CHALMERS

Applications of smart-contracts and smart-property utilizing blockchains

Master of Science Thesis in Computer Science: Algorithms, Languages and Logic

Erik Hillbom
Tobias Tillström

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, February 2016

Applications of smart-contracts and smart-property utilizing blockchains

ERIK HILLBOM
TOBIAS TILLSTRÖM

Examiner: David Sands
Supervisor: Katerina Mitrokotsa

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31-772 1000

Department of Computer Science and Engineering
Gothenburg, Sweden, February 2016

Abstract

Bitcoin, one of many cryptocurrencies, has in the last couple of years grown into a multi billion dollar industry. It is fully decentralized and utilizes a public ledger (blockchain), which allows for the currency to function without a central authority. As the Bitcoin protocol contains a programming language, it has potential to be used for much more than exchanging currency.

This thesis is about exploring the possibility of combining cryptocurrency with a concept called 'smart-contracts'. The term smart-contracts was coined in a paper written by Nick Szabo in 1997, thus predating Bitcoin by 12 years. In contrast to paper based contracts, these are computer protocols facilitating an agreement between parties. We have put extra emphasis on two types of contracts: a generic type called 'Double-Deposit-Escrow' (DDE), and one involving 'smart-property'. DDE achieves a double deposit escrow within the blockchain, allowing users to perform business with untrusted parties with minimum risk of losing money. Smart-property may be described as the ability of property itself to be part of a contract.

By refining Szabo's ideas using current technology, we have implemented a self-enforcing smart-contract executing the trade of smart-property referred to as the Smart-Property Ownership Exchange Protocol (SPOEP). It was implemented in Python and supports anonymous trades using Bitmessage, as well as NFC. We have analyzed our proposed protocol in terms of security and scalability, and compared it with related projects such as Ethereum and Colored coins.

There are several viable approaches for creating smart-contracts using cryptocurrency. Albeit not perfect, we have deemed Bitcoin to be the currently most suited cryptocurrency to be used for this purpose.

Keywords: Smart-contracts, Smart-property, Bitcoin, Colored-coins, Ethereum

Acknowledgements

First, we would like to thank our supervisors Kasper Karlsson and Martin Altenstedt at Omegapoint for their help and support throughout the project, and giving us the opportunity of performing this at Omegapoint.

We would also like to thank our supervisor Katerina Mitrokotsa and examiner David Sands at Chalmers University of Technology for her help and feedback throughout the project.

Erik Hillbom and Tobias Tillström, Gothenburg 2016-02-01

Contents

1

Introduction

This thesis is performed in collaboration with Chalmers University of Technology and the Swedish consulting firm Omegapoint. The purpose of this thesis is to research a concept called 'smart-contracts'. Smart-contracts eliminate the need of trust when entering agreements, and as will be shown, can be fully peer-to-peer.

1.1 Background

A smart-contract is a concept of a computer protocol facilitating an agreement between different parties. The contract may be self-enforced; a third party cannot enforce or interfere with the agreement. There are several areas of application for smart-contracts, one of which being 'smart-property'. Smart-property may be a car which knows its owner, where ownership is based on a transferable but non-forgeable digital token. A smart-contract can for example be set up to govern the transfer of ownership and accompanying rules; the digital token becomes a part of the contract itself.

While the idea of smart-contracts has been around for nearly two decades, little progress has been made implementation-wise until recently. With the release of modern cryptocurrencies, such as Bitcoin [1], this is starting to change.

Cryptocurrency is a form of digital money that relies on cryptography for security. In contrast to traditional currency, it typically features decentralized control, removing the need of a central issuer. Another implication of decentralization is that it removes the need of relying on third parties (such as banks) when transferring funds.

The idea of cryptocurrency is believed to originate from Wei Dai, who published a description of 'B-money' in 1998 [2]. B-money is described as an anonymous distributed electronic cash system and was thought of as a conceptual idea, rather than practical.

In 2005, Nick Szabo wrote a blogpost[1] where he proposed a new type of digital currency named "bit-gold". Bit-gold relied on the idea that a participant would solve

[1] http://unenumerated.blogspot.se/2005/12/bit-gold.html

cryptographic equations, provide proofs of these solutions to other participants, and a reward would be granted based on the amount of work that was needed. Each solution would be part of the next challenge, effectively forcing participants to verify current solutions before working on new ones. Apart from working as an incentive for participants to verify each others work, all new coins would at the same time be given a timestamp. While this proof-of-work scheme managed to assign value to computational power, it never gained recognition as it left a lot of unanswered questions. A major issue was how to value work, as different input data would require a varying amount of work. Bit-Gold is commonly viewed as a precursor to Bitcoin.

Bitcoin was proposed by the pseudonym Satoshi Nakamoto in 2008 and released in 2009. It solved all issues of Bit-gold and is the most popular cryptocurrency today [3]. The most notable contribution of Bitcoin is the blockchain (section 3.1.4), which is a distributed public ledger. Being open source, Bitcoin has been copied and altered numerous times and these projects are commonly known as 'altcoins'.

It is possible to extend the functionality of Bitcoin by inserting metadata within transactions. By writing protocols that interpret the metadata as 'assets', tokens of smart-property may be defined. A generic name for these protocols is 'Colored coins' (see section 3.3).

Ethereum (section 3.4) is a cryptocurrency sometimes referred to as 'Bitcoin 2.0' and is set to go live 'sometime this year'. We believe it will suffer from enormous scalability issues in regards to blockchain size, as smart-contracts are to be part of the blockchain itself.

1.2 Problem description

As of today, entering an agreement requires a trusted third-party to enforce that which was agreed upon. This requirement raises several issues; for instance, the third-party might be malicious, or not have sufficient information in order to resolve a potential dispute. A common problem arises when purchasing items online - who does a company side with if a dispute arises and there is no proof of who is being honest?

Another interesting question is how to securely implement ownership control, once again without the need of a trusted third-party.

In this thesis we investigate viable solutions to both problems by implementing smart-contracts utilizing cryptocurrency. With Bitcoin being the most thoroughly researched cryptocurrency, it has been selected as the basis for the implementations. Other approaches, such as using other types of cryptocurrencies, are analyzed as well.

1.3 Limitations

We limit ourselves to two types of smart-contracts: smart-property and Double Desposit Escrow. Smart-property is demonstrated by a proof-of-concept (chapter 4), which is an implementation that enables secure trading of cars by storing transferable tokens in a blockchain. The Double Deposit Escrow contract is described in section 2.2.2.

The frameworks we have chosen to analyze, in terms of viability of implementing smart-contracts in, are Bitcoin, Ethereum and Colored coins.

1.4 Related work

There are multiple projects claiming to support smart-contracts 2015. Due to being similar to the ones we have limited ourselves to, they will not be part of this thesis.

1.4.1 Bithalo / Blackhalo

In 2014, David Zimbeck released Bithalo[2] and Blackhalo[3]. These are two coin clients that have built-in support for Double Deposit Escrow contracts (section 2.2.2). Instead of being cryptocurrencies by themselves, they use the blockchains of Bitcoin and Blackcoin respectively. Zimbeck claims that his clients were the first to support smart-contracts, but they are as of yet not released as open source and have so far gained small recognition.

The reason why Blackcoin was chosen, as an alternative to Bitcoin, is its superior transaction times. Blackcoin uses a proof-of-stake protocol [4], which enables transactions to go through within minutes, in contrast to Bitcoin where a single transaction might require hours. The drawback is potential security issues, one being the "nothing-at-stake" problem [5].

1.4.2 Codius

At the start of writing this thesis, Codius [6] was an open source project in early stages of development developed by Ripple Labs (mostly known for their cryptocurrency Ripple). It provided both a smart-contract engine and the means to build a peer-to-peer network consisting of multiple Codius hosts. As stated in the documentation, the implementation of smart-property was an anticipated application, and had thus not yet been implemented.

Codius was discontinued during the course of writing this thesis[4].

1.4.3 Counterparty

Counterparty[5] is a cryptocurrency with full support for smart-contracts. Just as with colored coins, it uses the Bitcoin blockchain to store metadata. Contract code is run by Counterparty clients, and is said to be (mostly) compatible with Ethereum contract languages.

[2] https://www.bithalo.org
[3] https://www.blackhalo.info
[4] https://codius.org/blog/codius-one-year-later
[5] http://counterparty.io

2

Smart-Contracts

The term 'smart-contract' was coined by Nick Szabo in his paper *Formalizing and Securing Relationships on Public Networks* [7]. The idea behind smart-contracts is described as moving existing contractual clauses, such as collateral and bonding, into embedded hardware and software in such a way that breaching a contract becomes expensive.

While Szabo did not have a specific system for implementing smart-contracts, trust in a central authority was assumed to be required to some extent. With the release of cryptocurrencies, the idea of smart-contracts has rapidly regained momentum, as they provide a secure way of proving performance in a decentralized manner.

2.1 Objectives

Szabo's proposed design of a smart-contract is based on a two-phase model schema used in legal theory: ex-ante and ex-post[1]. The contractual phases are structured as:

Ex-Ante — "Before the event"

Search — "Gather information"
Negotiation — "Agree on terms"
Commitment — "Execution of obligations"

Ex-Post — "Actual result"

Performance — "What transpired"
Adjudication — "Verdict"

Szabo focuses mostly on performance. He assumes intermediaries may be used and defines three main objectives of a smart-contract in terms of the the phases listed above: observability, verifiability and privity.

[1]Legal terms: http://lsolum.typepad.com/legal_theory_lexicon/2003/09/legal_theory_le_2.html

2.1.1 Observability

It must be possible for the principles to observe each other's performance, or to prove their performance to other principals. A lack of observability may allow hidden knowledge to be utilized in the search and negotiation phases, and in combination with the inability to drop out of a contract during the performance phase, hidden actions may occur during the performance phase.

2.1.2 Verifiability

To evaluate performance, it must be possible for an adjudicator to verify that a contract has been performed or breached. This is only possible if a proof is provided by a principal, or if it is possible to find out by other means.

2.1.3 Privity

Privity is a term mostly used in contract law, referring to the connection between parties to a particular transaction. Szabo defines privity in this context to mean 'the principle that knowledge and control over the contents and performance of a contract should be distributed among parties only as much as is necessary for the performance of that contract'. This implies that a third party should be limited in terms of knowledge and control. Thus, achieving privity is in conflict with the use of third parties as a mean of adjucation.

2.2 Smart-contracts using cryptocurrencies

By basing smart-contracts on cryptocurrencies, the need of a trusted third-party may be eliminated altogether. In the following subsections, three types of smart-contracts are described proving this point.

2.2.1 Smart-Property

Amongst several proposed smart-contract applications by Szabo, one involves a digital security system for property. The idea is to embed security protocols in property involving actual contractual terms (turning it into 'smart-property').

One example of such property is a car, where a contract would give control of the cryptographic keys for operating the vehicle to a person based on the terms of the contract. Implementing this scheme using a cryptocurrency has been suggested [8], and is the basis for our proposed protocol SPOEP (chapter 4).

