University of Kaiserslautern

Department of Electrical Engineering and Information Technology

Microelectronic Systems Design Research Group

Bachelor Thesis

Design and Implementation of a Blockchain-Based Smart Outlet Concept

Entwurf und Implementierung eines Blockchain-basierten Smart Outlets Konzept

---

Presented:	May 16, 2019
Author:	Daniel Gretzke (392488)
Research Group Chief:	Prof. Dr.-Ing. N. Wehn
Tutor:	M.Sc. Frederik Lauer

Statement

I declare that this thesis was written solely by myself and exclusively with help of the cited resources.

Kaiserslautern, May 16, 2019

Daniel Gretzke

Abstract

Abstract English here.

Zusammenfassung

Zusammenfassung Deutsch hier.

Contents

1 Introduction

> Whereas most technologies tend to automate workers on the periphery doing menial tasks, blockchains automate away the center. Instead of putting the taxi driver out of a job, blockchain puts Uber out of a job and lets the taxi drivers work with the customer directly.
> — *Vitalik Buterin, co-founder of Ethereum*

The blockchain was first implemented in 2009 by Satoshi Nakamoto (pseudonym) and was called Bitcoin. Since then, it has steadily gained importance every year. Meanwhile thousands of cryptocurrencies and tokens were built on this technology. The hype in the year 2017 called the attention of many companies to blockchain and even last year, when prices fell as far as 95%, the interest in this field didn't drop.

Compared to traditional payment methods like Visa, Banks and PayPal, cryptocurrencies are built decentralized, meaning that there is no central organization that controls transactions, the issuance of new money, et cetera. The validity of the blockchain *P2P* (peer to peer) network is secured through cryptographic protocols. This brings some benefits. Usually traditional payment methods go with high transactions costs, most commonly in the amount of a few percent. On the contrary the cost of a single transaction on a blockchain averages out at just a few cents[1]. Some cryptocurrencies even work without any fees.

Because of this they are suited for micro transactions really well. There are some disadvantages through. The blockchain technology is still at an early stage and really immature. Compared to traditional electronic payments it only manages to achieve very few *TPS* (transactions per second) and has long transaction times. E.g. Bitcoin manages 4-5 TPS[2] as opposed to Visa, which manages to process almost 4000 TPS on average[3].

But, coming back to the quote from Vitalik Buterin, the key strength of blockchain and cryptocurrencies is the decentralization aspect. For many, it will reshape various markets we know today, potentially revolutionize the financial industry and even disrupt monopolies in the future.

Another future trend is the electric car. It's expected that in a few years most cars on the road and almost all cars sold will be electric. Often these need to be charged overnight. Unfortunately, most city residents are familiar with the problem that they rarely park in front of their own house let alone own a garage. It's foreseeable that recharging your car might bring difficulties.

This bachelors thesis devotes itself to this problem. It examines whether a smart electric socket, which is placed outside the house by a homeowner, can be used to

efficiently sell electricity and which payment method is suited best for this task. Afterwards a prototype is to be developed that implements the previously worked out concept. Additionally it will serve as an example on how to implement *M2M* (machine to machine) payments on a micro controller level.

2 Theory

The purpose of this chapter is to explain the most important terms about blockchain and the key underlying components of the technology. Most explanations will be kept superficial, as the technological implementation is too complex to deal with it in a few sentences.

Hash

The cryptographic hash function is a mathematical function that usually fulfills the following requirements[4]:

1. determinism: the same input always produces the same output
2. preimage resistance: it's nearly impossible to get the input from the output
3. second-preimage resistance: it's nearly impossible to find a second input that produces the same output as the first input

The output of this function is called hash. Bitcoin uses the SHA-256[5], Ethereum uses the Keccak-256[6] hashing algorithm which produces an output hash with a length of 32 bytes.

Public Key Cryptography

Public key cryptography[7] is the backbone of the blockchain technology and consists of a key pair – a private and a public key. As the name suggests the private key needs to be secret and the public key can be shared with anyone. A message can be signed using the private key resulting in a signature. This signature can then be verified using the public key to prove that the message was, in fact, signed by the corresponding private key. Bitcoin and Ethereum use the *ECDSA* (Elliptic Curve Digital Signature Algorithm) for signatures. They are used to prove that transactions were really sent by a specific account and not manipulated.

Wallet

Everything that's needed to send and/or receive cryptocurrencies is a private key[5], meaning a private key is a wallet in it's simplest form. More complex forms of a wallet encrypt a private private key with a password or store it on a dedicated hardware device.

Address

An address is used to identify participants on the blockchain and usually is the public key belonging to the private key of the account or is generated fromm the public key, e.g.

Ethereum uses the right most 20 bytes of the hash of the public key as the address[6].

Ledger
The equivalent of a traditional Record of all addresses and their balances. Every participant of the network has a copy of this Ledger.

Node
A node is a participant on the blockchain. When a new node joins the network, it downloads the ledger, currently accepted by the majority. A so-colled full node has the entire history of the ledger – from the first block, to the latest.

Transaction
A transaction updates the Ledger, e.g. reducing the balance of an address by a specific amount and adding it to the balance of another account.

Blockchain
A list of transactions is bundled together into a block. Also included in this block is the hash of the previous block, which contains the hash of the previous block, and so on and creates a chain of blocks[5] which always point to their predecessor, thus the name blockchain. This is also the reason why a blockchain is considered immutable: modifying any information in a block would automatically change the hashes of every block after it and would be rejected by the network.

Miner / Validator
A participant of the network whose task it is to secure the network and validate / mine new transactions and blocks.

Proof of Work
There are different approaches to reach consensus over which block should be added next to the network, PoW (Proof of Work) is one of them and is used by most cryptocurrencies. A *nonce* (number used once) is included in the block which is adjusted by a validator. Every different nonce produces a different block hash. The goal is to find a nonce that is numerically lower than a certain threshold value (also called difficulty). A validator who finds this number can submit their block to the network. All miners race to find the nonce as fast as possible, so their block will be accepted by all other

participants in the network and considered valid. The fastest submission gets a so-colled block reward, a reward minted by the protocol to the fastest miner and all transaction fees from that block.

Block Time

The time it takes to add a new block to the network is called block time. On the Bitcoin network the block time is 10 minutes[5], Ethereum wants to archive a block time of 12 seconds[8], realistically its averaging at about 15 seconds[9]. The difficulty is adjusted automatically to keep the block time roughly the same, no matter how much computing power is currently mining.

Confirmation Time

The time it takes until a transaction is mined. The higher the transaction fee the faster transaction is confirmed.

Ethereum Virtual Machine

The *EVM* (Ethereum Virtual Machine) is a turing complete virtual machine that can execute computer code on the Ethereum Blockchain. As a result the Ethereum network acts like a decentralized computer with all the features of a blockchain: the storage is entirely public and every computation or code execution is recorded on the blockchain.

