UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Oliver-Matis Lill

Bitcoin Scaling with Specialization

Bachelor's Thesis (9 ECTS)

Supervisor: Assoc. Prof. Vitaly Skachek

Tartu 2018

Bitcoin scaling with specialization

Abstract:
Bitcoin is a young cryptocurrency, which has seen a rapid growth of popularity in the recent years. If the current trend continues then the scaling issues will become critical for the future use of Bitcoin. In this thesis, we propose a new architecture for Bitcoin-like cryptocurrency, which efficiently facilitates Bitcoin scaling. Our scheme relies on a new type of node called validator - a node that validates transactions and detects double spending. We propose a system based on binary trees and pseudorandom number generators that distributes the transactions among validators such that the individual workload is small but the whole system is properly validated. We also provide mechanisms for validators and other nodes to cooperate in order to add new blocks and ensure the trustworthiness of the stored data. Finally, we study practical aspects of the proposed system, analyze various attack scenarios that may occur, and propose countermeasures.

Keywords:
Bitcoin, scaling, Merkle trees, randomization, hashing

CERCS: P170 Computer science, numerical analysis, systems, control

Bitcoini skaleerimine spetsialiseerumise abil

Lühikokkuvõte:
Bitcoin on noor krüptoraha mille populaarsus on viimaste aastatega hüppeliselt kasvanud. Praeguste trendide jätkumisel ootavad Bitcoini ees tõsised skaleerimisprobleemid. Selles uurimistöös me esitame uue arhitektuuri Bitcoini suguste krüptorahade jaoks mis võimaldab edukalt Bitcoini skaleerida. Meie skeem toetub uut tüüpi võrgutippudele, nimelt kontrollijatele, mis kinnitavad tehinguid ja tuvastavad raha topeltkulutamist. Me esitame süsteemi mis toetub kahendpuudele ja pseudojuhuarvugeneraatoritele ning võimaldab tehinguid jaotada kontollijate vahel ära nii et nende individuaalne töökoormus on väike aga kogu süsteem saab lõpuks korralikult läbi kontrollitud. Me esitame ka mehhanisme millega kontrollijad ja teised tipud saavad koostööd teha et uusi blokke esitada ning kinnitada andmete usaldusväärsust. Viimasena me analüüsime meie süsteemiga juurde tulevaid praktilisi kaalutlusi ja rünnakutsenaariumeid ning pakume viise kuidas neid lahendada.

Võtmesõnad: Bitcoin, skaleerimine, Merkle puud, juhuslikustamine, räsimine

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

Currently, Bitcoin is still a very young cryptocurrency with low adoption compared to traditional currencies. As of today, the total size of the underlying blockchain that contains all the transactions is 157 GiB [3]. Additionally the peak load the currency has had to handle has been about $5 \cdot 10^5$ transactions per day [4]. This means that its still fairly manageable for most personal computers to go through the entire blockchain and check the validity of every single incoming transaction. The fact that everyone can see what is going on makes it much easier to trust the system.

In recent years however the size of the blockchain has seen exponential growth, increasing from 26 GiB in 2015, to 50 GiB in 2016, 90 GiB in 2017 and 140 GiB in the beginning of 2018 [3]. Additionally if it reaches circulation volume comparable with traditional currencies, then it will grow far larger: in 2016 the EU processed a total of 122 billion non-cash transactions and about 1.4 trillion in total [2]. This corresponds to about $3.3 \cdot 10^8$ non-cash transactions per day. Given that the average transaction size is greater than 500 B, this corresponds to a blockchain growth rate of about 165 GB per day, which would quickly make transaction validation infeasible for most personal computers. Additionally, the global economy is constantly growing, and the amount of non-cash transactions is growing even faster, which means that if Bitcoin is successful, then the Bitcoin network will soon become incomprehensible for individual computers and compute nodes. The question is, if it is possible to deal with it while still keeping the network decentralized and trustworthy.

There have been various attempts to improve scalability of Bitcoin. For example, it is suggested to use coding and secret sharing in [9]. The authors of [6] propose to use microblocks to speed up block generation. In [11] a secure system for decreasing latency of block generation is proposed. While they all do propose ways to alleviate or solve some aspects of the scaling issue, transaction validation still remains the biggest issue facing Bitcoin in case of higher adoption.

In this thesis we analyze Bitcoin itself and the issue of Bitcoin scaling. Bitcoin allows a great degree of flexibility in how nodes interact with it and we utilize this flexibility to propose a system of node behavior that facilitates the scaling of Bitcoin. Currently Bitcoin has lightweight nodes that can validate the existence of any transaction and full nodes that validate all transactions and mine new blocks. If the volume of transactions grows large, then the high workload will dissuade new nodes from running full nodes, resulting in a more centralized Bitcoin system. We present a way to distribute out all the work of a full nodes so new nodes can still contribute to the system while not being overwhelmed by an unmanageable workload.

The proposed system relies on the fact that once a node detects an invalid transaction or a conflict between transactions (like double-spending), it can easily inform other nodes of the issue by sending information about the involved transactions. All other nodes can easily verify the existence of involved transactions in $O(\log n)$ using Merkle branches

and then verify the existence of error by analyzing the transactions. This means that verifying reported errors is easy, but combing transactions to find those errors is hard. Our system creates a new type of node called validator, that specializes in combing transactions for those errors. We also present a system of binary trees and pseudorandom number generators that divides up the work of validation and allows any validator to take on the amount of work it can handle. Additionally, we present a way for miners to outsource transaction validation to validators which permits mining to retain a low barrier of entry as well. Finally we analyze the new practical considerations and attack scenarios that arise from the increased complexity our system brings while providing solutions and countermeasures to them.

In Section 2 we will present the necessary background knowledge and the model our system operates in. In Section 3 we will present out proposed system of node behavior, describing the types of nodes, how they cooperate together and how they can divide up the work between them. In Section 4 we will analyze the practical considerations of our system and finally in Section 5 we will present a summary of our results.

2 Background

In this section we cover the setting of this thesis, presenting the necessary preliminary knowledge and the model we are dealing with.