2.2.2 Double Deposit Escrow

In 2014, a white-paper named *Two Party double deposit trustless escrow in cryptographic networks and Bitcoin* [9] was released by David Zimbeck. Zimbeck proposes a smart-contract which sets up a multi-signature transaction stored in the Bitcoin blockchain,

which is used as a deposition account. Two parties agree on some terms prior to committing their deposits, and unless both parties agree that the terms have been met, the funds are automatically donated to the Bitcoin mining network.

Both parties are thus incentivized to fulfill the predetermined terms, or they both lose their deposits. The generality of the protocol makes it suitable for any type of commitment. The only requirement is that the size of the deposit must be notably greater than the value of the involved item or service. Below follows a technical rundown of Alice and Bob using the protocol.

1. Bob creates a transaction, tx_1, which spends funds into a temporary funding account of his (but does not broadcast it). A hash representation of tx_1, $H(\text{tx}_1)$, and a newly generated Bitcoin address, p_B, is sent to Alice:

$$B \rightarrow A : \{H(\text{tx}_1), p_B\}$$

2. Similarly, Alice creates a transaction, tx_2, spending funds into a temporary account of hers and generates a new Bitcoin address, p_A. She proceeds by generating a multisignature address, p_{AB}, based on p_A and p_B, which will used as the deposition account. She creates and signs a transaction tx_3, which spends tx_1 and tx_2 into p_{AB}.

   She also creates a transaction, $\text{tx}_{\text{timeout}}$, which spends tx_3 with a 99,9 % mining fee. It is constructed using a feature called *locktime*, which dictates the earliest time a transaction may be added to the blockchain. The time has been agreed upon in advance. Transaction $\text{tx}_{\text{timeout}}$ is used as a penalty to both parties if the contract terms are not met in time. She signs all transactions and sends the now partially signed transactions tx_3 and $\text{tx}_{timeout}$ to Bob:

$$B \rightarrow A : \{\text{tx}_3, \text{tx}_{\text{timeout}}\}$$

3. Bob reviews the transactions and, if everything is correct, signs. Both parties now have a fully signed $\text{tx}_{\text{timeout}}$, which guarantees that when the depositions are in place, both parties may enforce the penalty if the contract is breached.

   He sends all transactions, including the non-hashed tx_1, back to Alice:

$$B \rightarrow A : \{\text{tx}_1, \text{tx}_3, \text{tx}_{\text{timeout}}\}$$

4. Alice is now able to broadcast all the transactions and the depositions are then stored in the blockchain.

Using temporary funding accounts avoids the possibility of early broadcasts; since Alice does not have the raw format of tx_1, and it is not yet funded, she cannot broadcast it (or any transaction that uses tx_1 as an input) prematurely.

When a party has fulfilled the terms of the contract, a transaction tx_4 may be created which returns the depositions. Thus, the two possible outcomes after establishing the deposition account are:

- Both participants sign tx_4 to return the deposition (contract has been performed)
- At least one participant chooses not to sign tx_4 (contract has been breached)

3

Theory

This chapter aims to describe the technical workings of cryptocurrencies (Bitcoin and Ethereum), existing smart-property implementations (Colored coins) and the communication protocol Bitmessage.

3.1 Bitcoin

Bitcoin is fully decentralized and runs on a peer-to-peer network. Anyone with a Bitcoin client is a node in the network and every node exchanges addresses, transactions and blocks with other nodes. A signed transaction gets broadcasted to known nodes, who in turn relay it to their known nodes. Participants known as miners pick up transactions, attempts to generate a block, and eventually the block becomes part of the blockchain. As every node has a copy of the blockchain[1], all clients see that the transaction has been processed. The details of these steps are further explained in the following subsections.

3.1.1 Keys

The first key that needs to be generated is a random 256-bit private key K_{priv} which is used for signing transactions:

$$K_{\text{priv}} \in \{0,1\}^{256}$$

It is crucial that a cryptographically strong random number generator is used; in 2013 it was revealed that all Android Bitcoin wallets were susceptible to attacks due to a flaw in the pseudo-random number generator SecureRandom. The default Bitcoin client uses RAND_bytes, which is part of the OpenSSL library[2], and is considered to be secure.

[1] With the exception of lightweight clients who fetch blockchain information remotely
[2] OpenSSL Github repository: https://github.com/openssl/openssl

A 512-bit public key is generated by using the private key as an input to the Elliptic Curve DSA algorithm:

$$K_{\text{pub}} = \text{ECDSA}_{512}(K_{\text{priv}})$$

The public key is not revealed until a transaction gets signed. Instead, using the hash functions RIPEMD_{160} and SHA_{256}, a 160-bit public key hash is shared and used as a recipient address of bitcoins:

$$K_{\text{address}} = \text{RIPEMD}_{160}(\text{SHA}_{256}(\text{SHA}_{256}(K_{\text{pub}})))$$

As a consequence, the security is reduced from 256 bits to 160. Assuming RIPEMD_{160} is a perfect oracle (it has no known security flaws[10]), the most efficient way to find a collision would be to perform a Birthday attack, requiring an average of 2^{80} tries before one is found. If an adversary would possess enough hashing power to make such an attack feasible, it would still be more profitable to use it for mining[3].

Bitcoin supports multisignature addresses as well. Instead of requiring a single private key to sign a transaction, m-of-n ($0 < m \leq n < 15$) keys may be required. One application of this scheme is the Double Deposit Escrow contract described in chapter 2.

3.1.2 Transactions

A transaction moves bitcoins between one or many inputs and outputs. An input may be seen as a previous transaction supplying bitcoins, and an output as a destination for said bitcoins.

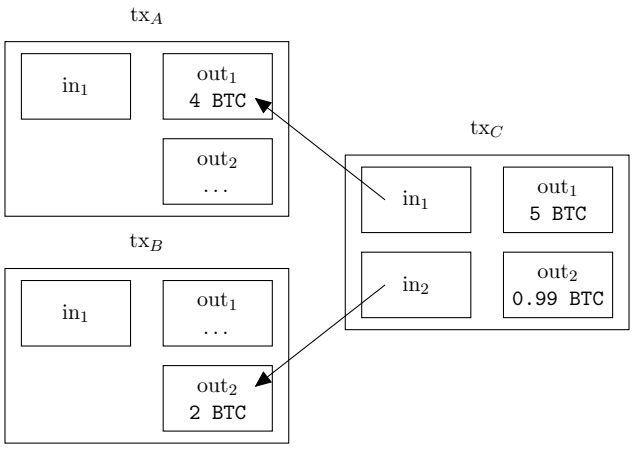

Figure 3.1: Outputs of both tx_A and tx_B are referred to as inputs 1 and 2 of tx_C

Consider figure 3.1 as an illustratory example of Alice sending 5 bitcoins to Bob by broadcasting transaction C. As inputs, she references two previous transaction outputs

[3] http://bitcoin.stackexchange.com/a/31

$(A_{\text{out}_1}, B_{\text{out}_2})$ where she was the recipient of $4+2$ bitcoins. When referencing a previous output, the entire value must be spent, which is why she creates two outputs C_{out_1} and C_{out_2}. The value of C_{out_1} is 5 bitcoins and is directed to Bob. The value of C_{out_2} is the change amount directed back to herself. The difference between the value of the inputs and outputs is considered a mining fee, which is rewarded to the miner who incorporates the transaction into a block.

The Bitcoin protocol contains a stack-based scripting language called Script[4]. Script contains approximately 80 operation codes including hash functions, stack manipulation and arithmetic. Constructing a transaction, one may choose whether a pay-to-pub-key-hash (p2pkh), or a pay-to-script-hash (p2sh) script is to be used. In the previous example, a p2pkh script could be used, implying that "the owner of the included public key hash must sign in order to spend these bitcoins in a future transaction". By adding a public key hash belonging to Bob in C_{out_1}, Bob is considered to be the owner of the 5 bitcoins. This illustrates the fact that there are no bitcoins per se, but rather transactions stating which keys may sign for bitcoins to belong to someone else.

By using a p2sh script, far more complex transactions are possible by combining the various operation codes of the Script language. The most common use case is when a multisignature scheme is desired; in section 2.2.2, a transaction is constructed stating that two out of two signatures are required to move bitcoins.

Every transaction includes a unique[5] id tx_{id}, which is needed for tracking purposes as well as for security reasons. A tx_{id} is a double SHA_{256} hash digest of the binary representation of the transaction involved:

$$\text{tx}_{\text{id}} = \text{SHA}_{256}(\text{SHA}_{256}(\texttt{transaction}))$$

From an endpoint user's perspective, not much is required to send a regular transaction if using the standard Bitcoin client. It includes a graphical interface, where all that needs to be specified is an address to send bitcoins to and the amount. The client finds unspent outputs to spend from and creates a change address automatically. For advanced transactions, the included Bitcoin daemon (bitcoind) may be used via a command prompt or suitable programming library.

3.1.3 Signatures

The signatures of transactions are located in the scriptSig field of the inputs. The data that gets signed is the hash of the entire transaction, which makes the process quite complex; a signature must be present before signing. Below is a simplified[6] summary of the signing process:

1. Use the scriptPubKey in the referenced output as a temporary signature:

$$\text{scriptSig} = \text{scriptPubKey}'$$

[4]https://en.bitcoin.it/wiki/Script
[5]https://en.bitcoin.it/wiki/Transaction_Malleability
[6]The full process of creating a transaction: http://bitcoin.stackexchange.com/a/5241

2. The data to be signed is a hash of the entire transaction tx:

$$\text{data} = \text{SHA}_{256}(\text{SHA}_{256}(\text{tx}))$$

3. Sign the data using the private key matching the public key hash of the referenced output:

$$\text{signature} = \text{ECDSA}_{512}(\text{data}, K_{\text{priv}}, \dots)$$

4. Replace the temporary signature with the signature concatenated with the public key:

$$\text{scriptSig} = \text{signature} \parallel K_{\text{pub}}$$

A signature serves two purposes. Firstly, it proves that a user has bitcoins to spend. Secondly, as the data signed includes the current transaction, it prevents man-in-the-middle attacks; if an adversary intercepts the transaction and redirect outputs elsewhere, the signature becomes invalid. Similarly, the signature scheme also ensures safe multi-signature transactions; when a transaction has been signed by one participant, others may not alter the content prior to signing, or the first signature would become invalid.

3.1.4 The blockchain

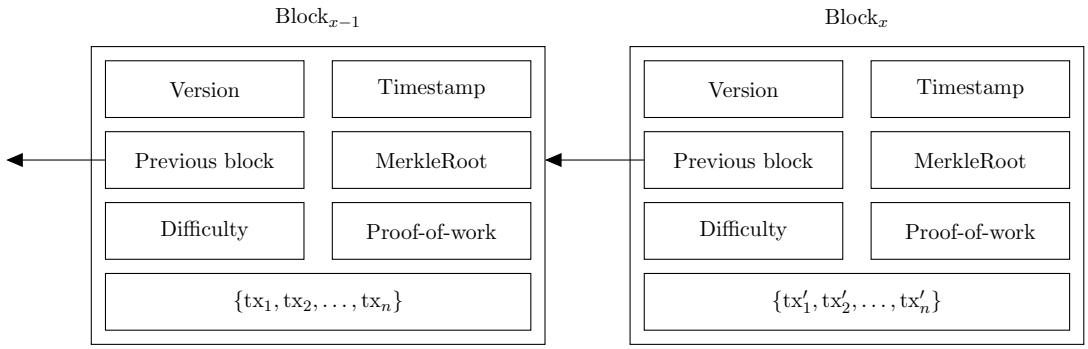

Figure 3.2: Bitcoin blockchain

The blockchain is a series of blocks that are linked together; each block has a reference to the hash of its predecessor. As a consequence, it is impossible to change the content of a block without affecting every subsequent block.