Smart Contract

Smart contracts are written in special programming languages, the most popular being called "Solidity", and are compiled into bytecode afterwards. This bytecode is then publicly stored on the blockchain and can be executed by everyone. The advantage of smart contracts is, that the code is public to everyone and immutable, thus can theoretically be used as a binding, programmable contract that can control monetary value on its own. As a drawback every computational step and byte stored on the blockchain costs money in form of transaction cost.

DApp / Web3

DApp stands for decentralized application. Web3 is also called the decentralized web, where websites are powered by smart contracts and DApps.

Account

There are two types of accounts on the Ethereum Blockchain: An *EOA* (externally owned account) and a contract account. Each of those accounts has a transaction count and a balance[6] associated with them. An EOA is controlled by a private key, whereas a contract account has code and storage associated with it.

Off-chain Transactions / Payment Channels

For every transaction on the blockchain fees have to be paid. One way to save transaction costs is to have multiple transactions that are not recorded on the blockchain, so-colled off-chain transactions, and settle them on the blockchain, once the payment process is finished.

Ether - Gwei - Wei

Ether is the currency used on the Ethereum network. It can be divided into smaller fractions. The most important are Wei and GWei. Wei is the smallest unit: 1 Ether equals 10^{18} Wei[6]. Gwei is primarily used for calculations of transaction costs: 1 Ether equals 10^9 Gwei, 1 Gwei equals 10^9 Wei. Ether is always used in Wei format in Smart Contracts.

Mainnet / Testnet

The main network of a blockchain is called the Mainnet. Testnets are blockchain networks that behave the same way as the Mainnet. This allows developers to develop smart contracts without the risk of losing any monetary value. Additionaly, improvements to the Mainnet are often tested on a Testnet prior to their official implementation.

3 Analysis of different payment methods

To revisit the concept from the introduction, the prerequisites have to be specified first:

1. A potential customer Alice, who owns an electric car and wants to charge it overnight, wants to purchase electricity.
2. A supplier Bob, who owns a smart electrical socket, wants to sell electricity.
3. Alice and Bob do not meet, Alice does not trust Bob to supply the electricity she paid for, Bob does not trust Alice to not wrongfully revert the payment after the electricity has been supplied. It has to be assumed that there are bad actors, who want to steal from the other party (in the form of monetary value or electricity).
4. For all calculations throughout this thesis the following values are assumed:
   a) Charging power: 3.7 kW
   b) Electricity price: 0.3 €/kWh
   c) Total transferred energy: 40 kWh
   d) Charging duration: 11 hours
   e) Electricity cost: 12€
   f) Price for the customer: 18€ (profit margin of 50%)

The objective of this chapter is to evaluate which payment method is suited best for the described use case. Next, the goals of the M2M payment system have to be defined:

1. Value has to be transferred from Alice to Bob.
2. Electricity has to be supplied in return for a payment, whereby the risk or impact of not getting electricity in return has to be minimal.
3. Alice needs to be able to stop paying for electricity when it's not longer needed, e.g. when the battery is fully charged, without overpaying.
4. Bob should only start supplying electricity after a valid payment was received and thus the risk of losing electricity is minimized.
5. Value should be transferred from Alice to Bob at least once per minute, to continuously pay for electricity to reduce risk, but transaction costs should stay at a minimum at the same time. For this thesis a payment interval of 15 seconds is assumed resulting in 0.007€ per payment.

Now that the requirements are defined it can be discussed, which payment method should be implemented on the prototype. In the following paragraph the advantages and disadvantages of traditional payment methods, established electronic and online payments and cryptocurrencies are compared.

The simplest form of monetary value is cash, it could be imagined that money could be paid through a coin slot and electricity would be returned as a result. Unfortunately it's not suitable for this use case, as the risk, of not receiving any electricity in return for the customer and the risk of theft for the supplier, e.g. someone violently stealing the cash, is too high.

Traditional electronic or online payments like VISA or PayPal pose a low risk of theft for the buyer because of customer protection methods, unfortunately the risk for the seller is non-negligible, e.g. in the form of credit card fraud or customer protection exploitation. Another disadvantage are high fees, PayPal, for example, has a pricing of 0.10€ + 10% for micro-transactions[10] and wouldn't be economically feasible for the predefined goals. The key strength are fast, near-instant transaction times.

Cryptocurrency payments have some major advantages over the previously analyzed payment methods. Compared to electronic and online payments, fees are low because they are fixed, not variable. Some cryptocurrencies even work without any fees at all. Payments can be broken down into fractions, as mentioned above with payment intervals of a minute or less. Because all payments are immutable the supplier has no risk, the customer risks just losing a less than one cent for a first payment. One of the biggest disadvantages of cryptocurrencies for M2M payments are long transaction times, but as the technology is still in its infancy, this might change in the future.

Over a thousand different cryptocurrencies exist and every one of them has their own rules that can drastically differ from the next one. Each underlying blockchain technology has their benefits and drawbacks. In the next paragraph some of these digital currencies will be evaluated whether or not they are suitable to reach the set goals.

Because Bitcoin is the earliest cryptocurrency, it suffers from problems other cryptocurrencies could improve upon. As demonstrated in the chart on the next page, scalability is one of these issues. The block time is 10 minutes[5] and during peak times the average confirmation time can take more than two hours. Thus bitcoin should not be considered for this use case.

Ethereum has a block time of 15 seconds. At the time of writing the fee to be included in the next block is 0.01€[1]. In this case the transaction would be fast enough to meet the goal of 4 transactions per minute, but the transaction cost would exceed the payment, raising the total price as much as 143%.

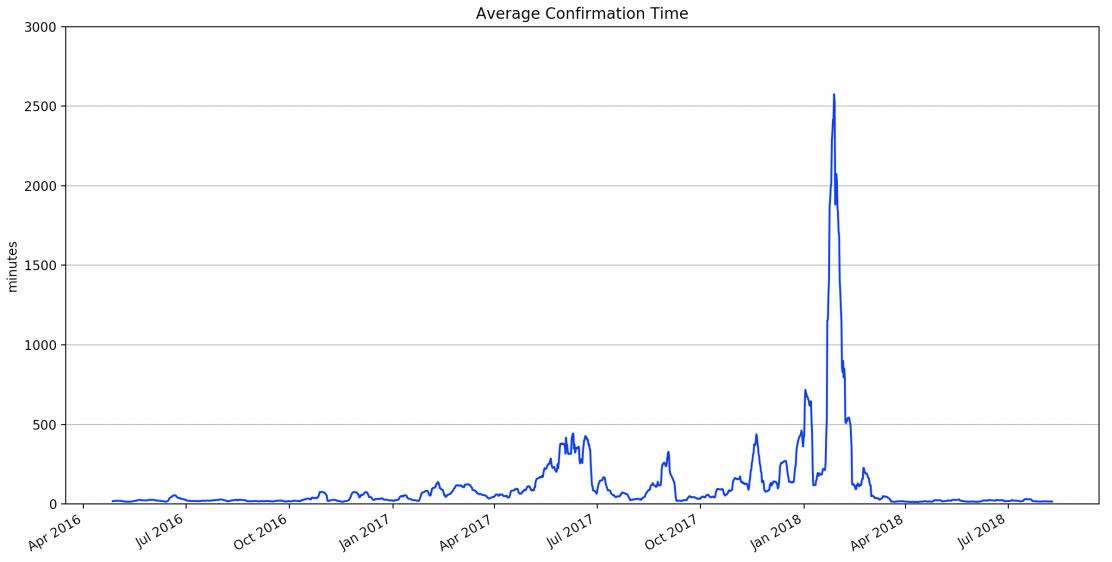

Figure 3.1: Average confirmation time of a transaction on the Bitcoin blockchain[11]