2.1 Preliminaries

Here we cover the main concepts used in the paper

2.1.1 Hash Functions

A hash function is simply a function $h : \mathcal{M} \to \mathcal{H}$ from the set of messages $\mathcal{M}$ to the set of hash values $\mathcal{H}$. However a good hash function generally has the following properties:

1. **Computational ease:** For any $x \in \mathcal{M}$, it is easy to compute $h(x)$
2. **Pre-image resistance:** Given some $y \in \mathcal{H}$ it is practically infeasible to find $x \in \mathcal{M}$ such that $h(x) = y$
3. **Collision resistance:** It is practically infeasible to find a pair $x_1 \in \mathcal{M}$ and $x_2 \in \mathcal{M}$ such that $x_1 \neq x_2$ but $h(x_1) = h(x_2)$

Analysis of specific hash functions is beyond the scope of this thesis, but Bitcoin relies on SHA-256[14].

2.1.2 Public-Key Cryptography

A public key encryption system is characterized by an encryption function $\mathcal{E} : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$ and decryption function $\mathcal{D} : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$. Here $\mathcal{M}$ is the set of messages, $\mathcal{C}$ is the set of encrypted messages and $\mathcal{K}$ is a set of keys. Generally we generate a pair of keys k_1 and k_2, such that for any message $m \in \mathcal{M}$ we can:

1. Encrypt it into c with $c \leftarrow \mathcal{E}(m, k_1)$
2. Decrypt c to get back m with $m \leftarrow \mathcal{D}(c, k_2)$

The system is generally designed such that it is infeasible to deduce k_2 from k_1 and vice-versa. This means that if Alice makes k_2 public but keeps k_1 private, then only she can encrypt messages with this pair, but everyone can read her messages and confirm that they were indeed sent by her. Conversely, if she makes k_1 public instead, then everyone can encrypt messages for her, but only she can read them. The specifics of such systems are out of scope, but a good and educational example for such systems would be the RSA cryptosystem[13].

2.1.3 Merkle Tree

A Merkele tree is a hash tree in which each node stores the hash of its children, and the leaves store the objects we want to hash. It is useful, because one needs only the root hash to be able to validate any of the objects hashed with the tree.

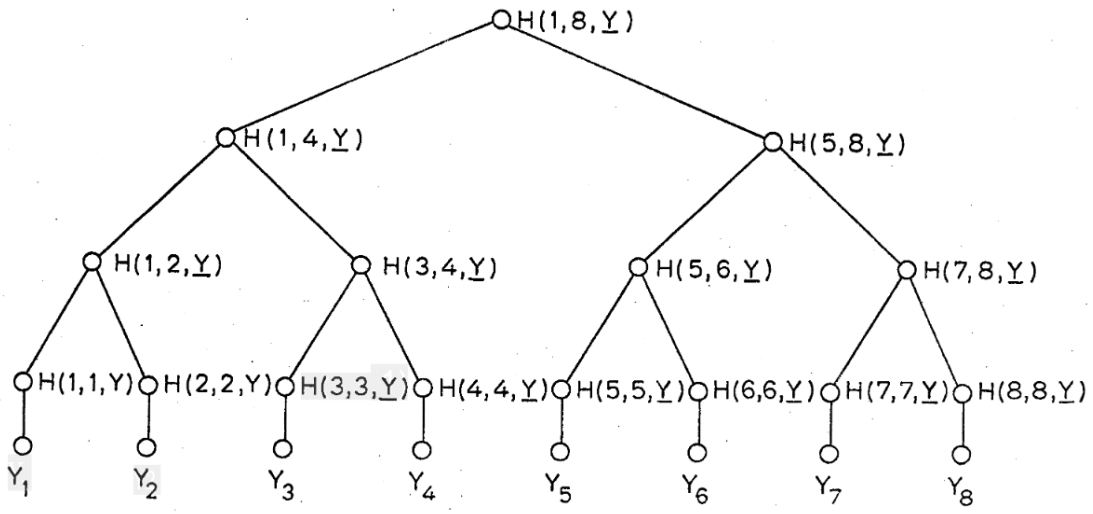

Figure 1. Merkle tree for hashing objects $Y_1, Y_2, \ldots, Y_8$ [7]

In Figure 1 given some hashing function h, we have $H(i, j, \underline{Y})$ defined as:

1. $H(i, i, \underline{Y}) = h(Y_i)$
2. $H(i, j, \underline{Y}) = h(H(i, k, \underline{Y}), H(k + 1, j, \underline{Y}))$
   where $k = (i + j)/2$

Thus in Figure 1 if Bob knows the root hash $H(1, 8, \underline{Y})$, then Alice can prove the existence of Y_5 in the tree by sending Y_5, $H(6, 6, \underline{Y})$, $H(7, 8, \underline{Y})$, $H(1, 4, \underline{Y})$, and Bob can easily verify it with the following steps:

1. Calculate $H(5, 5, \underline{Y})$ using Y_5
2. Calculate $H(5, 6, \underline{Y})$ using $H(5, 5, \underline{Y})$ and $H(6, 6, \underline{Y})$
3. Calculate $H(5, 8, \underline{Y})$ using $H(5, 6, \underline{Y})$ and $H(7, 8, \underline{Y})$
4. Calculate $H(1, 8, \underline{Y})$ using $H(1, 4, \underline{Y})$ and $H(5, 8, \underline{Y})$
5. Verify that the calculated $H(1, 8, \underline{Y})$ matches the root hash he has

If the hash function is decent then it will be practically impossible for Alice to generate different transactions that would pass this validation. This means that if there are a large number of transactions, then Bob will have to store only a single hash, the Merkle root, and Alice can easily prove the existence of any transactions by sending the $O(\log n)$ nodes corresponding to the Merkle branch of that transaction.

In the Bitcoin system (and in most other scenarios), Merkle tree is a perfect binary tree. It's possible relax the structure constraints and even allow non-binary variants, but we will not consider such use cases in this thesis since they would add unnecessary complexity to the Bitcoin system.

2.1.4 Bitcoin

The Bitcoin system facilitates the trading of units of currency called "Bitcoins". In this thesis, we consider a simplified model of a Bitcoin transaction. In the Bitcoin system, each transaction has a set of inputs that use up Bitcoins, and a set of outputs that generate new Bitcoins. The total amount of Bitcoins generated in outputs must be no greater than the amount used up in inputs (the difference will be used as a transaction fee). A high-level example of transactions is provided in Figure 2. Each transaction output is basically a check of Bitcoins owned by a specific person. To ensure that only the owner of an output can use it, the output will publish the public key of the new owner and that output can be spent only when the owner uses his private key to sign the corresponding input in some transaction.