A block is created by a miner and consists of a block header (metadata), and a list of transactions. The full content of a block is depicted in figure 3.2. In the block creation process, miners receive transactions broadcasted by regular Bitcoin users. The miner must then solve a computationally hard problem and provide a proof-of-work (section 3.1.4.2) before the block may be accepted by the network. To improve scalability, the block header includes a merkle root.

3.1.4.1 Merkle Trees

The merkle root included in the blockheader produces an overall digital signature of the list of transactions included in the block. This gives a very efficient method to verify whether or not a transaction actually is included in a block [11].

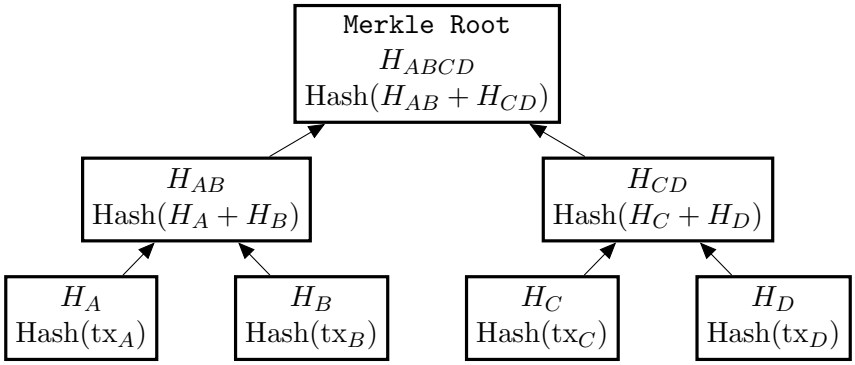

Figure 3.3: Node calculation in a merkle tree.

The merkle tree (figure 3.3) is constructed by first hashing each individual transaction, creating the leaf nodes in the tree. These nodes are then hashed together pairwise, resulting in a set of nodes one step up in the tree. The process repeats until there is only one node left, which becomes the merkle root.

The inclusion of a merkle root in blocks enables light-weight clients, clients that only store the blockheaders, to still be able to verify transactions. If Alice wishes to prove to Bob that transaction tx_K is included in Block_x, she can simply supply the set $\{\text{tx}_K, H_L, H_{IJ}, H_{MNOP}, H_{ABCDEFGH}\}$ (see figure 3.4) to Bob. Bob can then easily verify that tx_K is indeed included in Block_x by calculating the merkle root himself and compare with the one found in the blockheader. Since the structure of a merkle tree is the same as for a binary search tree, this verification would on average take $\mathcal{O}(\log n)$ time, and at most $\mathcal{O}(n)$ time (where n is the number of transactions). By only storing block headers, the required storage space of the blockchain is reduced by up to 99,9 %; the current maximum block size is 1 MB out of which 80 bytes constitute the header.

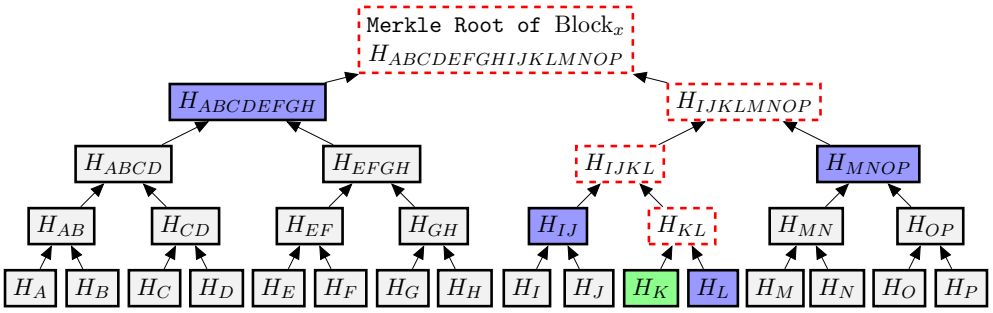

Figure 3.4: Only the elements highlighted in blue are needed to prove the inclusion of element H_K in the merkle root of Block$_x$.

3.1.4.2 Proof-of-work

The proof-of-work problem consists of finding an integer (nonce), such that when hashed together with the rest of the block header data, the resulting output has at least a given amount of leading zeros. The amount of zeroes depends on the current mining difficulty, which is dynamically changed every 2016th block. It is based on the total amount of hashing power in the entire mining network. The goal of the mining difficulty is that it should always take ten minutes on average to solve every problem and its purpose is to prevent double spending (section 3.1.5). The proof-of-work algorithm is depicted in figure 3.5.

```
header ← version ‖ prevBlock ‖ merkleRoot ‖ timeStamp ‖ target
nonce ← hash ← 0
while target > hash do
    nonce ← nonce + 1
    hash ← SHA256(SHA256(header ‖ nonce))
end
return nonce
```

Figure 3.5: The proof-of-work algorithm used by Bitcoin [12].

The incentive for a miner to carry out the work is monetary reward. The miner who is first to present a valid block to the network receives all the transaction fees[7] of the transactions included in the block. Additionally, the miner has included a coinbase transaction in the transaction list, which gives the miner a fixed amount of Bitcoins (a block reward). As the coinbase transaction is unique to each miner, every miner is working on a unique problem. The blockreward was originally 50 bitcoins and is halved every 210 000 th block ($\approx$ 4 years). These rewards is the only point where new coins are introduced in Bitcoin. Thus, the maximum amount of Bitcoins to ever come into

[7]Transaction priority is based on optional transaction fees.

existence is:

$$\sum_{n=0}^{\infty} \frac{50 \cdot 210\,000}{2^n} = 21 \cdot 10^6$$

A miner is incentivized (but not required) to validate the transactions included in a block prior to attempting to find the proof-of-work. If the block contains invalid transactions, other nodes will reject it. The validation is a security measure in place to prevent double spending.

3.1.5 Double spending

Double spending is the result of the same Bitcoins being spent more than once. It is made possible by a criteria used by nodes to accept new blocks; if multiple blocks are candidates to be added to the blockchain, the one that has the longest chain is selected. There are three well-known attacks that enable double spending: the majority attack[8], the race attack and the Finney attack.

3.1.5.1 Majority attack

The way the Bitcoin network is constructed makes it vulnerable to what is called a majority attack. This attack is based on the assumption that there might exist a possible user, who is in possession of more than half of the total hashing power in the network.

Assume malicious Malie is such a powerful user. Malie has the possibility to control whether or not a transaction will be added to the blockchain. Malie can commit to buying an item from Alice by broadcasting a transaction tx_i, which gets added to the blockchain in the block B_n, whereupon Alice grants Malie the item. The current state of the blockchain is illustrated in figure 3.6.

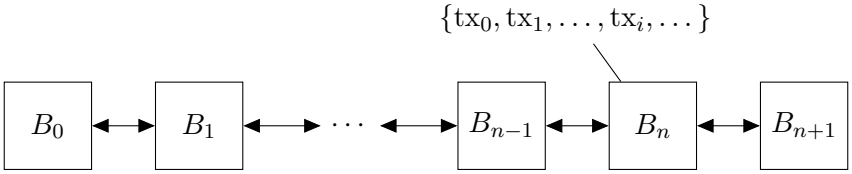

Figure 3.6: A blockchain where transaction tx_i exist within the list of block B_n

Malie proceeds by creating her own version of the blockchain where B_{n-1} is the latest block. Malie may now include tx'_i, which has the same input as tx_i but the output is redirected somewhere else, in a new block as its input has not been spent in the new version of the blockchain. (see figure 3.7).

[8] A majority attack is often referred to as a 51 % attack

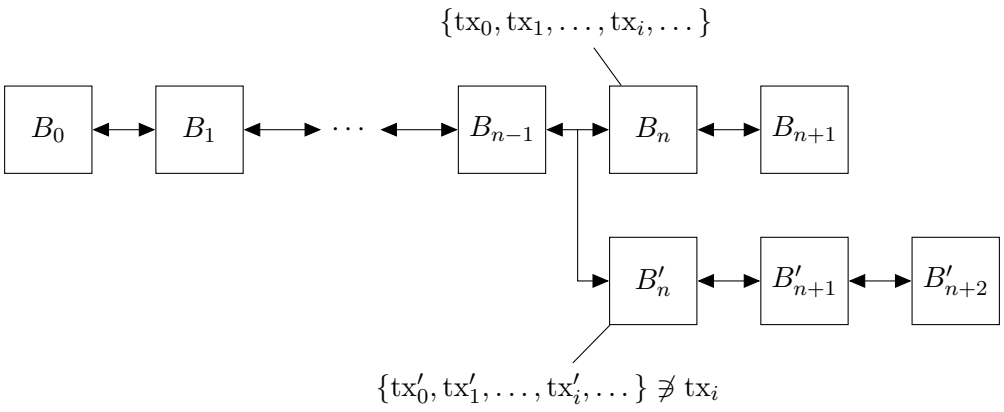

Figure 3.7: A forked blockchain where transaction tx_i gets replaced with tx'_i

Malie will as always need to provide a proof-of-work for every new block until her blockchain is longer than the real blockchain. This is bound to happen at some point since she is in control of the majority of the hash power of the network. At that point, she broadcasts her blockchain and has now successfully created a fork of the blockchain where tx_i is not included. This fork will be accepted by the rest of the network since the Bitcoin network will always choose to follow the longest blockchain [13].

It is highly unlikely that there exists such a user due to the fact that it would require an enormous amount of computing power, but if there exist one, that user would succeed in removing tx_i from the blockchain [12].

In order to avoid a potential majority attack, it is recommended that one should wait for at least six transaction confirmations[9] before being fairly sure that the transaction is secure. The probability of succeeding the attack decreases exponentially for every new block that the attacker has to catch up with [1].

3.1.5.2 Race attack

If a merchant accepts a payment immediately, without seeing any confirmations on the transaction tx_1, she will be susceptible to a double-spending attack from the buyer. The buyer can simply create another transaction tx_2 with the same inputs, but with other outputs, and broadcast it to the network. The buyer can hope that the some miner will create a valid block which includes tx_2 and adds it to the blockchain before any other block containing tx_1 gets added [14].

The merchant should always be well connected to the Bitcoin network so that she can broadcast the new transaction to a large amount of miners. This way, the transaction should get added before the buyer is be able to race his new transaction into the blockchain. These precautions will of course not always prevent double spending, so in

[9] One transaction confirmation equals one reference to the transaction in the blockchain. A transaction included in block B_n will have 3 confirmations in block B_{n+2}.

order for the merchant to be sure that she will receive her funds, she should always wait for some confirmations on the transaction before releasing her goods to the buyer.

3.2 Bitmessage

Bitmessage is a secure peer-to-peer communications protocol used to send encrypted messages between people. Like Bitcoin, Bitmessage is both decentralized and trustless. Bitmessage applies strong authentication, implying that one cannot spoof the identity of a sender, and it also hides the metadata of a message (such as sender and receiver) from wiretapping systems [15].