As mentioned above there are some cryptocurrencies that work without any fees that are worth to be considered. One of these currencies is IOTA. It was designed especially with M2M payments in mind and the underlying technology, called Tangle, differs from traditional blockchains. The way it works is that before someones transaction can be validated, they need to validate other transactions first[12]. In theory, this makes the network scale especially well — the more transactions are broadcasted, the shorter the transaction times get. In practice, transaction times are at 2.6 minutes at the time of writing[13].

Another feeless cryptocurrency is called Nano, formerly known as RaiBlocks. It's built upon a technology called block-lettuce. Each account on the network is an independent blockchain which can update itself asynchronously from the rest of the accounts. This makes nano not only have zero transaction costs but also allows it to have transaction times of less than a second. It can also handle way more TPS than Bitcoin and Ethereum, namely around 100 TPS over a longer period of time with peaks up to 300 TPS[14].
Unfortunately, at the time of development of the prototype, the Nano network faced some issues which resulted in transaction times in up to 20 seconds. After the V18 update these problems were solved and the transaction times returned back to less than a second[15].

Although increased transaction times did play a role, the main reason Nano was not chosen for the implementation is that it wasn't built with M2M transactions in mind. To prevent spamming, i.e. attacking the network through congesting it with transactions, PoW needs to be generated in order to send or receive transactions. According to the

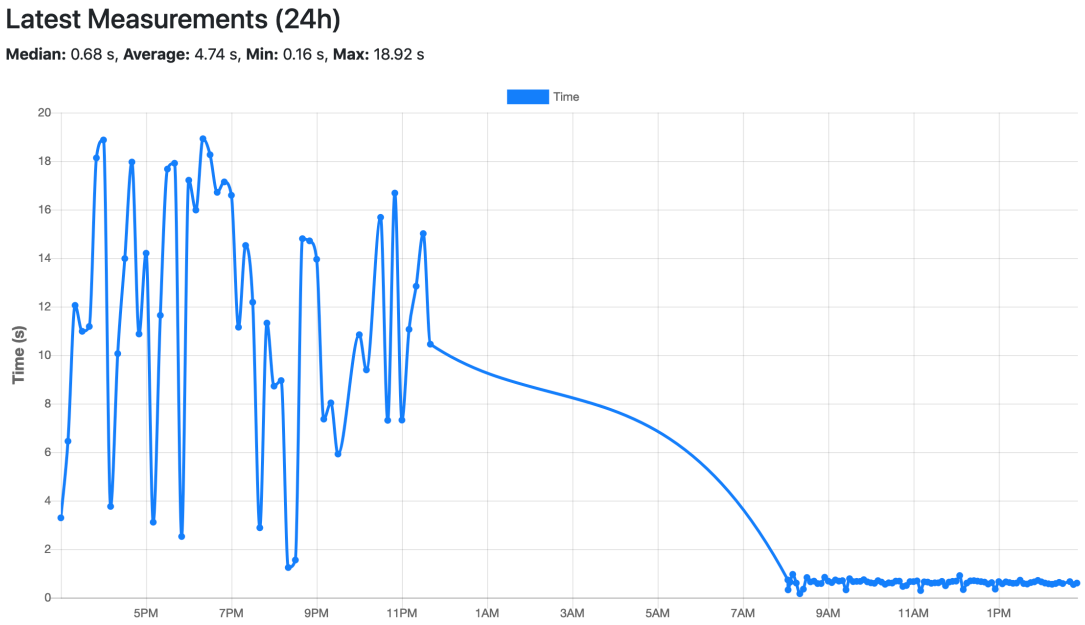

Figure 3.2: Improvements of transaction times after the update to V18 on February 22nd, 2019

Nano white paper[16] an Intel Core i7-4790K processor with 4.00 GHz can handle up to 0.33 TPS. A microcontroller would not be able to generate the PoW required for the transactions in a reasonable amount of time. Instead a dedicated server would be required for each supplier and customer, which would store the private keys and handle the transmission and reception of transactions. The micro controllers would merely communicate with said servers.

One way to cut transaction costs is the use of off-chain transactions. These work very similarly to on-chain transactions: information like a beneficiary and value can be signed by a sender, so it can be proven where the transaction originates from. They differ from each other, as off-chain transactions, as the name suggests, are not recorded on the blockchain and therefore are feeless. A number of these off-chain transactions can be bundled in a payment channel. In the end, these transactions have to be settled on the blockchain, actually modifying the ledger and updating the balances of all participants. Thus the 2,460 transactions required during a charging period of 11 hours can be bundled into a few transactions on the blockchain, while still meeting all the requirements listed above.
Ethereum was chosen for the implementation of this prototype, as it is Turing complete, therefore a payment channel can be implemented using smart contracts. The following steps briefly explain how this concept works:

1. The customer places a deposit (max. purchase value) into the smart contract and initializes the payment channel.

2. The customer signs an off-chain transaction and sends it to the supplier.

3. Once the supplier receives the transaction, the delivery of electricity begins.

4. Steps 2 & 3 are repeated until the customer or the supplier want to discontinue the exchange.

5. As soon as any party wants to close the payment channel, the supplier submits the offline transactions to the smart contract. Each party is now able to withdraw their share from the smart contract.

With payment channels, the risk of the supplier is minimal, as electricity must only be provided when a valid payment was received and the customer only risks losing one transaction for the initialization of the payment channel and one off-chain transaction if no electricity is provided in return (with the values above this adds up to 0.017€).
In total, only 4 transactions have to be made, one for the payment channel initialization, one for the settlement, and one by each party to withdraw the balances totaling all transaction costs at a couple cents.

4 Concept, Setup & Implementation

4.1 Concept

To build a functioning prototype, which implements the payment channel described in the previous chapter the following components are required:

1. A supplier in form of a socket
2. A customer in form of a plug
3. A server to run a node that connects to the Ethereum blockchain
4. A Smart Contract on the Ethereum blockchain

All components will communicate with each other over a Wifi connection and TCP. The following section will summarize all requirements each component of the prototype has to meet. An in depth explanation of the setup and technical implementation will follow in the next section.