This is an elegant system for protecting the owners, but there is still the issue of double-spending. An owner could create two transactions that spend the same transaction output. The question is how do we determine which transaction is valid and how do we prevent the owner from spending the same output repeatedly. The Bitcoin currency resolves this with the blockchain system.

A blockchain serves as a public ledger that stores all the transactions. It consists of blocks that store transactions, connected together into a chain by having the next block contain a hash of the previous block as shown in Figure 4. To make the blocks lighter, the transactions of a block are hashed into a Merkle tree, and only its root is put into the block. The simplified contents of the block are given in Figure 3. Lightweight nodes can store only the block header. A second party can easily prove the existence of transaction to them by providing a Merkle branch to that transaction (as shown in Figure 4), and the node itself just has to verify that the branch hashes up to the root given in the header (this takes $O(\log n)$ steps).

The blockchain is completely decentralized, anyone can see it, share it and add new blocks to it. To prevent malicious entities from spamming the chain with new blocks, the hash of each block must satisfy a difficulty criteria (for example, have certain number of leading zeroes). A nonce is an arbitrary string attached to the block, whose sole purpose is to make the block hash satisfy the required criteria. In fact the whole process

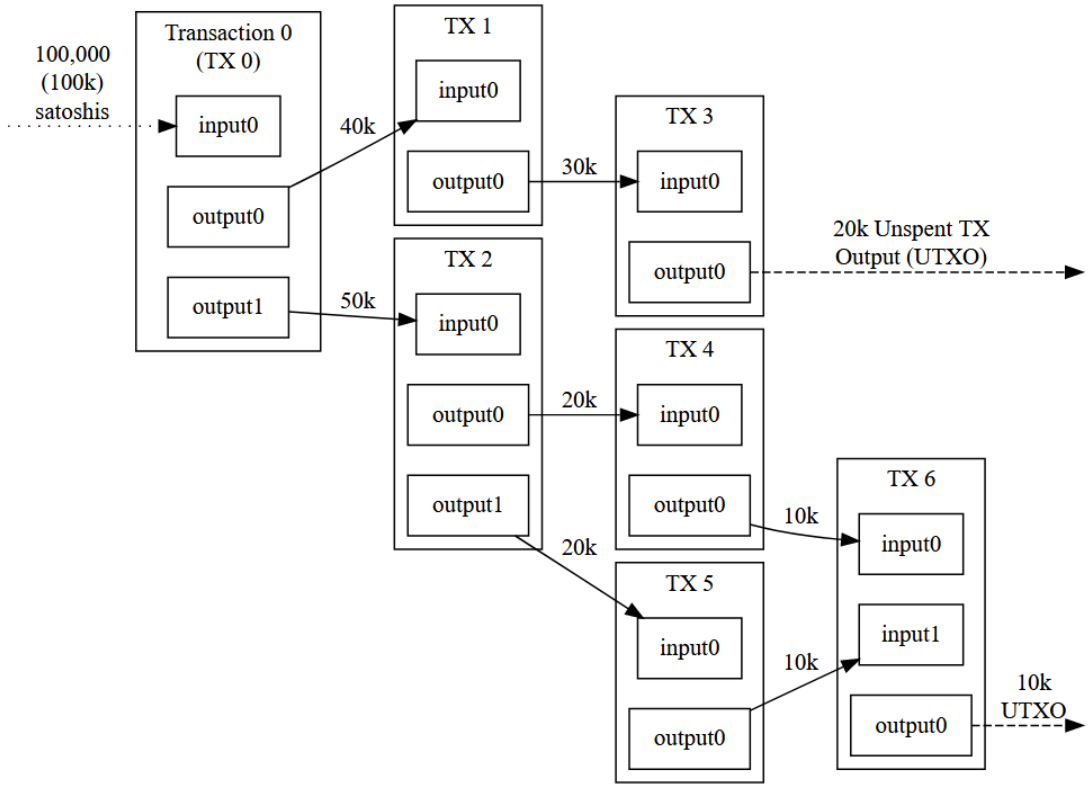

Figure 2. Bitcoin transactions and the connections between their inputs and outputs [5]

of "Bitcoin mining" is simply about finding an appropriate nonce. The difficulty is set so that the whole network generates blocks roughly once per 10 minutes, and is adjusted if the rate becomes too fast or too slow.

This Blockchain provides chronological ordering to the blocks. A block that contains a transaction output that was spent in some previous block of the chain is considered invalid and will be ignored by the network. The public ledger is considered to be represented by the longest valid chain. "Miners" are nodes that generate new blocks (by collecting transactions and computing nonce) and they add their block to the end of the longest chain they see. It is possible that two miners generate a block at the same time and that the longest chain will branch into two, but in that case the next miner will break the tie by adding its block to the end of one of them. This will result in the other block becoming "orphaned".

To make Bitcoin secure, there is an incentive for miners, namely they get a fixed reward of Bitcoins awarded to them for mined block, plus all the transaction fees from

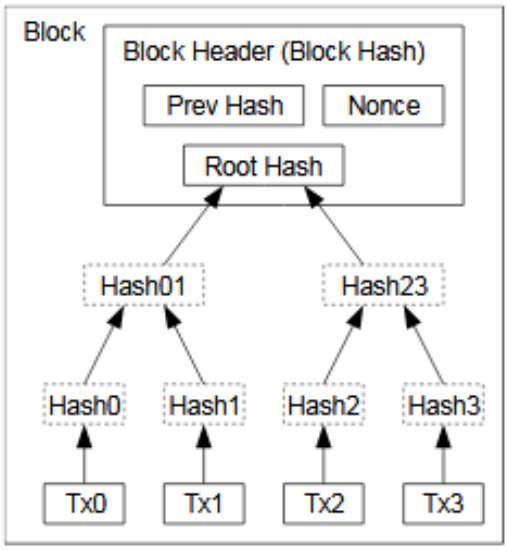

Figure 3. A single Bitcoin block with the Merkle tree of its transactions [8]

the transactions they collect. The reward has the form of a transaction with no outputs and it is contained inside the block they mined which provides an incentive to make sure their block is valid. The Bitcoin security is based on the assumption that most of the computational power is in the hands of honest miners. A dishonest miner cannot create invalid (for example double-spending) blocks, since these blocks will not be accepted by other nodes. They can however try reversing a past transaction by starting from its parent block and trying to build a new, longer chain that does not contain it. If the block of a transaction has enough blocks following it in the longest valid chain, then it will be effectively impossible to succeed with the attack. As the attackers build their malicious chain, the honest nodes will keep extending the longer honestly built chain and if honest nodes have more computational power, then the chances of the malicious chain becoming accepted is effectively nonexistent [8].