In the implementation of SPOEP (see chapter 4), Bitmessage is used to ensure integrity and confidentiality.

3.2.1 Addresses

Elliptic Curve Cryptography is the scheme used to generate the private and public key pairs by the Bitmessage protocol. The address used by the client is a double SHA_{512} hash of the public key encoded in Base_{58} [16]. The private key can either be randomly produced, or deterministically generated using a passphrase, where the former is recommended for security, and the latter is for example used in multiple system applications.

3.2.2 Network structure

Each client connects to what is referred to as a stream. A stream is a subset of the total number of clients within the network. The purpose of a stream is simply to not overwhelm the client. Once a client starts exceeding comfortable processing thresholds, it will merge into a new child stream, and thus it need only to maintain connections with the peers that are also members of this current child stream.

Unless the client is part of the root stream of the network, it should occasionally connect to clients in its parent stream in order to advertise its existence. Each client will thus maintain a list of peers in the current stream, and the client is also aware of peers within its two child streams. Furthermore, the client will also keep track of a couple of peers within the root stream in order to make sure that it will be able to send messages across the network even if the recipient stream is unknown to the client.

3.2.3 Message transfer

To send a message, the client connects to the recipient stream encoded within the recipient address. If the client is unaware of any peers within the recipient stream, it will instead connect to closest parent stream for which it knows any active node, and from this node it will download a list of active peers within its child streams.

This process is then done iteratively until the client reaches the recipient stream, into which it can send the message. The client will now await an acknowledgement from the intended recipient, and it can now disconnect from the peers of this stream.

3.2.4 Security and privacy

When sending a message over the Bitmessage network, the client encrypts both the message and its metadata with the recipient's public key before it is broadcasted to the network. None of the nodes within the network knows to whom the message is intended since the metadata is encrypted. Instead, all clients will try to decrypt every message. This way, the recipient is hidden from view, and only the intended recipient will be able to read the content of the message.

3.2.5 Spam prevention

A client cannot broadcast just any message to the network, or more precisely, the network will not further relay just any message. This is because Bitmessage uses a proof-of-work (PoW) schema similar to the one of Bitcoin (see section 3.1.4.2). This means that the client always needs to provide a valid proof-of-work with every message it tries to broadcast to the network. A message that does not have a correct proof-of-work attached will be dropped and ignored. This way, the client cannot spam the network with messages since the proof-of-work problem takes a considerable amount time to solve for each new message. The problem to solve is defined as finding a hash value of a nonce concatenated with the payload, which is less than the target:

$$\texttt{target} = \frac{2^{64}}{\texttt{nonceTrialsPerByte}\left(\texttt{payloadLength} + \texttt{extraBytes} + \frac{\texttt{TTL(payloadLength + extraBytes)}}{2^{16}}\right)}$$

The variables `nonceTrialsPerByte` and `extraBytes` are user-defined; since a client may wish to ignore messages that are too small or with proofs that were too easy to calculate, the sender may want to increase these variables. TTL (Time To Live) is defined as the number of seconds $t_{\text{expires}} - t_{\text{now}}$.

All messages are in the form of `nonce` $\|$ `payload`, and the algorithm used to generate the correct `nonce` is depicted in figure 3.8. To verify the proof-of-work, one simply repeats the process of hashing the nonce (the first 8 bytes) and the payload (bytes 9-) and checks if the resulting integer is less than the defined target.

initialHash $\leftarrow$ SHA_{512}(payload)
trialValue $\leftarrow$ 99999999999999999999
nonce $\leftarrow$ 0
while trialValue $>$ target **do**
  nonce $\leftarrow$ nonce + 1
  resultHash $\leftarrow$ $\text{SHA}_{512}(\text{SHA}_{512}$(nonce $\|$ initialHash))
  trialValue $\leftarrow$ the first 8 bytes of resultHash converted into an integer
end
return nonce

Figure 3.8: The proof-of-work algorithm used by Bitmessage [17].

3.2.6 Average times

According to the Bitmessage documentation[10], the average time (for an average computer) of the various processes in Bitmessage are:

Action	**Time**
Address generation	1 second †
1-2 characters shorter address generation	5 minutes †
Do work necessary to send broadcast	2 minutes
Do work necessary to send message	4 minutes
Message propagation through the network	10 seconds

† The client then continues doing work to send out the public key to the network (≈ 2 minutes).

Figure 3.9: Average Bitmessage times.

3.3 Colored coins

'Colored coins' is a generic name for protocols that are used to issue assets into the Bitcoin blockchain. A selection of colored coins implementations are Coinprism[11], Coinspark [12] and Bitcoinx[13]. A common denominator for these protocols is that they all work directly on top of the Bitcoin protocol by using an instruction called OP_RETURN. If OP_RETURN is found in a transaction output script, the subsequent instructions will not be executed. Thus, the remaining bytes of the script may be used as store space for metadata.

A popular basis for colored coins is the Open Assets Protocol[14] (OAP), which will be used as a representative for colored coins throughout the rest of this section.

3.3.1 Assets

Issuing an asset is almost identical to generating addresses in Bitcoin. The first step is to generate a private key:

$$K_{\text{priv}} \in \{0,1\}^{256}$$

The corresponding public key is calculated:

$$K_{\text{pub}} = \text{ECDSA}_{512}(K_{\text{priv}})$$

[10]https://bitmessage.org/wiki/PyBitmessage_Help
[11]Coinprism website: https://www.coinprism.com
[12]Coinspark website: http://coinspark.org/
[13]Bitcoinx Github repository: https://github.com/bitcoinx
[14]OAP Github repository: https://github.com/OpenAssets/open-assets-protocol

The *pubkeyhash* (address) is calculated:

$$K_{\text{address}} = \text{RIPEMD}_{160}(\text{SHA}_{256}(\text{SHA}_{256}(K_{\text{pub}})))$$

A standard pay-to-pubkey-hash script[15] is constructed:

$$\texttt{scriptPub} = \texttt{OP\_DUP OP\_HASH160 } K_{\text{address}} \texttt{ OP\_EQUALVERIFY OP\_CHECKSIG}$$

The asset id is generated by converting the hash of the script to a more human readable format (a Base_{58} string). OAP uses a different checksum for Base_{58} (version byte 23) in order to have all asset ids begin with an 'A' [16].

$$\text{asset}_{\text{id}} = \text{Base}_{58}(\text{RIPEMD}_{160}(\text{SHA}_{256}(\texttt{scriptPub})))$$

Issuing and transferring assets is done by broadcasting transactions that contain metadata describing asset ids and quantities. Each output of a transaction that is to be considered 'colored' must include a non-null asset id and a positive asset quantity.

3.3.2 Transactions

An OAP transaction can either issue new assets or transfer existing ones and is identified by a marker output. The full content of such an output is depicted in figure 3.10.

[15] Script documentation: https://en.bitcoin.it/wiki/Script
[16] Example asset id: AHthB6AQHaSS9VffkfMqTKTxVV43Dgst36

Field	Description	Size
OP_RETURN opcode	The OP_RETURN opcode (0x6a).	1 byte
PUSHDATA opcode	The PUSHDATA opcode required to push the full payload onto the stack (0x01 to 0x4e, depending on the size of the payload).	1-5 bytes
OAP Marker	A tag indicating that this transaction is an Open Assets transaction. It is always 0x4f41.	2 bytes
Version number	The major revision number of the Open Assets Protocol. For this version, it is 1 (0x0100).	2 bytes
Asset quantity count	A var-integer representing the number of items in the asset quantity list field.	1-9 bytes
Asset quantity list	A list of zero or more LEB128-encoded unsigned integers representing the asset quantity of every output in order (excluding the marker output).	Variable
Metadata length	The var-integer encoded length of the metadata field.	1-9 bytes
Metadata	Arbitrary metadata to be associated with this transaction. This can be empty.	Variable

Figure 3.10: Marker output as summarised on the OAP github repository.

Outputs in a transaction placed prior to the marker are used for asset issuance (anyone in possession of the private key that was used to create an asset may use it for reissuing).

Outputs placed after the marker are assigned the asset id referenced by the **first** input of the transaction, if they have a non-zero asset quantity. Any remaining outputs are considered to be uncolored (regular outputs).

Assigning asset ids to outputs is done by *order-based coloring*. The transaction inputs are interpreted as a sequence of asset units and asset ids that will be assigned to the outputs. Each input and output is selected in order, and each adds the number specified in the asset quantity list. The protocol proceeds by assigning the asset ids of the inputs to the outputs of the same index.l

3.4 Ethereum

Ethereum is an ongoing project striving to become much more than a cryptocurrency in its traditional sense. As with Bitcoin, it may be used to exchange monetary value. Its purpose is however to be used as a platform for running decentralized applications and smart-contracts. It features a turing-complete programming language, which makes it possible to create any type of contract. These contracts have no restrictions in terms of size and are stored on the blockchain. A measure to prevent the blockchain to grow

unfeasible large is the requirement that contracts are 'paid for', using an internal currency named 'ethers'. The contract code is executed as a part of the block validation process performed by the miners.

Ethereum was originally proposed in 2014 by its co-creator V. Buterin [18]. The following subsections provide an overview of the Ethereum documentation [19].

3.4.1 Accounts

There are two types of accounts in Ethereum. One being 'externally owned', meaning it is controlled by private keys (just as with Bitcoin); a transaction may be sent by providing a correct signature. The other type is having the account be controlled by contract code; when the contract account receives a message, the code is executed. The account may communicate with other accounts, create new accounts and read/write within the storage field.

An account contains a 20-byte address as well as four fields: a *nonce* (incremented with every transaction made), the current *ether balance*, *contract code* (optional), and *data storage*.

3.4.2 Messages

Contracts may communicate with other contracts. This is achieved by sending messages, that contain five fields: a *sender address*, a *recipient address*, an *ether amount*, a *data field* (optional), and a *startgas value*. Obviously enough gas needs to be sent to comply with the requirement of the contract at hand.

3.4.3 Transactions

A transaction is a signed message containing six fields: a *recipient address*, an *ether amount* to be sent, a *data field* (optional), a *startgas value* (restricting the maximum number of computational steps allowed), and a *gasprice value* (the fee paid for each computational step). The startgas and gasprice variables are used as a measure against denial-of-service attacks; without them infinite loops would be possible. In addition, five ethers must be paid for every byte of the datafield in order to prevent the blockchain from growing too large. Thus, a potential attacker would need to pay for every resource consumed.

3.4.3.1 State transitions

Transfers of value or information between accounts may be defined in terms of state transitions. Depicted in figure 3.11 is the Ethereum state transition function, which is to be applied to every transaction included in a block:

- Check if the transaction is well-formed[17], else return an error.

[17] Well-formed transaction if: correct number of values, signature is valid and nonces match.

- Calculate transaction fees and verify that there is enough balance in the sender's account to spend, if not, return an error.
- Initialize GAS to the supplied STARTGAS minus storage costs.
- Transfer the payment to the recipient. If the receiving account is a contract, then execute the contract code, either to completion or until the execution runs out of GAS.
- If the sender did not have enough ethers, or if the execution of the contract ran out of GAS, all state changes are reverted (except the payment of fees). Else, the remaining GAS fees are refunded to the sender, and the fees paid for the GAS consumed is rewarded to the miner.