4.1.1 Socket

An AC electrical socket is required. A microcontroller, which can be placed between the electrical circuit and the socket, needs two additional components: a WiFi module to communicate with the web and the plug and a relay to switch the current on and off.

4.1.2 Plug

An AC electrical plug is required, e.g. a short extension cord, that can be connected to any electronic device. A hall effect-based current meter can be attached to the hot wire to measure the current. This current meter needs to be attached to a microcontroller which also has a WiFi module to communicate with the socket and make http requests to the server.

4.1.3 Server

The server will run an Ethereum node. This node has to be reachable from the outside, so the microcontrollers can send http requests to it.

4.1.4 Smart Contract

The Smart Contract acts as a trustee, managing the money during the exchange. It has to be programmed in a way that guarantees that no party can steal from the other.

4.2 Setup

This section will focus on the technical setup of the hardware components and the installation and setup of all required software.

4.2.1 Socket

The Sonoff S20 smart socket was used for the technical implementation of the concept, as it meets all requirements listed in the section above. A microcontroller is automatically powered by the socket it's plugged into. The S20 also has a WiFi module and a relay, which can be switched on and off by said microcontroller. Lastly the microcontroller can be reprogrammed via a serial port. To program the Sonoff S20, the screws, as seen in the image below, have to be unscrewed first, revealing the logic board with the relay and the serial port.

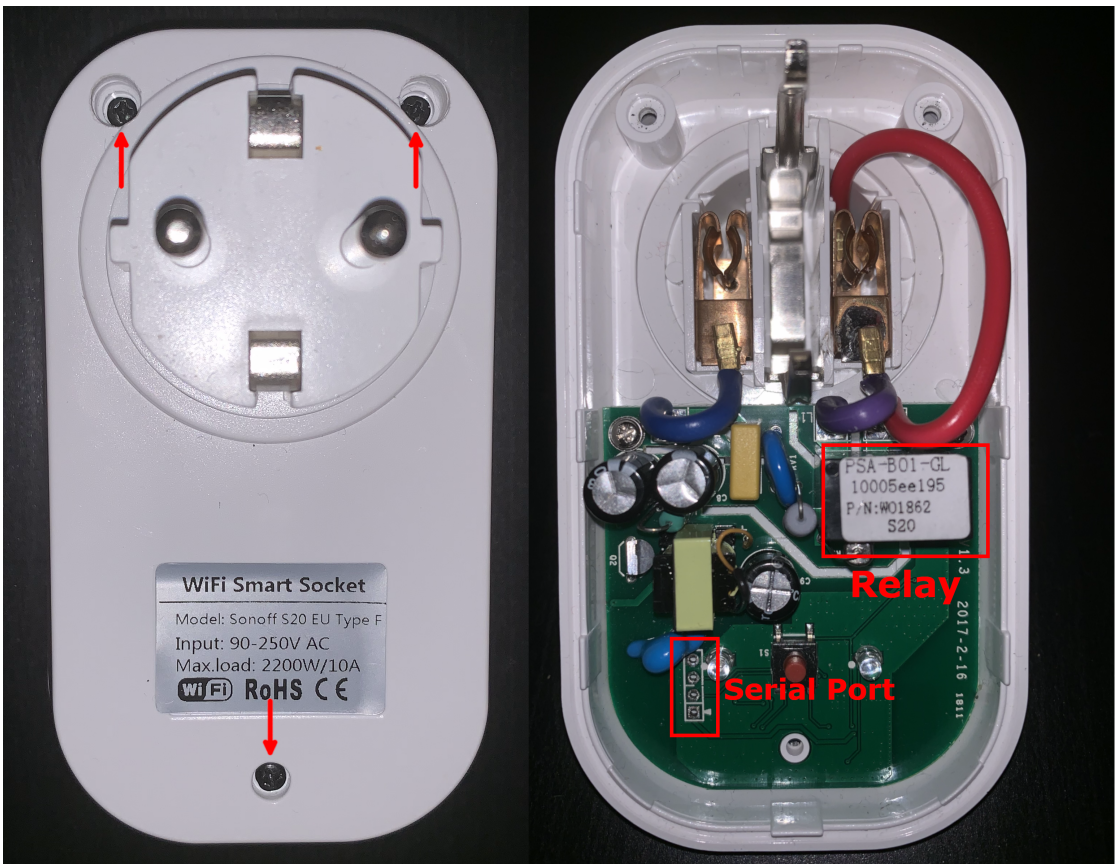

Figure 4.1: Sonoff S20

To program the microcontroller with a computer a FTDI USB to serial converter is required. The converter has to be plugged into the S20 as follows:

FTDI Converter	Sonoff S20
GND	GND
TX	RX
RX	TX
3.3V	3.3V

It's important to notice that the FTDI converter must operate at 3.3V entirely. Caution: some converters only switch the TX and RX pin to 3.3V while the VCC remains at 5V. This can fry the internals of the S20. To program the microcontroller, the button has to be pressed before plugging the pins into the serial port to put it in programming mode. After the pins have been inserted, the button can be released shortly after.