Currently a convenient aspect is that the blockchain is fairly small (about 157 GiB) and the transaction rate rather low (less than $5 \cdot 10^5$ per day). As such it is easy for most of the users to validate the whole blockchain. However if Bitcoin grows large, then that will no longer be the case. This thesis will analyze how to keep things decentralized and trustworthy in such a scenario. The possible ways Bitcoin could handle such surge of popularity is either by allowing for a higher rate of block generation, or by removing the limit on the total size of transactions a block can contain. We assume the latter approach, since the former will produce similar incomprehensibility while increasing the risk of "chain races" and the number of orphaned blocks.

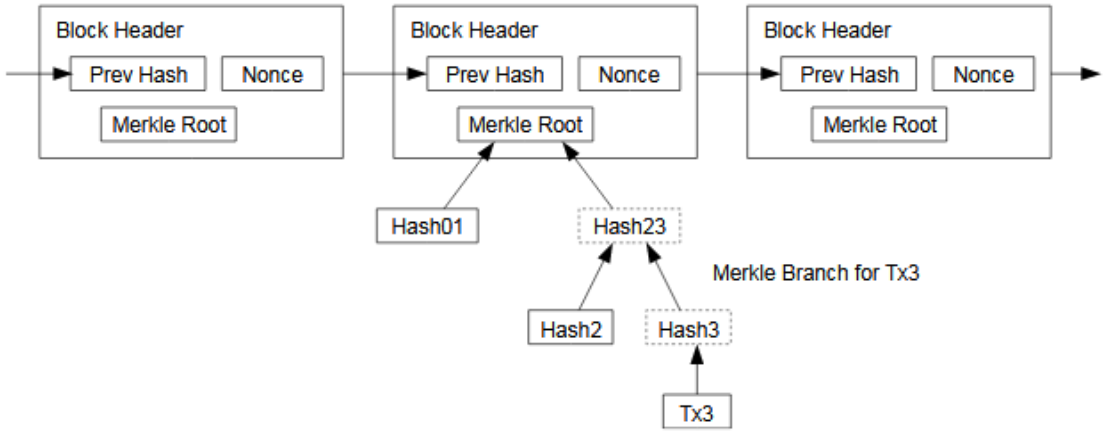

Figure 4. The blockchain with individual blocks [8]

2.2 Model

In this thesis, the model assumes that we have the aforementioned Bitcoin system (with all block size limits lifted) running on a network of nodes in an "open internet" (i.e. every pair of nodes can communicate with each other). The network will be characterized by the following parameters:

- R - the number of transactions per day
- N - the number of transactions that have happened until now

We are looking at a single day and we assume that N is much larger than R, so that day won't make a noticeable difference in the total number of transactions. We are looking for ways to efficiently divide up the N stored transactions and the processing of the R new transactions per day.

3 Specialization System

The proposed solution is as follows: we divide up the work among the nodes. We will describe an approach that keeps decentralization while still keeping a great degree of trustworthiness. In the proposed system, the nodes are divided into three types:

1. **Observers** - Lightweight nodes whose main task is spreading information across the network
2. **Validators** - Nodes that validate blocks and transactions and will report any detected errors
3. **Miners** - Nodes that "mine blocks", same as in the existing Bitcoin system

The main and most interesting addition here is the validator type of nodes. Currently Bitcoin miners also act as validators, since validation is still simple. In our model, the specialized validators were added in order to deal with the significantly increased difficulty of validation.

3.1 Node Subtypes

In this section we describe in detail the behavior of the aforementioned node types.

3.1.1 Observers

The observers are lightweight nodes that anyone can set up with little effort and that anyone can use to check the existence of transactions, and the state of the blockchain. Their benefit for the network is that they will validate and propagate critical information across the network. An observer will store only the block headers. The current number of blocks is about $5 \cdot 10^5$ [10]. Considering that a block header takes 80 bytes, this corresponds to total storage cost of about 40 MB, with a growth rate of 4.2 MB per year, which is easily manageable by any modern computer [8].

Being an observer is useful, because it allows the node to validate transactions. The other party simply has to send the block and Merkle branch with $O(\log N)$ hashes corresponding to the transaction (as shown in Figure 5) and the observer just has to check that he has the hashes are consistent in the structure (the child hashes together hash to the parent hash) and that the root hash matches the one in the block header. This also allows the observer to see the existence of outputs generated by that transaction, but not whether these outputs were spent later. For that the observer would need to ask for help from some validator.

The main benefit to the network from observers is that they validate and spread critical information. They can trivially validate that the block has a correct nonce by

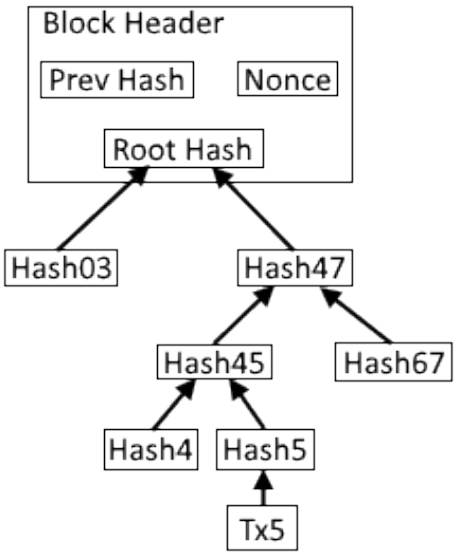

Figure 5. The block and Merkle branch for validating transaction Tx5

hashing the block. They can not validate that all transactions inside the block are correct (since they do not store transactions), but they can validate the presence of any reported error. With the following procedure the whole network can be informed of any erroneous blocks:

1. Someone who detects an error transaction or double-spending will send the involved transactions and their Merkle branches to many observers
2. The observers will confirm the existence of those transactions by checking that the branch is valid
3. If an error indeed appears, then the observers inform other nodes in the network

Validators will check that individual transactions are valid, report errors to the observers and the observers will spread the information across the network.