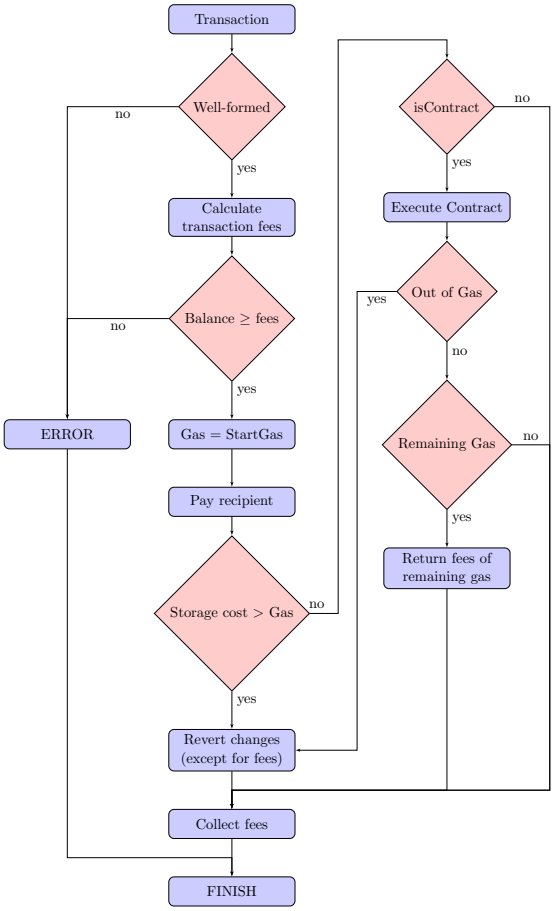

Figure 3.11: Flowchart visualising the Ethereum state transition function.

3.4.4 Blockchain

As with Bitcoin, Ethereum blocks contain transactions, proof-of-works, timestamps, references to previous blocks etc. The major difference is that included in every Ethereum block, is a copy of the global state. The global state contains all existing accounts, including contract code, data storage and balances. A special type of tree data structure has been implemented to make storing that much data viable: a Merkle Patricia Tree[18]. The tree structure allows for $\mathcal{O}(\log n)$ efficiency for look-ups, insertions and deletions.

In terms of scalability, the developers claim that the tree structure should make Ethereum comparable to Bitcoin and is motivated by two reasons. Firstly, only a small part of the tree needs to be modified between adjacent blocks. Thus, data can be stored once and referenced by by pointers to subtrees. Secondly, the entire blockchain is not needed; only the latest block needs to be stored, as it contains all of the state information.

[18]Merkle Patricia Tree: https://github.com/ethereum/wiki/wiki/Patricia-Tree

4

Smart-Property Ownership Exchange Protocol

This chapter describes the proposed smart-contract (protocol) for secure ownership exchange of smart-property referred to as SPOEP. The goal of the smart-contract is to construct a transaction, which in one atomic step pays the seller and reassigns ownership to the buyer. Ownership change is defined as a Bitcoin transaction where the last ownership transaction is used as an input in a new transaction. Ownership itself is represented as a public key hash used in the transaction, which differs from the current ownership hash.

4.1 The protocol

In the following sections, a basic notation of encryption and decryption of the secure Bitmessage communication between each participant is denoted with index e and d respectively:

$$B \rightarrow S : \{m, m_{d_B}\}_{e_S}$$

In the example above, the buyer B sends a message m via Bitmessage to the seller S. The buyer also signs (decrypts) the message m using his private key, denoted as m_{d_B}. The message and its signature are both encrypted using the seller's public key, denoted as $\{\dots\}_{e_S}$.

4.1.1 Protocol overview

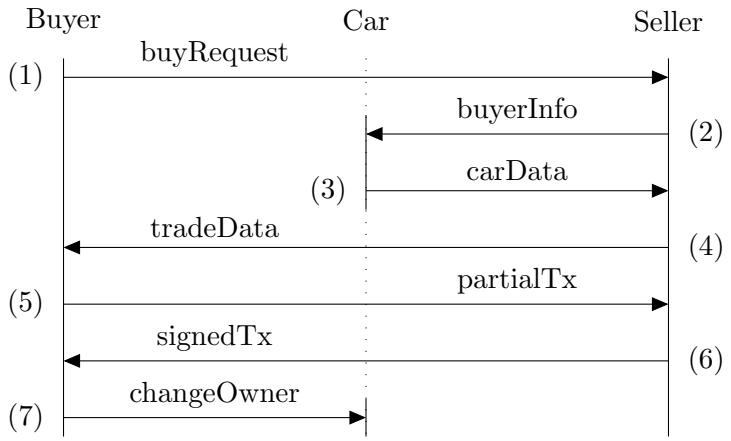

Figure 4.1: Communication scheme of SPOEP

The buyer starts by generating a 128 bit random nonce N, and a new public key K_{new}, which is to become the new owner key of the car after the transaction is complete.

(1) $B \to S : \{\text{buyRequest} = \{N, K_{new}\}\}_{e_S}$

The seller relays the buyRequest to the car.

(2) $S \to C : \{\text{buyRequest}, \text{buyRequest}_{d_S}\}_{e_C}$

The car verifies that the request originates from its current owner. If so, the car replies with a message containing internal data (such as its unique identifier, its public key, and a digital certificate issued by the car manufacturer) together with buyRequest and a signature. The car is now also actively awaiting an owner change request.

(3) $C \to S : \{\text{carData} = \{\text{internalData} = \{\dots\}, \text{buyRequest}\}, \text{carData}_{d_C}\}_{e_S}$

The seller relays the carData to the buyer.

(4) $S \to B : \{\text{carData}, \text{carData}_{d_C}\}_{e_B}$

The buyer verifies that the carData corresponds to the predetermined conditions of the contract. If so, the buyer constructs and signs a transaction partialTx reassigning the ownership of the car, based on what was agreed upon in the contract specification. It is not complete until both parties have signed, and is thus named partial.

(5) $B \to S : \{\text{partialTx}\}_{e_S}$

The seller verifies that the partial transaction created by the buyer contains the correct information (such as the correct price). If so, the seller also signs the partial transaction, creating the complete transaction signedTx, which the seller broadcasts to the network.

(6) $S \rightarrow B : \{\text{signedTx}\}_{e_B}$

The buyer has the transaction needed to prove to the car that he is its new owner, and thus passes the transaction to the car.

(7) $B \rightarrow C : \{\text{signedTx}\}_{e_C}$

4.1.2 Protocol analysis

Stage 1

Random number generation is performed using OpenSSL. The protocol implements a cryptographically secure pseudo-random-number-generator (PRNG). The seed used by the PRNG is consists of a system-specific entropy source, in Linux-based based systems this is usually the /dev/urandom [20].

The new ownerkey is generated in two steps. A 256-bit Bitcoin public key generated by feeding the Elliptic Curve Digital Signature Algorithm (ECDSA_{512}) a 512 bit random number [21]. The public key is then hashed using RIPEMD_{160} in order to reduce key to a more manageable size of 160 bits.

The first message transferred consists of an encrypted message containing the new ownership key, a randomly generated nonce, a Bitmessage recipient address, and a signature. The message is encrypted by using the public Bitmessage-key of the current owner, and is signed using the private Bitmessage-key of the buyer. The message is broadcasted to all known Bitmessage nodes, that all try to decrypt the message in order to find out if they are the intended recipient (by successfully decrypting and parsing the recipient address).

So in order for an adversary to claim ownership of the car, a private key must be found that corresponds to the 160 bit hash. In order for an adversary to eavesdrop on the message content throughout the communication, the private Bitmessage keys of the two parties must be found. Calculating these keys is currently infeasible [21]. The security aspects of encrypting, decrypting and signing messages are to be implied in the following stages.

Stage 2-3

Once the seller has received the first message, he signs it using the private key corresponding to the current ownerkey. After verifying the signature, the car responds with a signed message containing its current ownership status, together with a certificate issued by the car manufacturer. If this is a trusted certificate, the buyer can be sure of the validity of the car.

Stage 4-5

The seller decrypts the received message, encrypts it with the public key of the buyer, and sends it to the buyer. The message now contains the car message, signed by the private key of the car.

The buyer verifies the certificate and compares the car message with the content of the pre-determined contract specification (see appendix A.1.1 for a specification example). If everything checks out, the process of creating an ownership change in the form of a Bitcoin transaction is begun.

The transaction includes two inputs and two outputs, representing the transfer of a token amount of bitcoins moving from the current ownerkey to the new ownerkey, as well as the price in bitcoins moving from the new ownerkey to the old. The logic of determining the new owner is simply choosing the public key that is not the current ownerkey.

The buyer can only sign the part of the transaction that is to move bitcoins from the new ownerkey to the new. Once this signature is in place, any modifications to the transaction, other than adding a signature for moving the token amount from the current ownerkey, would invalidate the signature, leading to the transaction being rejected by the Bitcoin network. Thus, the seller can only choose whether or not to sign the transaction; the amount received by the seller cannot be modified in order to cheat the buyer. An example of a partially signed transaction may be found in appendix A.1.2 beginning at row 158.

Once the (now partially signed) transaction has been created, it is sent via Bitmessage to the seller.

Stage 6-7

The seller verifies that the correct amount of bitcoins are to be moved between the correct addresses, and if so adds the final signature to the transaction. It is now up to the seller to broadcast the transaction to the Bitcoin network in order to actually move the funds. An example of a finalised ownership transaction may be found in appendix A.1.3.

Possible outcomes

- **The seller decides not to broadcast the transaction**: The seller gets to keep the car but cannot claim the funds of the buyer. The buyer can still spend his funds elsewhere, as the transaction input (see section 3.1.2) used for moving the funds is still spendable.
- **The buyer spends the bitcoins marked for buying the car before the seller has broadcasted the transaction**: The ownership change transaction is rejected by the Bitcoin network, as the input used for moving funds to the seller becomes invalid. The seller gets to keep the car.

Thus, the only two possible outcomes are either that the transaction goes through, or the funds and the car remain with its original owners.

4.2 Test environment

The protocol was implemented in Python 2.7 using the Bitcoin library *bitcoin-rpc*. We added three different choices for communication: Bitmessage, a custom Bitmessage simulation, and Near Field Communication (NFC) using a pair of Android phones. Instead of using real bitcoins, a private testnetwork was set up by using the bitcoin-core client with the regtest-flag set. A Raspberry Pi 2+ running Ubuntu Mate was used to represent the car. The Bitmessage implementation was tested using Ubuntu 14.04, and the Android implementation using a Google Nexus 4 and a Oneplus One. Support for Windows was deemed unnecessary, but would only require minor changes in the code if desired.

4.2.1 Bitmessage

The scripts communicate with Bitmessage via Remote Procedure Calls (RPCs). This feature needs to be enabled in the configuration file by adding:

```
1 apienabled   = true
2 apiport      = xxxx
3 apiinterface = 127.0.0.1
4 apiusername  = username
5 apipassword  = password
```

Figure 4.2: Additions in the Bitmessage configuration file in order to enable RPC calls.