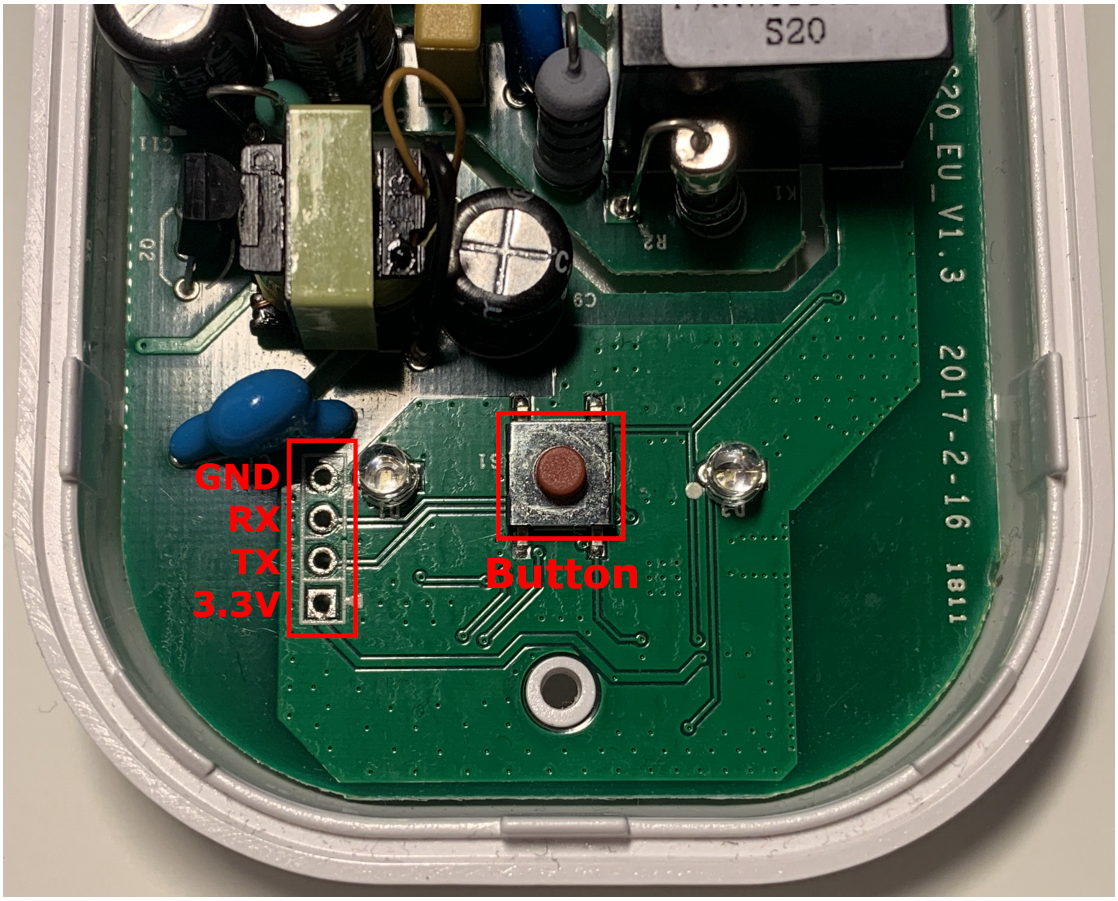

Figure 4.2: Serial ports of the Sonoff S20

Both, the socket and the plug, will be programmed using the Arduino IDE. The following steps need to be followed to install the ESP8266 Board, which the S20 is based on:

- Inside the Arduino IDE open "Preferences"
- Enter `http://arduino.esp8266.com/stable/package_esp8266com_index.json` under "Additional Boards Manager URLs"
- Open Tools → Board → Boards Manager
- Search and install "esp8266" by "ESP8266 Community"

After connecting the FTDI converter to the computer, it should appear under Tools → Port. To successfully flash code to the S20 the following settings have to be set:

- *Board*: "Generic ESP8266 Module"
- *CPU Frequency*: "80 MHz"
- *Flash Size*: "1M (no SPIFFS)"

4.2.2 Plug

The device to control the measurement of current in the plug and handle the communication with the socket is the Heltec WiFi Kit 8, which is based on an ESP8266 as well and has a 0.91 inch OLED display.

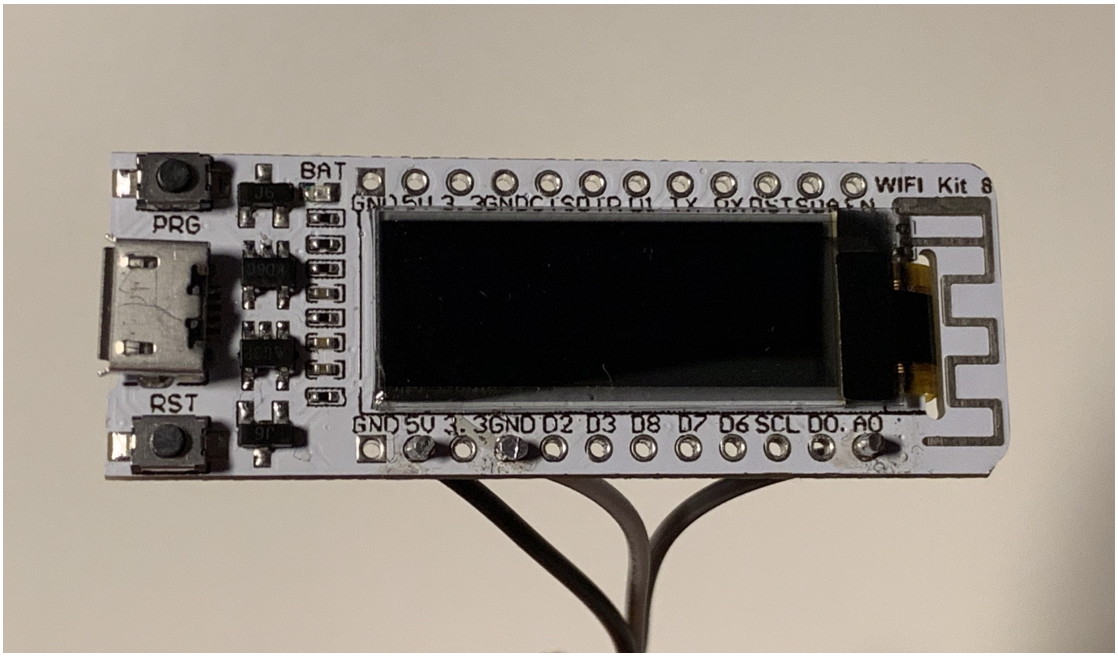

Figure 4.3: Heltec WiFi Kit 8

The ACS712 20A current meter is used to measure the current. It's hall effect-based and provides galvanic isolation up to a minimum of 2.1 kV (RMS)[17]. To connect the current meter, a part of the hot wire leading to the plug has to be cut and stripped. Both ends have to be inserted into the screw terminal of the ACS712. The current from the plug will now be redirected underneath the hall sensor.

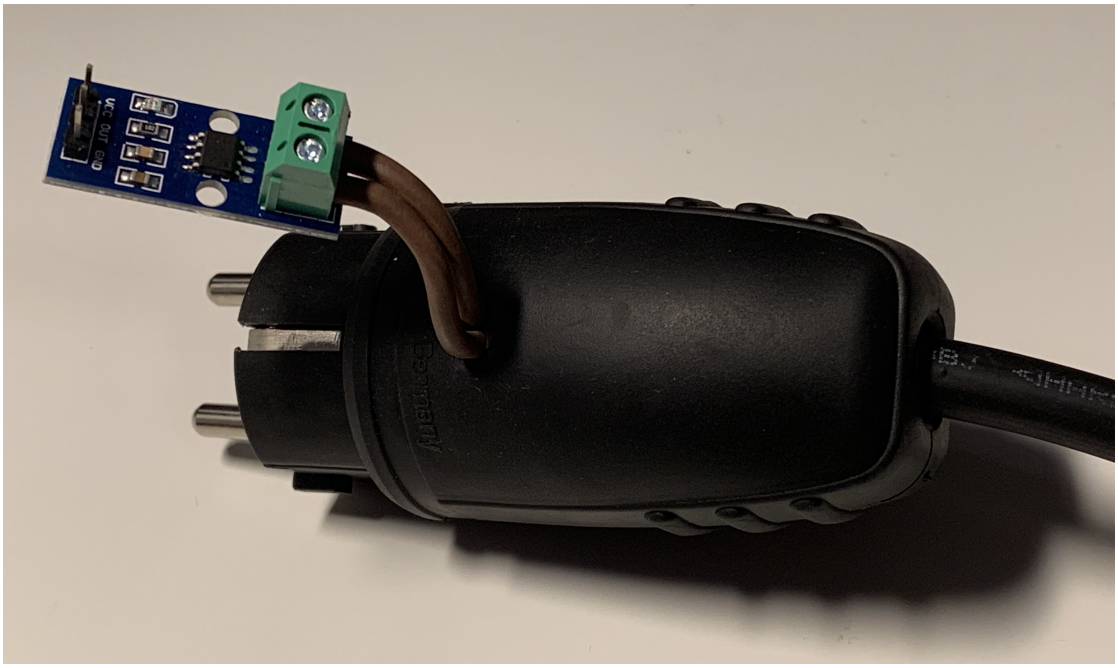

Figure 4.4: ACS712 current meter connected to plug

The pins of the current meter have to be soldered to the Heltec board as follows:

