2b-rev1.html (visited on 01/25/2019).

[15] James Ray. *A Next-Generation Smart Contract and Decentralized Application Platform*. 2018. URL: https://github.com/ethereum/wiki/wiki/White-Paper (visited on 01/25/2019).

[16] Trent Rhode. *Blockchain Consensus Mechanisms fffdfffdfffd Proof of Work vs Proof of Stake and More*. 2018. URL: https://unhashed.com/cryptocurrency-coin-guides/blockchain-consensus-mechanisms/ (visited on 01/25/2019).

[17] Rototron. *Raspberry Pi INA219 Tutorial*. 2017. URL: https://www.rototron.info/raspberry-pi-ina219-tutorial/ (visited on 01/25/2019).

[18] Bryan Wann. *Solar/battery/load power logging with Raspberry Pi and INA219*. 2014. URL: https://binaryfury.wann.net/2014/04/solarbatteryload-power-logging-with-raspberry-pi-and-ina219/ (visited on 01/25/2019).

[19] Wikipedia. *One-way function*. URL: https://en.wikipedia.org/wiki/One-way_function (visited on 01/15/2019).

[20] Gavin Wood. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151 (2014), pp. 1–32.