As stated in chapter 3.2, sending messages is a time consuming process, whereas generating an address is done in a second. For initial testing purposes we implemented a hybrid version of the contract, which uses Bitmessage to generate addresses but relies on file-IO for sending / receiving messages locally. This enables quick testing; multiple scenarios of running the contract may be tested in a matter of seconds.

4.2.2 NFC

Two android phones running an API version of at least 15 is required, as well as a NFC-reader connected to the Raspberry Pi. The code was tested using the reader ACR122U[1].

For consistency purposes, we ported the Python code directly to Android using python-for-android. The logic of the contract is exactly the same, but obviously a GUI

[1] http://www.acs.com.hk/en/products/3/acr122u-usb-nfc-reader

and communication alterations were needed. To compile the code we recommend using Buildozer[2].

4.2.3 How to run

For the Android version, four python scripts were implemented and converted to Android APKs; two for the initialisation process (assign the first owner key to the car), and two for the buyer and seller respectively.

We begin by setting up a new car, and proceed by pairing the phones either with each other or the car depending on where we are in the protocol. A button for 'unlocking' the car is available in both the `Buy` and the `Sell` application. Below is a full run of the protocol:

- The seller runs `firstSeller`, which creates a newly funded Bitcoin address, which is presented to the car.
- The car runs `carSetup` and waits for a Bitcoin address that is to become the initial owner key.
- The seller runs `Sell` and follows the instructions on the screen.
- The buyer runs `Buy` and follows the instructions on the screen.

For the Linux version, three scripts were implemented. The car is initialised by running `carSetup`, a script that has access to the Bitcoin client of the car as well as the seller's client[3]. The buyer runs `Buy` and the seller runs `Sell`. As the scripts are implemented as state machines, the scripts will pick up wherever they left off in case they get interrupted at any point.

[2] https://github.com/kivy/buildozer

[3] Giving access to the seller's client is purely for convenience purposes; in real life it should rather be set up as in the Android version.

5

Results

In this chapter the result of comparing different approaches to implementing smart-contracts is presented. As we concluded that using the raw Bitcoin protocol for implementing smart-contracts would be the most resource efficient approach, we also implemented the SPOEP-protocol described in chapter 4.

5.1 Bitcoin

As stated in chapter 3.1, Bitcoin was deemed secure due to using strong cryptography and its large consensus network. However, due to its long transaction times, it is arguably not optimal for smart-contracts. A transaction should be confirmed every ten minutes (but as seen in figure 5.1, it is usually somewhat faster). To ensure that a transaction does not get reversed, six confirmations are actually recommended (see section 3.1.5.1), leading to unfeasible long waiting times in many types of contracts.

In terms of scalability, the growth rate and size of the blockchain must be taken into consideration. As seen in figure 5.2, the size of the blockchain grows exponentially and is currently 36 GB.

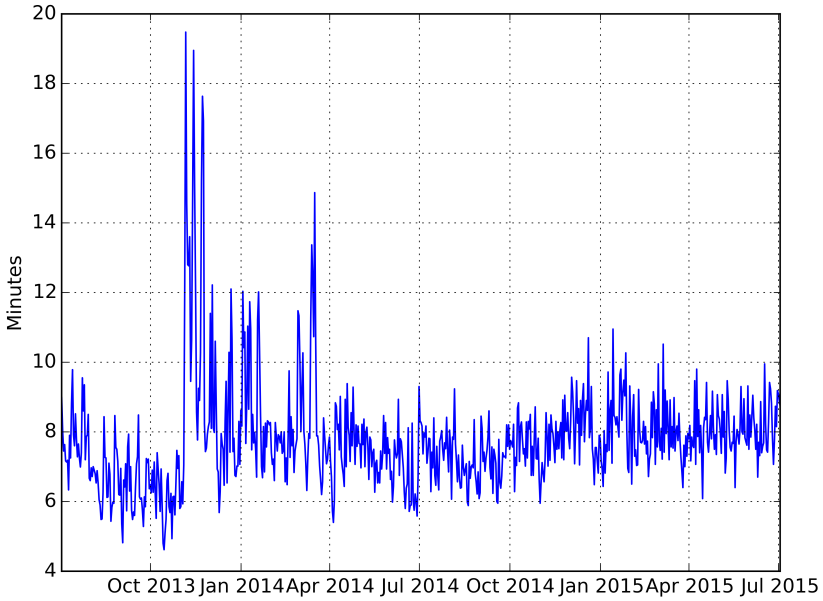

Figure 5.1: Average Bitcoin transaction confirmation time [22].

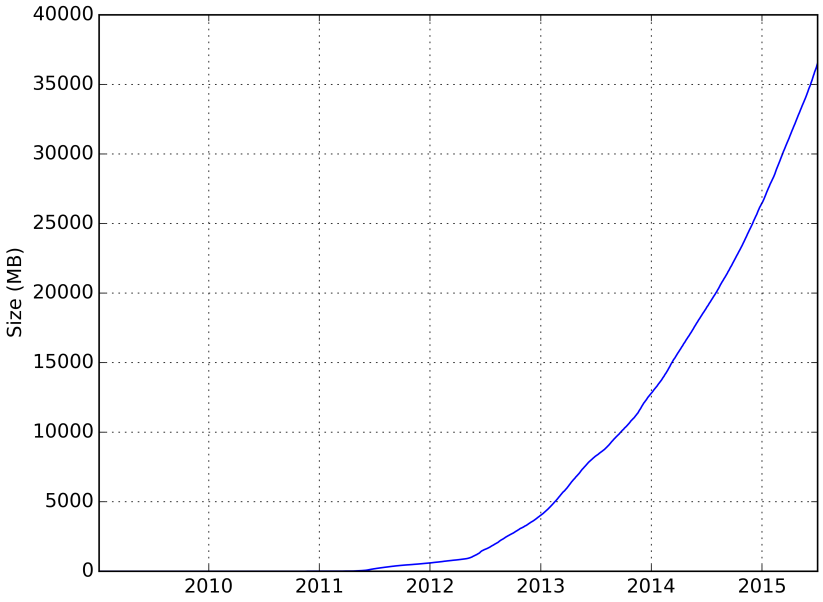

Figure 5.2: Bitcoin blockchain size [23].

5.2 Colored coins

Colored coins (CCs) implementations rely on using the 'OP_RETURN' instruction in Bitcoin to insert metadata into transaction outputs. Many CC implementations, such as Coinprism, are based on the Open Assets Protocol (OAP). OAP transactions are identified by the byte sequence representing the ASCII string 'OP_RETURN OA'. Other CC implementations, such as Coinspark, use the same logic but other identifiers. Coinspark transactions use the identifier 'OP_RETURN SPK'.

We scanned the last 10 000 blocks[1] in the Bitcoin blockchain for these two identifiers and compared the daily average transaction sizes. These blocks contained 8 489 812 transactions, 34 497 had an OAP identifier (0,41 %) and 9 339 a Coinspark identifier (0,11 %). The result is shown in figure 5.3.

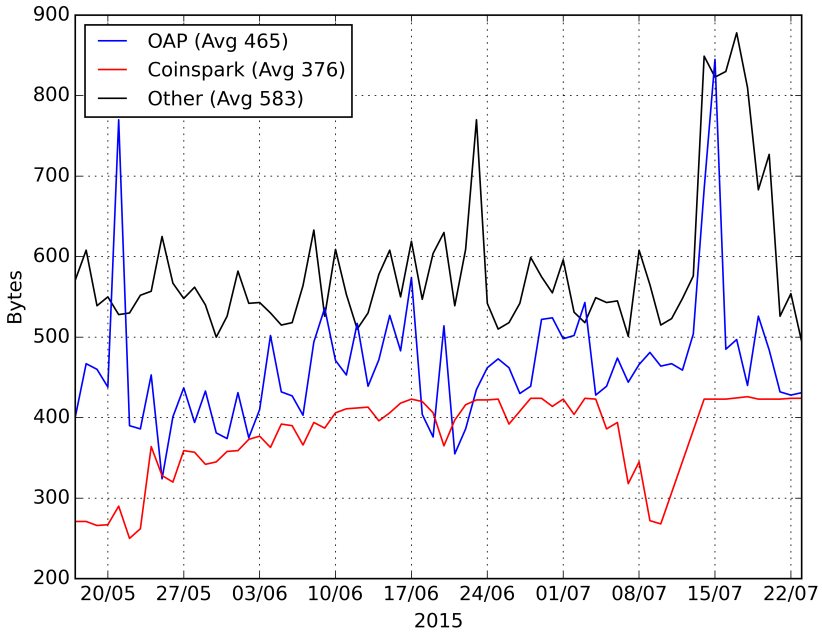

Figure 5.3: Daily average transaction sizes.

The results suggest that today's usage of colored coins has less of an impact on the Bitcoin blockchain size than regular types of transactions.

5.3 Ethereum

At the time of writing, Ethereum has not yet been released. The only obtainable data comes from analysing performance within the Ethereum test-network. As the currency

[1]Scanned blocks: #356602 - #366601

within this network holds no value, the current blockchain growth rate gives no indication of how the real network will perform once online; participants lose nothing by creating expensive transactions. However, test data does indicate that the average transaction time correlates to the intended 12 seconds in the current version [24].

5.4 SPOEP

We implemented a protocol for securely transferring ownership of smart-property (a car) peer-to-peer using the Bitcoin protocol. The only footprint in the blockchain is a regular transaction where the buyer receives a valid owner key of the car and at the same time the seller gets funded. Instead of utilising metadata for asset creation and contract details, assets are defined in terms of regular Bitcoin keys and outputs, and contracts as code running independently of Bitcoin.

The size of the transaction is 373 bytes. If instead metadata would be used, an extra output would need to be included, adding a maximum of 80 bytes.

The code is hosted at our Github repository[2] and the output of a full run of the Bitmessage version may be found in appendix A.

[2]Code repository: https://github.com/hillbom/smart-contracts

6

Discussion

The results clearly shows that Bitcoin suffers from issues in terms of scalability, in regards to its blockchain size, as well as slow transaction times. Today, the size of the Bitcoin blockchain is around 40 GB and is growing exponentially. If Bitcoin would gain wide adoption, this would quickly rise to unrealistic proportions. To make things worse, the block size would have to be increased in order to reduce bandwidth usage and to allow for more transactions to be able to get processed per minute. The increase of blockchain size is a much debated topic and one of its advocates is Bitcoin Foundation chief scientist Gavin Andresen[1].

The data collected for Colored Coins is a bit surprising. Adding metadata within transactions should of course lead to larger transaction sizes. We have two possible explanations for the contradicting result. First, colored coins is still a new concept and we believe the majority of these transactions are created for testing the protocols. Second, we believe the reason as to why an average bitcoin transaction size is large is that the majority of Bitcoin holders use the currency for trading on exchanges; the contents of exchange transactions usually include thousands of inputs and outputs.

By basing SPOEP on the Bitcoin protocol, it inherits the problems with scalability and transaction times. It should however leave a smaller footprint in the blockchain when compared to Colored Coins. All that gets added to the blockchain is a regular transaction, as the the contract details are kept separate.