3.1.2 Validators

Validators are the new type of nodes, that store and validate individual transactions. In the proposed model, the number of transactions is possibly too large for a single node to validate all of them, so we propose a system to divide divide the work into smaller chunks. The proposed system relies on a pseudorandom number generation. We use a pseudorandom function $g : U \to \{0, 1\}$ (where U is any set of inputs for which g is defined) with the following properties:

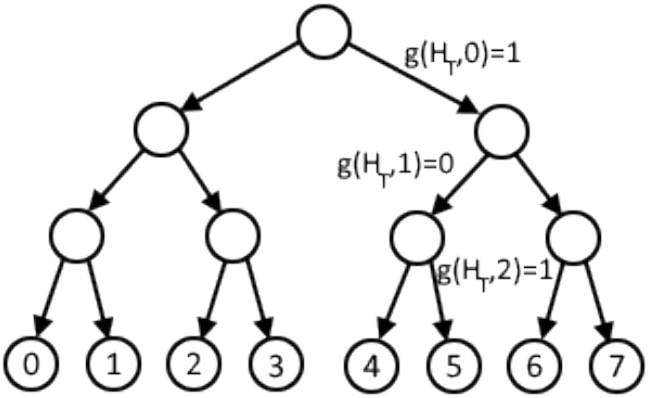

Figure 6. A distribution tree that assigns a transaction with hash H_T to leaf 5

- **Determinism** - For some input x, the output of $g(x)$ will always be the same
- **Uniform Distribution** - For random input X, we have $P(g(X) = 1) = 0.5$

The function g can be naturally extended to take tuples as inputs (we simply denote it as $g(x_1, x_2, x_3)$). The function is instrumental in distributing transactions among validators. We use a perfect binary "distribution tree" with k levels as shown in Figure 6. Each transaction is assigned to a leaf with the following recursive algorithm:

1. Assign the transaction to the root
2. If $g(H_T, l) = 0$ then assign the transaction to the left child of its current node, otherwise to the right child. Here H_T is the hash of transaction, and l the level of its current node.
3. Repeat Step 2 until transaction reaches a leaf

Each validator picks a leaf they are responsible for. We want leaves to have roughly even number of validators responsible for them and a simple way to accomplish that is to have nodes pick their leaf randomly, though we do allow them to pick a leaf with their own discretion. A validator will then be "responsible" for all the transactions associated with their leaf and we will call those transactions the "primary transactions" of that validator.

The main purpose of the distribution tree is to distribute out the transactions so that individual nodes would have to handle less of them. Let us show that the proposed mechanism accomplishes that effectively. Suppose the hash of transaction is defined by uniform random variable X and that for each level l the pseudorandom function satisfies $g(X, l) \equiv \mathcal{U}\{0, 1\}$ where $\mathcal{U}\{l, r\}$ is an uniform distribution over integers $(l, l+1, \ldots, r)$.

Let Y be the leaf index assigned to X with the algorithm. It's easy to see that for all $y \in 0, \ldots, 2^k - 1$ we have $P(Y = y) = \frac{1}{2^k}$, in other words $Y \equiv \mathcal{U}\{0, 2^k - 1\}$.

Now suppose we have n i.i.d. transaction hashes $X_1, \ldots, X_n$. The fraction of transactions a leaf i gets is given by:

$$f_i = \frac{1}{n}(P(Y_1 = i) + P(Y_2 = i) + \ldots + P(Y_n = i))$$

where $Y_1, \ldots, Y_n$ are the leaf indices given to transactions. Note that each $P(Y_j = i)$ has the same expected value $\frac{1}{2^k}$. This means that according to the law of large numbers $f_i \to \frac{1}{2^k}$ as $n \to \infty$. With a good pseudorandom function g, this will be a reasonably good approximation of reality, and it provides strong evidence that the approach will distribute transactions roughly evenly between leaves.

The main reason we introduced this "distribution tree" over simply randomly assigning transactions directly to leaves is that this tree allows us to increase k while keeping consistency. For example if we increase k by one, then the transaction will simply be assigned to a new leaf according to $g(H_T, k)$ and all validators have two choices:

1. Pick either the left or right child of their current node in the "distribution tree". They will assume responsibility of all transactions in that leaf, while discarding all transactions that belonging to the other leaf

2. They choose to be responsible for both children, retaining responsibility for all transactions they have and acting as two validators at once

Note that this minimizes network communication and shuffling of data, since they do not have to ask for any new transactions. They either remain responsible for all their transactions, or discard about half of them.

Recall that every transaction has potentially several new outputs and these outputs will be used by the future transactions. This means that validators also have to check for double spending. We facilitate that by having validators store "primary transactions" (given by the distribution tree), and the transactions that use outputs from primary transactions (will be called "secondary transactions").

In summary, each validator stores:

1. **primary transactions** - All transactions the validator is responsible for according to the distribution tree. The average transaction is around 500 to 1000 Bytes.

2. **secondary transactions** - All transactions that use outputs from primary transactions. Transactions have an average of less than 3 outputs per transaction [1], so there are about 3 times more secondary transactions than primary transactions.

3. **Merkle branches** to all stored transaction. Branches consist of 32-Byte hashes and the depth of Merkle tree is $O(\log N)$. In realistic scenarios this should correspond to less than 2000 Bytes per transaction.

Each validator checks that its primary transactions has valid data and whether other nodes have reported these transactions as double spending. They also check that their secondary transactions do not double spend, and in case of double spending they report that to the rest of the network. This way all transactions and the spending of their outputs is tracked by the network of validators.