ACS712	Heltec WiFi Kit 8
VCC	5V
OUT	A0
GND	GND

To program the Heltec WiFi Kit 8, USB to UART drivers need to be installed first. The download link can be found under "References"[18]. Next, if the ESP8266 was not installed inside the Arduino IDE yet, the instructions found in the section above can be used to install the board.
After a successful driver installation, the board should be found under Tools → Port when the device is connected to the computer.

The following settings are required to ensure a successful flash of the Heltec WiFi Kit:

- *Board*: "NodeMCU 1.0 (ESP-12E Module)"
- *CPU Frequency*: "160 MHz"
- *Flash Size*: "4M (3M SPIFFS)"

4.2.3 Libraries

The prototype relies on some external libraries that have to be installed additionally:

Display
To use the display of the Heltec WiFi Kit a special library needs to be installed. Under Sketch → Include Library → Manage Library search for and install the "U8g2" library by "oliver".

WebSocket server
The communication between the plug and the socket relies on WebSockets. Download the library as a zip file from the git repository[19].
Install it via Sketch → Include Library → Add .ZIP Library.

ECDSA Library
Ethereum relies on the Elliptic Curve Digital Signature Algorithm, although there are some differences to the standard implementation of that algorithm, which will be explained later. The zip library is provided with the source code of this prototype and extends the "micro-ecc" library by Kenneth MacKay[20].

4.2.4 Server

A server is mandatory to act as a gateway to the Ethereum blockchain. Geth is the official "Golang implementation of the Ethereum protocol"[21] and is used to run a full node. After the installation it will be used to send transactions and make smart contract calls.

A *VPS* (Virtual Private Server) was used for this implementation running Ubuntu 18.04. The server has a six core CPU, 16 GB of RAM, 400GB SSD and 400 MBit/s unlimited traffic. The following instructions can be used to install the program under Ubuntu. For other environments the link to the instructions can be found under "References"[22].

To install geth add the repository first:

```
$ sudo add-apt-repository -y ppa:ethereum/ethereum
```

Next, install geth:

```
$ sudo apt-get update
$ sudo apt-get install ethereum
```

To interact with the Ethereum Blockchain the entire chain history has to be downloaded first. This can take up to 40 GB of disk space and will take several hours of synchronizing. Geth will be started via this console command:

```
$ geth console --rinkeby --rpc --rpcapi="db,eth,net,web3,
personal,txpool" --rpcaddr X.X.X.X --rpcport 8545 --cache=1024
```

The launch options have the following purposes[23]:

- *rinkeby*: synchronizes the Rinkeby testnet
- *rpc*: enables the HTTP-RPC server, allows to receive JSON RPC requests
- *rpcapi*: exposed APIs, listing of all APIs can be found under "References"[24][25]
- *rpcaddr*: IP address of RPC interface, replace "X.X.X.X" with the IP address of the server, defaults to "localhost". Exposing the RPC interface without any restrictions is not advisable, especially on the mainnet, as it's a severe security concern.
- *rpcport*: listening port of the RPC server
- *cache*: memory allocated in MB, a minimum of 1024 MB is advisable for a faster synchronization

The console parameter starts a JavaScript console, allowing to interact with the blockchain using the web3 library. Geth currently comes with web3 version 0.20.1[26] but might be upgraded to version 1.0[27] soon. The links to the documentation of both versions can be found under "References".
The version of the web3 library can be checked using:

```
> web3.version
```

The output of the synchronization could hamper the ability to properly read the output of the JavaScript console. The verbosity can be set via the following command:

```
> debug.verbosity(x)
```

Replace x with 0 for silent, 1 for error, 2 for warn, 3 for info, 4 for debug and 5 for detail. The verbosity defaults to info[23].

```
> eth.blockNumber
```

returns the block number of the latest synchronized block, which is the amount of all previous blocks. The first block, the also called the genesis block, starts with the block number 0. The current block number can be checked through so-called block explorers, e.g. rinkeby.etherscan.io.

```
> eth.syncing
```

returns the current block number and the highest block number. As soon as the client is synchronized it returns "false"[26].

4.2.5 Smart Contract

4.2.5.1 Wallet The first thing needed to start programming smart contracts is an Ethereum wallet. MetaMask is a browser extension for Chrome, Firefox and Opera, that not only allows to manage multiple accounts on multiple test chains, it also injects the web3.js library into websites allowing to interact with the Ethereum blockchain and smart contracts on web pages. Visit the MetaMask[28] website and download the browser extension. A new account will be generated using a mnemonic phrase, defined in the BIP39 (Bitcoin improvement proposal)[29]. It usually consists of 12 words which represent a private key, essentially creating an easy way to remember / write down private keys. An example for a mnemonic phrase is:

```
short heavy hidden anger nephew tragic
fade dad renew finger among tiny
```

This phrase translates to the following seed:

```
b7b36d9ca1e105045344ecb7ca7b9449bfc0889139c9719876d03cf7b5814
86137e905b9e94e50c03ca22871937ae3c754dea1427eede8198c6774d90f
c1a1f4
```

Using the BIP44[30] standard an unlimited amount of private keys can be derived from the seed. For example the first private key derived from this seed would be:

```
0xfb8502c03ea336344dc44b66b1a3c01e
  2917138e92bfa93c54725166394cd46b
```

with the corresponding address

```
0x4d43c1E254a9333fB0D8A50BD3f01b6787ee8895
```

The second derived private key is:

```
0x64b45c024041178aff2f9ed7b7026fff
  6890c871818c39c1c7bd826e6aa33773
```

with the corresponding address

0xd38F7dc2d9B6F6D9d5CB6C8813e213D5DC541458

and so on. This means that the single mnemonic phrase will act as a backup phrase for all accounts that will be created inside the MetaMask wallet. After MetaMask was set up create a second account and set the network to Rinkeby.

4.2.5.2 Getting Ether The next step would be to get Ethers on the Rinkeby network to interact with the Blockchain, via the website faucet.rinkeby.io. A public Facebook post or Twitter tweet containing the desired destination address has to be provided to receive the Ethers.