By not relying on metadata, some assumptions had to be made regarding the identification of an asset and the notion of ownership. Instead of issuing assets within the blockchain, SPOEP interprets one of possibly many output addresses to be the new owner. As a result, it is important that the buyer does not tamper with the code that constructs the ownership transaction unless being aware of the consequences. For instance, adding multiple inputs and outputs will randomly assign the new ownership to one of the output addresses, which may or may not be a sought-after contract.

[1]Increasing the blocksize: http://www.coindesk.com/gavin-andresen-bitcoin-hard-fork/

SPOEP has a very specific area of application. The question is if storing data in the blockchain is completely unavoidable when designing other types of contracts. In most cases where goods or services are to be exchanged, we do believe that the double deposit escrow scheme described in section 2.2.2 may be applied. Where a person used to have something to gain by being dishonest, everyone now has an incentive to be honest. The obvious drawback is that a large deposition might be needed in order for this incentive to exist.

7

Conclusion

It is viable to form smart-contracts using cryptocurrency without the need of changing existing protocols. A standard practice of today is to accomplish this by adding metadata within the blockchains. As we have shown, this leads to worsening the scalability issues from which Bitcoin already suffers.

By comparing Bitcoin to Colored Coins implementations and Ethereum, it is clear that each approach has its own advantages and disadvantages in regard to implementing smart-contracts. Using the raw Bitcoin protocol (as we did in our SPOEP implementation) we do at least not worsen scalability or transaction time issues already present. It does however require a lot of effort when designing contracts. Colored coins make it easier to create assets, and thus creating smart-property, with the downside being a worsening of scalability. In terms of transaction times, Ethereum stands out as the average time is merely 12 seconds. By supporting multiple high level programming languages, designing any type of smart-contract should be possible without too much effort. But since contract code is stored in the blockchain, it is predicted to suffer from even worse scalability issues when compared to other cryptocurrencies.

If Ethereum developers manage to solve the scalability problem, Ethereum should be the ideal platform on which to create smart-contracts. As of today, we believe Bitcoin is still the best option.

7.1 Future work

- Bitcoin relies on a time consuming proof-of-work schema in order to prevent double spending. As an alternative, we recommend looking into cryptocurrencies based on the proof-of-stake schema. These feature significantly faster transaction times and eliminate the need for dedicated mining hardware. As a result, smart-contracts should execute much faster and the networks should be prone to become more decentralised.

- Proper certificate handling needs to be implemented in SPOEP. The manufacturer of the car should act as a central authority and issue a certificate for each car in order to be able to verify that a given public key corresponds to a specific car.
- When a stable release of Ethereum becomes available, a proper re-evaluation should be conducted in order to get an accurate measurement of the blockchain growth rate and its performance.
- When researching Bitmessage it became apparent that it may be used to implement decentralised markets via smart-contracts. Bitmessage has a feature called 'channels', which we believe could be combined with the Double Deposit Escrow contract to create trust-less decentralised markets.

Bibliography

[1] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system [2008].
URL: *https://bitcoin.org/bitcoin.pdf*

[2] Dai, W. B-money [1998]. Retrieved: 29 April 2015.
URL: *http://www.weidai.com/bmoney.txt*

[3] Coinmarketcap.com. Crypto-currency market capitalizations [2015]. Retrieved: 15 January 2015.
URL: *http://www.coinmarketcap.com*

[4] Vasink, P. Blackcoin's proof-of-stake protocol v2 [2014]. Retrieved: 04 May 2015.
URL: *http://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf*

[5] Houy, N. It will cost you nothing to 'kill' a proof-of-stake crypto-currency [2014].
URL: *http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2393940*

[6] Thomas, S. and Schwartz, E. Smart oracles: A simple, powerful approach to smart contracts [2014]. Version: 17 July 2014.
URL: *http://github.com/codius/codius/wiki/Smart-Oracles:-A-Simple,-Powerful-Approach-to-Smart-Contracts*

[7] Szabo, N. Smart contracts: Formalizing and securing relationships on public networks. *First Monday* [1997] vol. 2(9).
URL: *http://dx.doi.org/10.5210/fm.v2i9.548*

[8] Bitcoin.it. Smart property [2014]. Retrieved: 29 April 2015.
URL: *https://en.bitcoin.it/wiki/Smart_Property*

[9] Zimbeck, D. Two party double deposit trustless escrow in cryptographic networks and bitcoin [2014]. Retrieved: 04 May 2015.
URL: *http://blackhalo.info/wp-content/uploads/2014/06/whitepaper_twosided.pdf*

[10] Florian Mendel, Norbert Pramstaller, C.R. and Rijmen, V. *On the Collision Resistance of RIPEMD-160.* Tech. rep., Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria [2006].
URL: *https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=17675*

[11] Antonopoulos, A.M. *Mastering Bitcoin.* O'Reilly Media [2014]. Chapter 7.7: Merkle Trees.
URL: *http://chimera.labs.oreilly.com/books/1234000001802*

[12] Juan A Garay, A.K. and Leonardos, N. The bitcoin backbone protocol: Analysis and applications [2014].
URL: *https://eprint.iacr.org/2014/765.pdf*

[13] Bitcoin.it. Block chain [2014]. Retrieved: 07 July 2015.
URL: *https://en.bitcoin.it/wiki/Block_chain*

[14] Bitcoin.it. Double-spending [2015]. Retrieved: 07 July 2015.
URL: *https://en.bitcoin.it/wiki/Double-spending*

[15] Warren, J. Bitmessage: A peer-to-peer message authentication and delivery system [2012]. Retrieved: 29 April 2015.
URL: *https://bitmessage.org/bitmessage.pdf*

[16] Bitmessage. Wiki: Public key to bitmessage address [2013]. Retrieved: 23 June 2015.
URL: *https://bitmessage.org/wiki/Public_key_to_bitmessage_address*

[17] Bitmessage. Wiki: Proof of work [2014]. Retrieved: 9 July 2015.
URL: *https://bitmessage.org/wiki/POW*

[18] Buterin, V. Ethereum: A next-generation cryptocurrency and decentralized application platform [2014]. Retrieved: 04 May 2015.
URL: *https://bitcoinmagazine.com/9671*

[19] Buterin, V. A next generation smart contract & decentralized application platform [2014].
URL: *http://www.ethereum.org/pdfs/EthereumWhitePaper.pdf*

[20] Kallal, J.S. Linux programmer's manual: Random [2015]. Retrieved: 30 August 2015.
URL: *http://man7.org/linux/man-pages/man4/random.4.html*

[21] MS, A. Elliptic curve cryptography - an implementation guide [2007].
URL: *http://www.infosecwriters.com/text_resources/pdf/Elliptic_Curve_AnnopMS.pdf*

[22] Blockchain.info. Bitcoin average transaction confirmation time [2015]. Retrieved: 03 July 2015.
URL: *https://blockchain.info/charts/avg-confirmation-time?timespan=2year*

[23] Blockchain.info. Bitcoin blockchain size [2015]. Retrieved: 03 July 2015.
URL: *https://blockchain.info/charts/blocks-size?timespan=all*

[24] Etherchain.org. Ethereum - basic statistics [2015]. Retrieved: 14 July 2015.
URL: *https://etherchain.org/statistics/basic*

A

SPOEP-trading via Bitmessage

A.1 Example run

The following subsection shows the output given from a run of the SPOEP-protocol using an illustrative contract specification.

A.1.1 Contract specifications

```
[seller_data]
price = 7.97061008298
bm    = BM-2cXM6hE8D7XwgGyZEouLDDuXcKPGqWJpbc

[car_data]
car_id    = s712
car_pk    = mnLCAYXccLj2Y7Rxhc4U8u6cUwdYAqYvxg
car_model = Forrrd
car_owner = mvZgwRC2NtsKYCnxnJr7E8DCkZWuNfhoDx

[man_cert]
pk = forrrd
```