3.1.3 Miners

In the proposed system, the miners have a similar role as the old miners, but they are more specialized. In the current Bitcoin system miners also act as validators of all Bitcoin transactions, but in our scenario validation is too difficult for a single miner to perform. In the proposed system, the miners simply take some valid block header with a pre-assembled Merkle root and then mine for a nonce. Assembling that Merkle root will be tricky. An obvious solution would be for many miners to pool their resources together and collaboratively act as validators of the whole system, however we will propose a more decentralized system where miners can be more independent, which in turn should lower the barrier for entry for miners.

3.2 Coordination for Block Generation

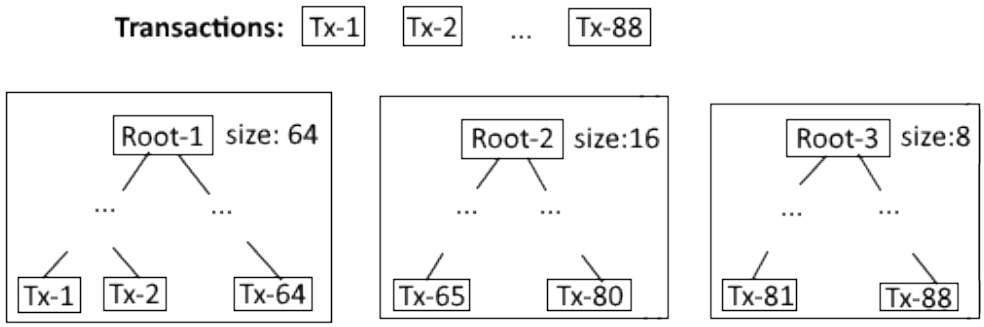

Figure 7. A distribution of 88 transactions into disjoint sets of sizes in the powers of two. Merkle trees are constructed inside those sets, and the validator only has to provide the 3 roots to a miner

Since miners no longer validate transactions themselves, they will have to ask validators for help. Each new transaction will get assigned to a leaf, according to the distribution tree in Figure 6. A validator responsible for that leaf will wait until validators that the transaction as secondary confirm that no double-spending occurred, and then the validator will declare this new transaction valid. A validator can not broadcast all the

transactions individually to miners, so it compresses them into Merkle roots as shown in Figure 7. Next the validator has two choices:

1. It broadcasts the necessary Merkle roots to all miners
2. It either provides or sells the Merkle roots to a specific miner

Note that a Merkle tree is a perfectly balanced binary tree, so it has 2^h leaves. To generate the roots, we need to divide new transactions into disjoint sets with sizes in powers of two and then construct Merkle trees on top of those sets to receive the roots. We can always find a way to divide any number of transactions into $O(\log n)$ such sets and constructing the Merkle trees takes $O(n)$ in total. Only the $O(\log n)$ set roots have to be communicated to the miner. It's possible to add a new transaction to the set of existing Merkle roots in $O(\log n)$ using a method similar to Algorithm 1.

A miner will find a validator for each leaf of the distribution tree, and then acquire the Merkle roots from them. Now the miner has a lot of Merkle roots, but they still have to somehow combine them together to get one Merkle root containing all the new transactions that they can attach to the block header. To present an algorithm for combining Merkle roots, we first need to consider how the current Bitcoin system deals with situations where the number of transactions in a block is not a power of two.

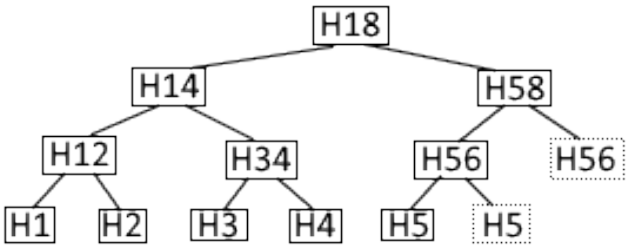

Figure 8. A Merkle tree that the Bitcoin system constructs from 5 transactions. Note how on levels that have an odd number of vertices the last vertex is duplicated

The construction process of a Merkle tree in the Bitcoin system behaves in the following fashion:

1. Hash the transactions to get the lowest level, and start from that level
2. If the number of vertices in the current level is odd, then duplicate the last vertex and append it to the end of that level
3. Hash all consecutive pairs of vertices to obtain the next level
4. Go up a level and if we are not at the root yet, the go back to Step 2

It is a bottom-up constructon algorithm[12] and the way it duplicates and connects is illustrated in Figure 8.

Next we introduce an algorithm for the miner to construct a full Merkle tree from the partial roots they received from the validators. Suppose the miner has acquired an array of root hashes $H = (H_1, \ldots, H_m)$, let $L(H_i)$ correspond to the number of leaves the root hash H_i covers (obvously $L(H_i)$ is a power of two). The full Merkle tree can be constructed with Algorithm 1.

Algorithm 1 An algorithm the miner can use to merge partial roots acquired from validators

Sort elements of H according to $L(H_i)$ in decreasing order
for $p = 0$ **to** ∞ **do**
  if $|H| = 1$ **then**
    return H_1 {This will be used as the Merkle root for the block header}
  end if
  $T \leftarrow ()$
  $T' \leftarrow ()$
  while $L(H_{|H|}) = 2^p$ **do**
    Append $H_{|H|}$ to T
    Erase $H_{|H|}$ from H
  end while
  if $|T|$ is odd **then**
    $T \leftarrow T||(T_{|T|})$ {Duplicates and appends the last element}
  end if
  for $i = 0$ **to** $\frac{|T|}{2} - 1$ **do**
    Append $h(T_{2 \cdot i+1}, T_{2 \cdot i+2})$ to T' {h is the hash function for the Merkle tree}
  end for
  $H \leftarrow H \| T'$
end for

The algorithm sweeps up levels while pairwise merging the lowest length element, duplicating the lowest element if necessary. For Merkle roots with leaf counts $(64, 64, 32, 16, 16, 8, 8)$ the algorithm will process the array of roots in the following fashion:

$$(64, 64, 32, 16, 16, 8, 8) \rightarrow (64, 64, 32, 16, 16, 16) \rightarrow$$
$$(64, 64, 32, 16, 16, 16, 16) \rightarrow (64, 64, 32, 32, 32) \rightarrow$$
$$(64, 64, 32, 32, 32, 32) \rightarrow (64, 64, 64, 64) \rightarrow (128, 128) \rightarrow (256)$$

4 Analysis

In this section we analyze the practical considerations of the given scaling system.