4.2.5.3 IDE The Smart Contract was developed using the Remix. It's an online IDE including a compiler for the language Solidity, various debugging and testing tools and can be found under remix.ethereum.org.
First a compiler has to be set inside the "Compile" tab. Because the programming language was designed especially for Ethereum, it is still under very heavy development with frequent updates coming out. The documentation for each specific version can be found under
`https://solidity.readthedocs.io/en/v0.5.8/`
while replacing "0.5.8" (the latest stable version at the time of writing) with the desired compiler version.

Inside the "Run" tab the connection to the blockchain can be chosen under "Environment". The most important options are:

- *JavaScript VM*: A personal Ethereum blockchain implemented in JavaScript that runs locally. It comes with 5 accounts which are preloaded with 100 Ether. It's suited for early development stages, unit testing, and all in all quick tests, as there are no transaction times.
- *Injected Web3*: As mentioned before, MetaMask injects Web3 into websites. When choosing this option, the currently active account and the chosen network in MetaMask are used for development.

Underneath the Smart Contract can be chosen and deployed to the blockchain. Next to the deploy button, constructor arguments can be passed. Optionally an existing Smart Contract can be loaded from an address.

After a Contract has been deployed or loaded, all functions and public variables will be listed under the "Deployed Contracts" section. This will be the main way to interact with the Smart Contract.

4.2.5.4 Solidity This paragraph will explain the basics and key features of the Solidity programming language. A solidity source file has the ".sol" file extension. The language has a C++ style syntax and works very similarly to object-oriented programming and is also called contract-oriented[31]. Contracts can be viewed as classes and also work with interfaces and inheritance.
See 4.1 for an example of a minimalistic smart contract, which demonstrates the general syntax as well as some of the features listed below.

General notice
There are a few important aspects in how certain things behave during smart contract programming and execution:

- Solidity does not implement floats at the time of writing. This is especially important for calculations using Ether. Therefore it's important to remember that Ether is always assumed as Wei by Smart Contract functions.
- When a function throws, all changes to the state that were made up to this point are reverted and the transaction is marked as failed.
- *undefined* and *null* does not exist in Solidity, variables rather have a default type, e.g. 0 for integers[32].

Types
The most important data types in Solidity are[32]:

- *bool*: The possible values are "true" and "false".
- *integer*: There are signed (*int*) and unsigned (*uint*) integers in Solidity. *uint* is an alias for *uint256*. The smallest size for an integer is 8 bit (e.g. *uint8*), the sizes grow by 8 up to 256 bit. The same applies to signed integers as well.
- *address*: The address variable stores 20 bytes.
- *address payable*: The address payable variable stores 20 bytes as well. Additionally it holds the members *transfer* and *send* to send Ether to that address.
- *bytes32*: Holds 32 bytes of data.
- *bytes[]*: Dynamically-sized byte array.
- *string*: Dynamically-sized UTF-8 encoded string.

Pragma
The first line of a solidity file defines the compiler version[33]:

```
pragma solidity ^0.5.4;
```

The ^ symbol means that the code can be compiled by a compiler with the versions 0.5.4 and above, but below 0.6.0. Without the ^ symbol only the compiler version 0.5.4 can compile the source code.

Important variables and functions Solidity features some global units, variables and functions which can be very useful, if not necessary for Smart Contract programming:

- *ether*: The ether unit multiplies the current value by 10^{18}, e.g. 3 ether equals 3000000000000000000.
- *time units*: The following time units are available: "seconds", "minutes", "hours", "days", "weeks". 1 seconds equals 1, 1 minutes equals 60 seconds or 60, 1 hours equals 60 minutes or 3600 and so on.
- *now*: An alias for *block.timestamp*, a *uint256* variable as seconds since unix epoch. The variable is set by the miner during validation so it should not be used for random number generation as the number can be varied by ± up to 15 seconds, because the timestamp of a block has to be higher than the one of the previous block.
- *msg.sender*: Of type *address payable* and contains the sender of the current message. If a function is called directly by an EOA and not by a Smart Contract, *msg.sender* will contain the sender of the transaction.
- *msg.value*: Of type *uint256* and contains the number of Wei sent with the current message.
- *<address payable>.transfer(uint256 value)/<address payable>.send(uint256 value)*: Both functions send "value" in Wei to the payable address. Transfer throws, send returns false on failure.
- *function ()*: A function declared without any name is also called the fallback function. This function is called, when no matching function identifier is provided. Usually this function is triggered, when a simple transaction is sent to the smart contract, that's why the fallback function can often be seen with the *payable* modifier.
- *require(bool condition, string memory message)*: The require function throws if the condition is not met and provides the custom error message.
- *keccak256(bytes memory) returns (bytes32)*: Computes the Keccak-256 hash of an input.

- *ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s) returns (address)*: For a given message and r, s, v values of a ECDSA signature the function returns the address associated with the public key from the signature.

Constructor
A constructor function is optional and can be used to execute code directly when the Smart Contract is written to the blockchain[34].

Modifiers
A function can have multiple different modifiers, that make it behave in different ways[35]. First, there is the visibility modifier, which manages the access to functions and variables:

- *public*: The public modifier makes a function visible from inside and outside the smart contract. Adding the modifier to a variable automatically generates a getter function with the same name as the variable.
- *external*: The external modifier is only applicable for functions. It makes them only visible from outside the smart contract. To call the function from inside the smart contract it has to be called via "this.func()" instead of "func()".
- *internal*: The internal modifier makes a function or variable only visible internally. This means they can only be accessed from the contract and all derived contracts.
- *private*: The private modifier makes a function or variable only visible from the contract itself. It's important to notice that although a variable might be private, it still can be read, since all data stored on the blockchain is public.

Reading from the blockchain does not require mining or involve transaction fees. Therefore functions that do not write to storage can be marked as such via a modifier:

- *view*: If a function has a view modifier, it cannot write to storage, only read from it.
- *pure*: If a function has a pure modifier, it cannot modify storage. Additionally it cannot read state, e.g. read from variables. It's primarily used for computations.

The *payable* modifier allows a function to receive Ether. If Ether is included in a function call that doesn't have the *payable* modifier, it throws.

It's also possible to write custom modifiers. The example 4.1 will show the function of a custom modifier through a commonly used *onlyOwner* modifier, which only allows the owner of a smart contract to call the function.

Events

Events are an important part of Smart Contract programming for 2 key reasons:

1. With a complex Smart Contract handling thousands of transactions, it can get confusing to keep track of all changes made to the state of the Smart Contract. Events are a good way to sort these changes into different categories and make them searchable by specific filters.

2. Some time passes until a transaction was mined, therefore the return value of a function is not returned to the sender of the transaction. Events can be used to act as a return value. A computer system or a frontend can then scan for these events and act accordingly, e.g. show updates to the user.