A.1.2 Program output

```
Car: Entered new state
waitForBuyRequest

Car: Waiting for buyer_info
buyer_info

Seller: Entered new state
waitForNonce

Buyer: Entered new state
sendBuyRequest

Buyer: Sending buy_request to Seller
"buy_request": {
  "nonce": "197293679455728155344879880342230494756",
  "new_owner_key": "mvuqB21gj2LYMujksYpYZKXMJHLKugL9iD",
  "car_id": "s712"
}

Buyer: Waiting for trade_data
trade_data

Seller: Waiting for buy_request
buy_request

Seller: Entered new state
hasBuyRequest

Seller: Sending buyer_info to Car
"buyer_info": {
  "seller_sign": "IIfM245zcus4mhwQD2AFp3dECWnjRULM4lqOmo8o
                  X7KRPm9fNpFzf219cUDsnsaDUTZJ5ZhVHpmK4oDr
                  pFZTKls=",
  "buyer_info": {
    "nonce": "197293679455728155344879880342230494756",
    "new_owner_key": "mvuqB21gj2LYMujksYpYZKXMJHLKugL9iD"
  }
}

Seller: Waiting for car_data
car_data

Car: Entered new state
hasBuyerInfo

Car: Signing car_data
"car_sign": "HOZMw66eIiPYRfbTqWlWF2JtmSvnJKQ7tR+jbr/r34Sv
             dysBwpWd8V0MVEVVXs9nc5B8vU6bh1QHqJXObBGpOOk=",

Car: Sending car_data to Seller
"car_data": {
  "car_sign": "HOZMw66eIiPYRfbTqWlWF2JtmSvnJKQ7tR+jbr/r34Sv
               dysBwpWd8V0MVEVVXs9nc5B8vU6bh1QHqJXObBGpOOk=",
  "signed_data": {
    "nonce": "197293679455728155344879880342230494756",
    "car_data": {
      "last_tx": "3836bb7925226dc6c11c60de5e1de860e5fe4b68
                  7db11f6768b223679ba401d2",
      "previous_owner": "mvTNu3U71efmpdN1x1ZPSu1he3Dkv1toDt",
      "car_model": "Forrrd",
      "man_cert": "forrrd",
      "car_pk": "mnLCAYXccLj2Y7Rxhc4U8u6cUwdYAqYvxg",
      "car_owner": "mvZgwRC2NtsKYCnxnJr7E8DCkZWuNfhoDx",
      "car_id": "s712"
    },
    "new_owner_key": "mvuqB21gj2LYMujksYpYZKXMJHLKugL9iD"
  }
}
```

```
Car: Entered new state
sentCarData

Car: Waiting for change_owner
change_owner

Seller: Car signature valid
Signature valid

Seller: Entered new state
hasCarData

Seller: Sending trade_data to Buyer
"trade_data": "car_data"

Seller: Entered new state
sentTradeData

Seller: Waiting for partial_tx
partial_tx

Buyer: Entered new state
hasTradeData

Buyer: Received last transaction
"last_tx": "3836bb7925226dc6c11c60de5e1de860e5fe4b687db11f
           6768b223679ba401d2"

Buyer: Found output in lastTx to spend from
0

Buyer: Change amount to Seller
9.99

Buyer: Creating raw transaction from
{
  "inputs": [{
      "vout": 1,
      "txid": "db007ae36daef5f65993e930c5fcf068b92cee22a0f
               9241f30800c4f803d63a3"
    },
    {
      "vout": 0,
      "txid": "3836bb7925226dc6c11c60de5e1de860e5fe4b687db
               11f6768b223679ba401d2"
    }],
  "to": {
    "mvZgwRC2NtsKYCnxnJr7E8DCkZWuNfhoDx": 17.96061008298,
    "mvuqB21gj2LYMujksYpYZKXMJHLKugL9iD": 0.01
  }
}

Buyer: Created raw_tx
0100000002a3633d804f0c80301f24f9a022ee2cb968f0fcc530e99359
f6f5ae6de37a00db0100000000ffffffffd201a49b6723b268671fb17d
684bfee560e81d5ede601cc1c66d222579bb36380000000000ffffffff
0250b70d6b000000001976a914a50e50f70a796c461bacbb5a8bab4d5d
7088e94288ac40420f00000000001976a914a8dd826230d10f6ca9fdf1
6c9db8788fe0707a9588ac00000000

Buyer: Sending partial_tx to Seller
"partial_tx": {
  "hex": "0100000002a3633d804f0c80301f24f9a022ee2cb968f0fc
          c530e99359f6f5ae6de37a00db010000006b483045022100
          e8804d004d36f610ea311c3555df47c1dcd44a4a44f0aacb
          9430168f2b07c6b9022049f5be798b016c1e18ebc589f687
          70c8a1d061dee65de33a909b701d9f784268012103ea8965
          807d3e6e3e2c10e06192c45bc85d734d9b8d00e44a490a9b
          29eea635f7ffffffffd201a49b6723b268671fb17d684bfe
          e560e81d5ede601cc1c66d222579bb363800000000000ffff
          ffff0250b70d6b000000001976a914a50e50f70a796c461b
          acbb5a8bab4d5d7088e94288ac40420f00000000001976a9
          14a8dd826230d10f6ca9fdf16c9db8788fe0707a9588ac00
          000000",
  "complete": false
}

Buyer: Entered new state
sentPartial

Buyer: Waiting for signed_transaction
signed_transaction

Seller: Last amount
10.0

Seller: Change amount
9.99

Seller: Expected amount to receive
17.96061008

Seller: Partial transaction is valid
{
  "scriptPubKey": {
    "reqSigs": 1,
    "hex": "76a914a50e50f70a796c461bacbb5a8bab4d5d7088e942
            88ac",
    "addresses": ["mvZgwRC2NtsKYCnxnJr7E8DCkZWuNfhoDx"],
    "asm": "OP_DUP OP_HASH160
            a50e50f70a796c461bacbb5a8bab4d5d7088e942
            OP_EQUALVERIFY OP_CHECKSIG",
    "type": "pubkeyhash"
  },
  "value": Decimal("17.96061008"),
  "n": 0
}

Seller: Entered new state
hasPartialTx

Seller: Decoded and verified partial raw transaction
{
  "locktime": 0,
  "version": 1,
  "vin": [{
      "sequence": 4294967295,
      "scriptSig": {
        "hex": "483045022100e8804d004d36f610ea311c3555df47
                c1dcd44a4a44f0aacb9430168f2b07c6b9022049f5
                be798b016c1e18ebc589f68770c8a1d061dee65de3
                3a909b701d9f784268012103ea8965807d3e6e3e2c
                10e06192c45bc85d734d9b8d00e44a490a9b29eea6
                35f7",
        "asm": "3045022100e8804d004d36f610ea311c3555df47c1
                dcd44a4a44f0aacb9430168f2b07c6b9022049f5be
                798b016c1e18ebc589f68770c8a1d061dee65de33a
                909b701d9f7842680103ea8965807d3e6e3e2c10e0
                6192c45bc85d734d9b8d00e44a490a9b29eea635f7"
      },
      "vout": 1,
      "txid": "db007ae36daef5f65993e930c5fcf068b92cee22a0f
               9241f30800c4f803d63a3"
    },
    {
      "sequence": 4294967295,
      "scriptSig": {
        "hex": "",
        "asm": ""
      },
      "vout": 0,
      "txid": "3836bb7925226dc6c11c60de5e1de860e5fe4b687db
               11f6768b223679ba401d2"
    }],
  "vout": [{
      "scriptPubKey": {
        "reqSigs": 1,
        "hex": "76a914a50e50f70a796c461bacbb5a8bab4d5d7088
                e94288ac",
        "addresses": ["mvZgwRC2NtsKYCnxnJr7E8DCkZWuNfhoDx"],
        "asm": "OP_DUP OP_HASH160
                a50e50f70a796c461bacbb5a8bab4d5d7088e942
                OP_EQUALVERIFY OP_CHECKSIG",
        "type": "pubkeyhash"
      },
      "value": Decimal("17.96061008"),
      "n": 0
    },
    {
      "scriptPubKey": {
        "reqSigs": 1,
        "hex": "76a914a8dd826230d10f6ca9fdf16c9db8788fe070
                7a9588ac",
        "addresses": ["mvuqB21gj2LYMujksYpYZKXMJHLKugL9iD"],
        "asm": "OP_DUP OP_HASH160
```

```
                a8dd826230d10f6ca9fdf16c9db8788fe0707a95
                OP_EQUALVERIFY OP_CHECKSIG",
        "type": "pubkeyhash"
      },
      "value": Decimal("0.01000000"),
      "n": 1
    }],
  "txid": "1478e825e72fb5e1b08dec90a092abb8d6f4044b52c98a0
           a10311b7a34fe36ce"
}

Seller: Signing partial transaction
{
  "hex": "0100000002a3633d804f0c80301f24f9a022ee2cb968f0fc
          c530e99359f6f5ae6de37a00db010000006b483045022100
          e8804d004d36f610ea311c3555df47c1dcd44a4a44f0aacb
          9430168f2b07c6b9022049f5be798b016c1e18ebc589f687
          70c8a1d061dee65de33a909b701d9f784268012103ea8965
          807d3e6e3e2c10e06192c45bc85d734d9b8d00e44a490a9b
          29eea635f7ffffffffd201a49b6723b268671fb17d684bfe
          e560e81d5ede601cc1c66d222579bb3638000000006a4730
          4402203c85a71b84d87c65c27b1e282c3f793f57c786f7c9
          716ad54fa5ca20313204c102206603a210ca859d5694292d
          08090f8a5295e4c392d84f30478f4bd7b5aac5ce9e012102
          e4d1eab90dba0e79b1b1ad379e2dd1da0f90721881295eb3
          14876d87139a2be2ffffffff0250b70d6b000000001976a9
          14a50e50f70a796c461bacbb5a8bab4d5d7088e94288ac40
          420f00000000001976a914a8dd826230d10f6ca9fdf16c9d
          b8788fe0707a9588ac00000000",
  "complete": True
}

Seller: Broadcasting transaction
"txid": "f4bd3ad455acb1ea919915380e94137f8e0117c872df848b0
        23621818b7fc58e"

Seller: Sending signed_transaction to Buyer
"signed_transaction": {
  "txid": "f4bd3ad455acb1ea919915380e94137f8e0117c872df848
           b023621818b7fc58e"
  "carBM": "BM-2cWJYYT8DqWGdNXKKif7zQ21tpXLC9VMET"
}

Buyer: Entered new state
hasTranscation

Buyer: Sending change_owner to Car
"txid": "f4bd3ad455acb1ea919915380e94137f8e0117c872df848b0
        23621818b7fc58e"

Car: Verifing transaction in blockchain
Exploring blocks...

Car: Transaction found in block 417
[
  "a61a002de09e9a4fb660ccaa17687e57
   d05bba742b60d0d0495960e59cd5d6c8",
  "db007ae36daef5f65993e930c5fcf068
   b92cee22a0f9241f30800c4f803d63a3",
  "f4bd3ad455acb1ea919915380e94137f
   8e0117c872df848b023621818b7fc58e"
]

Car: Entered new state
changeToNewOwner

Car: Changed owner
"car_owner": "mvuqB21gj2LYMujksYpYZKXMJHLKugL9iD"

Car: Entered new state
hasNewOwner
```

A.1.3 Final transaction

```
{
  "txid" : "f4bd3ad455acb1ea919915380e94137f8e0117c872df84
            8b023621818b7fc58e",
  "version" : 1,
  "locktime" : 0,
  "vin" : [{
       "txid" : "db007ae36daef5f65993e930c5fcf068b92cee22a
                 0f9241f30800c4f803d63a3",
       "vout" : 1,
       "scriptSig" : {
        "asm" : "3045022100e8804d004d36f610ea311c3555df47c
                 1dcd44a4a44f0aacb9430168f2b07c6b9022049f5
                 be798b016c1e18ebc589f68770c8a1d061dee65de
                 33a909b701d9f78426801 03ea8965807d3e6e3e2
                 c10e06192c45bc85d734d9b8d00e44a490a9b29ee
                 a635f7",
        "hex" : "483045022100e8804d004d36f610ea311c3555df4
                 7c1dcd44a4a44f0aacb9430168f2b07c6b9022049
                 f5be798b016c1e18ebc589f68770c8a1d061dee65
                 de33a909b701d9f784268012103ea8965807d3e6e
                 3e2c10e06192c45bc85d734d9b8d00e44a490a9b2
                 9eea635f7"
       },
       "sequence" : 4294967295
      }, {
       "txid" : "3836bb7925226dc6c11c60de5e1de860e5fe4b687
                 db11f6768b223679ba401d2",
       "vout" : 0,
       "scriptSig" : {
        "asm" : "304402203c85a71b84d87c65c27b1e282c3f793f5
                 7c786f7c9716ad54fa5ca20313204c102206603a2
                 10ca859d5694292d08090f8a5295e4c392d84f304
                 78f4bd7b5aac5ce9e01 02e4d1eab90dba0e79b1b
                 1ad379e2dd1da0f90721881295eb314876d87139a
                 2be2",
        "hex" : "47304402203c85a71b84d87c65c27b1e282c3f793
                 f57c786f7c9716ad54fa5ca20313204c102206603
                 a210ca859d5694292d08090f8a5295e4c392d84f3
                 0478f4bd7b5aac5ce9e012102e4d1eab90dba0e79
                 b1b1ad379e2dd1da0f90721881295eb314876d871
                 39a2be2"
       },
       "sequence" : 4294967295
      }
  ],
  "vout" : [{
       "value" : 17.96061008,
       "n" : 0,
       "scriptPubKey" : {
        "asm" : "OP_DUP OP_HASH160
                 a50e50f70a796c461bacbb5a8bab4d5d7088e942
                 OP_EQUALVERIFY OP_CHECKSIG",
        "hex" : "76a914a50e50f70a796c461bacbb5a8bab4d5d708
                 8e94288ac",
        "reqSigs" : 1,
        "type" : "pubkeyhash",
        "addresses" : [
         "mvZgwRC2NtsKYCnxnJr7E8DCkZWuNfhoDx"
        ]
       }
      }, {
       "value" : 0.01000000,
       "n" : 1,
       "scriptPubKey" : {
        "asm" : "OP_DUP OP_HASH160
                 a8dd826230d10f6ca9fdf16c9db8788fe0707a95
                 OP_EQUALVERIFY OP_CHECKSIG",
        "hex" : "76a914a8dd826230d10f6ca9fdf16c9db8788fe07
                 07a9588ac",
        "reqSigs" : 1,
        "type" : "pubkeyhash",
        "addresses" : [
         "mvuqB21gj2LYMujksYpYZKXMJHLKugL9iD"
        ]
       }
      }]
}
```