4.1 Storage and Communication Costs

First we focus on the validators. Recall from Section 3.1.2 that each validator stores the transactions they are responsible for (primary transactions), the transactions that use outputs from primary transactions (secondary transactions, on average 3 per primary transaction as the average number of outputs is less than 3) and a Merkle branch to each stored transaction. Each transaction takes less than 1000 Bytes, and Merkle Branch takes around 2000 Bytes. This means that the number of Bytes a validator has to store is $\frac{1}{2^k} N \cdot 4 \cdot 3000$. In terms of network communication, the main expense for validators is acquiring the transactions. The same transactions that are stored, will also have to be transmitted, so the daily rate of communication for a node is around $\frac{1}{2^k} R \cdot 4 \cdot 3000$ Bytes. Note that the validators store $\frac{1}{2^k} N \cdot 4$ transactions, so they store and transmit 3000 Bytes per stored transaction, which is 3 times higher than the less than the 1000 Bytes the current Bitcoin system would store per transaction.

Now let us suppose Bitcoin were to process all cash and non-cash transactions in EU. According to the EU daily transaction rate [2] if Bitcoin were to store all transactions from the past 10 years, then it would have to store around $10 \cdot 365 \cdot 3.3 \cdot 10^8 \equiv 1.2 \cdot 10^{12}$ transactions. If we used a distribution tree of height $k = 17$, then the average storage would be around 109 GB, less than the size of the current blockchain and easily to handle for almost all personal computers. Here $k = 17$ would mean that the distribution tree would have 131072 leaves, which means that for good validation, we need a number of validators much greater than that. Fortunately increased adoption will probably mean more nodes collaborating on the network and powerful entities can act as multiple validator nodes in one, so full validation will still be very manageable for the network.

In our system, the miners and observers do not have to store, nor communicate large amounts of data. The miners will rely on validators to do the transaction handling, they just have to mine nonce for their block headers, so they will be doing less work than in the current Bitcoin system. Observers store only headers, so their storage requirements are very small. There will be some communication of metadata on the network, but this will be much cheaper than handling transactions.

4.2 Validation of Validation

While validators are the one doing the validation of individual transactions, the rest of the network will probably want some assurance that all transactions have been validated, and that the latest blocks are valid. Also validators will want to ensure that they have got all the transactions, that there are not some transactions inside the block that have not

been shared with validators. For that validators will have to share data on their validation process. For each new block a validator will report:

1. The total number of primary transactions they validated
2. The total number of inputs from their primary transactions
3. The total number of secondary transactions they checked for double spending
4. The total number of primary transaction outputs these secondary transactions used

For each leaf of the distribution tree, the validators of that leaf should reach an agreement on the above values. They can cooperate to resolve any discrepancies. The sum of primary transactions validated in each leaf should come to the total number of transactions in the block, since the leaves are responsible for disjoint sets of transactions. Also the sum of observed inputs should be equal to the total sum of validated outputs. If the number of validated transactions does not cover all the block transactions, then there must be some hidden transaction inside not shared with the validators and this makes the block untrustworthy and candidate for orphaning. This means that it is in the interest of validators who worked with the miner to ensure other validators of the same leaf get all the relevant block transactions and it is in the interest of miners to work with validators that are willing to do that. If the sum of the numbers of processed inputs for leaves does not match the sum of the numbers of validated outputs then it means that for some leaf the validators did not do their job properly and is an even greater cause of concern.

A little complication is a fact that the block headers do not contain the number of transactions the block holds. Fortunately that can be deduced from the Merkle root by using the branch to the last transaction. Start with the number of leaves, and then for each level of the branch in which the two elements are the same, subtract a power of two corresponding to that level (for example in Figure 8, there are 8 total leaves and duplicated transactions are on levels 0 and 1, so subract 2^0 and 2^1 to get the correct number of transactions 5).

4.3 Double Spending Within a Block

For blocks that are already in blockchain, the validators ensure that no double-spending occurs. However things are slightly more complicated for miners when assembling new blocks. Suppose we have two transactions, one in leaf 5 and the other in leaf 7, and suppose they use the same Bitcoin. If the validator of leaf 5 gives our miner the first transaction and validator of leaf 7 gives the second transaction, then since the miner does not see individual transactions, it will assemble the partial roots together into a block that contains both transactions, and consequently has double-spending. As a result the miner will spend a lot of effort into mining a block that is actually invalid, resulting in wasted work at no fault of the validators.

A solution could be to track and report which new transactions conflict with other new transactions. A validator of the output used in both transactions (say it was from transaction in leaf 3) will recieve both conflicting transactions. Then the issue can be resolved in the following steps:

1. Validator of leaf 3 will report to validators 5 and 7 that the two transactions conflict
2. Validators of leaves 5 and 7 will exclude the involved transactions from the Merkle roots they provide to the miner
3. They will send the Merkle roots without conflicts to the miner
4. They will send the conflicting transactions individually to the miner
5. When assembling block header, the miner will themselves pick which of the conflicting transactions to include

This works well as long as there are not too many new conflicting transactions. If there are, then the miners can set a limit to how many conflicting transactions they will accept into consideration in order to reduce communication costs.

4.4 Malicious Nodes

We consider the case of malicious validators in greater detail, since that is the most problematic case. Malicious validators can pretend that a block or transaction is valid when it in fact contains double-spending. Fortunately it takes a single honest validator in a leaf to report the double-spending for the whole network to see it, which consequently takes a toll on the reputation of the dishonest validator. A dishonest validator also can not pretend a valid block is invalid, because it would have to provide proof of that invalidity. Therefore as long as there is a single honest validator that sees the transactions of a leaf, the transactions and outputs of that leaf will be properly validated. Things become more interesting when a large number of malicious validators decide to take over some leaf.

Suppose a distribution tree leaf has 1000 validators, and that 80% of them are dishonest and colluding. Suppose some good-intenioned miner decides to ask for partial roots from one of the dishonest validators and that they provide corrupted roots that for example contain double spending of some output. Next, once the block gets proposed, the 80% of dishonest validators can claim that everything is fine, and then refuse to provide the 20% of honest nodes with the actual transactions, effectively barring honest nodes from validating that leaf. For the rest of the network it looks like everything should be fine. They might notice that significant number of validators in that leaf complain about not being able to acquire transactions, but they can not really confirm that the complaints are legitimate unless they decide to validate the leaf themselves.