See 4.1 for an example of an event.

```
pragma solidity 0.5.8;

contract ModifierExample{
    // variable to store the owner of the smart contract
    address owner;
    // number to demonstrate the view function
    uint256 public num;

    // definition of an event
    // the indexed keyword allows to use the parameter as a filter
    event ownerChanged(address indexed oldOwner, address indexed newOwner);

    // the constructor gets called after contract creation
    // arguments can be passed to the constructor
    constructor(uint256 _num) public {
        // set owner to the sender of the transaction
        owner = msg.sender;
        // set the number to the passed argument
        num = _num;
    }

    // custom modifier
    modifier onlyOwner() {
        // throws if sender of call is unequal to value stored in owner variable
        require(owner == msg.sender, "sender is not owner");
        // _; defines where the code of the function, the modifier is used on, runs
        _;
    }

    // function to change the address of the smart contract
    function changeOwner(address _owner) external onlyOwner {
        // emits the ownerChanged event
        emit ownerChanged(owner, _owner);
        // sets the owner variable to the passed argument
        owner = _owner;
    }

    // the function will execute the code and return the calculated number
    // because of the view modifier no state is modified and no transaction has to be sent
    function calculatedNum() public view returns(uint256) {
        return num * 2;
    }

}
```

Listing 4.1: Basic structure of a smart contract

5 Appendix

List of Figures

List of Tables

List of Listings

List of Abbreviations

References

[1] ETH Gas Station. Fees. https://ethgasstation.info/, 2019. [Online; accessed 2019-05-06].

[2] blockchain.com. 7 day average chart. https://www.blockchain.com/de/charts/transactions-per-second?daysAverageString=7. [Online; accessed: 2019-05-06].

[3] Visa. Annual report 2018. https://s1.q4cdn.com/050606653/files/doc_financials/annual/2018/Visa-2018-Annual-Report-FINAL.pdf. [Online; accessed: 2019-01-05].

[4] Shrimpton T. Rogaway P. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. *Roy B., Meier W. (eds) Fast Software Encryption. FSE 2004. Lecture Notes in Computer Science, vol 3017. Springer, Berlin, Heidelberg*, (2004).

[5] Satoshi Nakamoto. Bitcoin white paper. https://bitcoin.org/bitcoin.pdf, 2008. [Online; accessed 2019-05-06].

[6] Dr. Gavin Wood. Ethereum yellow paper. https://ethereum.github.io/yellowpaper/paper.pdf, 2019. [Online; accessed 2019-05-06].

[7] Scott Vansto Don Johnson, Alfred Menezes. The elliptic curve digital signature algorithm (ecdsa). https://web.archive.org/web/20170921160141/http://cs.ucsb.edu/~koc/ccs130h/notes/ecdsa-cert.pdf. [Online; accessed 2019-05-06].

[8] Vitalik Buterin. Toward a 12-second block time. https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/, 2014. [Online; accessed 2019-05-06].

[9] etherscan.io. Ethereum average block time chart. https://etherscan.io/chart/blocktime, 2019. [Online; accessed 2019-05-06].

[10] Paypal. Fees for micro transactions. https://www.paypal.com/de/webapps/mpp/paypal-fees, 2019. [Online; accessed 2019-05-06].

[11] blockchain.com. Average confirmation time. https://www.blockchain.com/en/charts/avg-confirmation-time?timespan=all&daysAverageString=7. [Online; accessed: 2019-05-12].

[12] Serguei Popov. The tangle. https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf, 2018. [Online; accessed 2019-05-06].

[13] The Tangle Monitor. Average confirmation time. https://tanglemonitor.com/, 2019. [Online; accessed 2019-05-06].

[14] Brian Pugh. Stress testing the raiblocks network: Part ii. https://medium.com/@bnp117/stress-testing-the-raiblocks-network-part-ii-def83653b21f, 2018. [Online; accessed 2019-05-06].

[15] Repnode.org. Block propagation and confirmation times. http://repnode.org/network/propagation-confirmation, 2019. [Online; accessed: 2019-02-22, 14:54].

[16] Colin LeMahieu. Nano: A feeless distributed cryptocurrency network. https://nano.org/en/whitepaper. [Online; accessed 2019-05-06].

[17] Allegro MicroSystems Inc. Fully integrated, hall effect-based linear current sensor. https://www.sparkfun.com/datasheets/BreakoutBoards/0712.pdf. [Online; accessed: 2019-05-11].

[18] Silicon Labs. Cp210x usb to uart bridge vcp drivers. https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers. [Online; accessed: 2019-05-12].

[19] Links2004. Websocket server and client for arduino. https://github.com/Links2004/arduinoWebSockets. [Online; accessed: 2019-05-12].

[20] Kenneth MacKay. micro-ecc. https://github.com/kmackay/micro-ecc. [Online; accessed: 2019-01-15].

[21] Ethereum Foundation. Go ethereum. https://github.com/ethereum/go-ethereum. [Online; accessed: 2019-05-11].

[22] Ethereum Foundation. Installing geth. https://github.com/ethereum/go-ethereum/wiki/Installing-Geth. [Online; accessed: 2019-05-12].

[23] Ethereum Foundation. Command line options. https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options. [Online; accessed: 2019-05-12].

[24] Ethereum Foundation. Json rpc. https://github.com/ethereum/wiki/wiki/JSON-RPC. [Online; accessed: 2019-05-12].

[25] Ethereum Foundation. Management apis. https://github.com/ethereum/go-ethereum/wiki/Management-APIs. [Online; accessed: 2019-05-13].

[26] Ethereum Foundation. Javascript ethereum api 0.2x.x. https://github.com/ethereum/wiki/wiki/JavaScript-API. [Online; accessed: 2019-05-13].

[27] Ethereum Foundation. Javascript ethereum api 1.0. https://web3js.readthedocs.io/en/1.0/. [Online; accessed: 2019-05-13].

[28] Metamask. Wallet browser extension. https://metamask.io/. [Online; accessed: 2019-05-13].

[29] Bitcoin. Bip39 mnemonic phrases. https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki. [Online; accessed: 2019-05-13].

[30] Bitcoin. Private key derivation. https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki. [Online; accessed: 2019-05-13].

[31] Ethereum Foundation. The solidity contract-oriented programming language. https://solidity.readthedocs.io/en/v0.1.5/README.html. [Online; accessed: 2019-05-14].

[32] Ethereum Foundation. Types. https://solidity.readthedocs.io/en/v0.5.3/types.html. [Online; accessed: 2019-05-16].

[33] Ethereum Foundation. Layout of a solidity source file. https://solidity.readthedocs.io/en/v0.5.8/layout-of-source-files.html. [Online; accessed: 2019-05-14].

[34] Ethereum Foundation. Constructor. https://solidity.readthedocs.io/en/v0.5.8/contracts.html. [Online; accessed: 2019-05-15].

[35] Ethereum Foundation. Modifiers. https://solidity.readthedocs.io/en/v0.5.8/miscellaneous.html#function-visibility-specifiers. [Online; accessed: 2019-05-15].