Some other node might notice the disagreement and decide to check the leaf out by becoming its validator, but then it simply will not receive the transactions and the rest of the network will only see that there is one more node complaining, but that most of the validators still claim that everything is fine. Also if some nodes decide to put effort into investigating such disagreements, then dishonest nodes can simply join properly validated leaves and claim that they are not provided transactions in those leaves, forcing those investigators to waste a lot of effort on empty leads.

A good solution here would take into consideration the real world reputation of a validator. A validator that has a well known real-world entity behind it, a stake in their reputation, a record of honesty and no motive to decieve could be considered reputable. If in a block, the majority of reputable nodes, or some very reputable ones of some leaf claim that they are not recieving transactions, then we can consider that the block is likely dishonestly generated and not worth keeping in the blockchain, which will reduce the effectiveness of flooding a leaf with malicious validators. Note that reputable nodes have an especially strong motive to stay honest, because if they claim that some block is valid, then it takes a single honest node (reputable or not) to show invalidity for their reputation to go down the drain. Now there is still the exceedingly unlikely possibility of a majority of reputable nodes in a leaf falsely claiming that they do not have access to transactions when they do. In such a case we can have miners check out the leaf and then vote with the blocks they have validated in the past (resulting in a sort of proof-of-work vote). If the blocks that have more total difficulty claim that the transactions are available and the leaf is correct, then the complaining validators will lose reputation. This will result in a lot of wasted work, but is hopefully an extremely unlikely scenario.

Dishonest observers are less of a problem. They can not really send false information, since all information will be signed by originator and/or verifiable. They can however hamper the flow of information. Still a decent number of reputable observers can easily spread the necessary information around, and honest but less reputable and visible nodes can pass on the messages from reputable observers. Dishonest miners will be least problematic, because their job is to mine blocks, and their blocks will be heavily scrutinized.

4.5 Comparison to the Current Bitcoin System

The proposed system closesly resembles the original Bitcoin system. The block proposal mechanism is the same and the blocks must follow the same rules (except for the lifted block size limit). The proposed system however has different node behavior, which is necessary because if the scale of Bitcoin grows large, then single nodes will no longer be able to handle the functionalities they are currently handling. The main addition to the Bitcoin system is the validator nodes, since the observer and miner nodes already exist in similar forms. Transaction validation is exactly the part that becomes difficult as Bitcoin grows and the proposed system provides a mechanism to distribute the work

around while still keeping the system decentralized and ensuring thorough validation. The proposed system has a greater reliance on the reputation of nodes, and that will likely be necessary as the incentive to attack and capabilities of attackers grow. The proposed system should be easy to build on top of the current Bitcoin, since Bitcoin in theory allows significant flexibility with how nodes interact with it. We tried to keep our system as close to the principles of Bitcoin as possible within the limits of the scalability problem.

5 Conclusion

In this thesis, we have analyzed the issue of scaling Bitcoin in case of wider adoption and have provided a system to facilitate that. The main addition of that system was to add a class of nodes called validators: nodes that check the validity of all transactions and that ensure that no double spending occurred. We proposed a system to divide up the transactions among validators, allowing us to greatly reduce the barrier to entry for validation while still ensuring that everything gets properly validated. Finally, we analyzed the practical considerations and potential attack scenarios of this system while presenting ways to deal with them.

The proposed system allows for an efficient way to divide up the work, so that even weak compute nodes can contribute. It has a new attack scenario that the Bitcoin system does not thanks to the addition of validators, but fortunately this attack scenario proved to be combatable with an increased reliance on real-world reputation of nodes. The proposed system was built on the fact that Bitcoin allows great deal of flexibility in how nodes interact with it.

As an early proposal of the system, not all possible weaknesses have been considered. One aspect worth investigating is how common attack scenarios stack up against the system, and whether there are some attacks that truly shake it to the core. The proposed solution to an attack of malicious validators demanded an increased reliance on the reputation of nodes and a good avenue of research would be whether this issue could be solved without relying on reputation. Additionally, this system has significant storage and communication overhead per transaction compared to the current Bitcoin system and it would be worth investigating whether those overheads can be reduced.

References

[1] Bitcoin outputs today. https://outputs.today/ (checked 12.05.2018).

[2] European Central Bank. Press release / 15 september 2017, payments statistics for 2016. https://www.ecb.europa.eu/press/pdf/pis/pis2016.pdf?be9989f6bd72483ebe27d8dfae1f0362 (checked 09.05.2018).

[3] Bitcoin.com. Blockchain size chart. https://charts.bitcoin.com/chart/blockchain-size (checked 09.05.2018).

[4] Bitcoin.com. Daily transactions chart. https://charts.bitcoin.com/chart/daily-transactions (checked 09.05.2018).

[5] Bitcoin.org. Bitcoin developer guide. https://bitcoin.org/en/developer-guide (checked 16.05.2018).

[6] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. Bitcoin-ng: A scalable blockchain protocol. https://pdfs.semanticscholar.org/d28d/d2fe450d50a414e83461bfa3449750a573d0.pdf.

[7] Ralph Merkle. *Secrecy, authentication and public key systems*. PhD thesis, 1979.

[8] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf.

[9] Ravi Kiran Raman and Lav R. Varshney. Dynamic distributed storage for scaling blockchains. https://arxiv.org/pdf/1711.07617.pdf.

[10] Blockchain Luxembourg S.A. Bitcoin blockchain statistics. https://blockchain.info/ (checked 12.05.2018).

[11] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing fast money grows on trees, not chains. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.433.6590&rep=rep1&type=pdf.

[12] Bitcoin Wiki. Protocol documentation. https://en.bitcoin.it/wiki/Protocol_documentation (checked 13.05.2018).

[13] Wikipedia. RSA (cryptosystem). https://en.wikipedia.org/wiki/RSA_(cryptosystem) (checked 16.05.2018).

[14] Wikipedia. SHA-2. https://en.wikipedia.org/wiki/SHA-2 (checked 16.05.2018).

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Oliver-Matis Lill**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

   of my thesis

   Bitcoin Scaling with Specialization

   supervised by Vitaly Skachek

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 16.05.2018