# QUANTIFYING AND IMPROVING THE SECURITY OF BLOCKCHAIN SYSTEMS

A Dissertation Outline Presented

by

A. PINAR OZISIK

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2019

College of Information and Computer Sciences

# QUANTIFYING AND IMPROVING THE SECURITY OF BLOCKCHAIN SYSTEMS

A Dissertation Outline Presented

by

A. PINAR OZISIK

Approved as to style and content by:

Brian N. Levine, Chair

Philip S. Thomas, Member

Yuriy Brun, Member

Nikunj Kapadia, Member

James Allan, Chair
College of Information and Computer Sciences

# ABSTRACT

## QUANTIFYING AND IMPROVING THE SECURITY OF BLOCKCHAIN SYSTEMS

SEPTEMBER 2019

A. PINAR OZISIK

B.Sc., BRANDEIS UNIVERSITY

M.S., UNIVERSITY OF MASSACHUSETTS

Directed by: Professor Brian N. Levine

In this thesis, I analyze and improve the security and performance of blockchain systems across three primary themes. In the first theme, I analyze blockchain algorithms for setting block discovery difficulty. Unfortunately, churn in mining power can cause uneven inter-block delays when the difficulty is not set accurately. Mining power can change due to many reasons, including the miners' allocation of hardware and swings in the exchange rate of a currency. For example, Bitcoin Cash has seen enormous variance in mining power since its creation and the existing algorithm for difficulty did not easily converge. I propose two alternatives to accurately update difficulty: one that solely uses information that is currently available in blockchain networks, and another based on status reports regularly broadcast from some or all miners of their partial proof-of-work (POW). Status reports can also be used for emergency difficulty adjustment, an algorithm the network resorts to when a block takes unusually long to discover.

Status reports add overhead into networks because they require the broadcast of additional information. In a second theme, I introduce a novel method of interactive set reconciliation for the distribution of status reports in order to reduce traffic. Even without status reports, this protocol works for the efficient distribution of blocks. The approach, called Graphene, couples a Bloom filter with an IBLT. Then I evaluate performance analytically and show that Graphene blocks are always smaller and therefore network performance is improved.

In the third theme, I analyze the practical feasibility of double-spend and selfish mining attacks on blockchain systems. The hash rate of miners is the primary quantitative factor that determines the security of any POW based blockchain consensus algorithm. Most analyses generally assume that the hash rate of honest and malicious miners is known. However, I show that hash rate estimation is difficult and introduces high variance. Therefore, I argue that these double-spend and selfish mining attacks are difficult to carry out with high precision, and use reinforcement learning techniques to realistically evaluate these attacks when an attacker does not have full knowledge of the networks mining power.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

**Figure** **Page**

# INTRODUCTION

## Contributions

The following is a summary of the contributions in each chapter of this proposal.

**1.**

## Collaborators

All research activities are conducted under the supervision of Brian Levine. Preliminary work for Chapter 2 was completed in collaboration with George Bissias and Brian Levine; that for Chapter 3 was completed in collaboration with Gavin Andresen, George Bissias, Amir Houmansadr and Brian Levine; and that for Chapter 4 was completed in collaboration with Phil S. Thomas and Brian Levine.

# CHAPTER 1

# OVERVIEW OF BLOCKCHAIN SYSTEMS

## 1.1 Basic Operation

**Account balances.** A Bitcoin is a unit of currency, which is fungible, divisible (up to eight decimal places), and recombinable. It is measured as a balance across multiple accounts, which are themselves manifested in *addresses*.[1] Each address comprises a stored asymmetric cryptographic key and an associated balance of Bitcoin. The public portions of an address are the public key and the balance of coin. When an address is involved in a *transaction* with one or more other addresses, Bitcoins are transferred among them.

**Roles.** Users wishing to exchange coins broadcast the details of their transactions over Bitcoin's p2p network, signed with their private keys. A set of *miners* on the p2p network verify that each transaction is signed correctly and does not conflict with another transaction. Miners independently agglomerate a set of valid transactions into a *block* and attempt to solve a predefined proof-of-work (POW) problem involving this block and a chain of prior valid blocks. In Bitcoin, the POW computation is dynamically calibrated to take approximately ten minutes per block. The first miner to solve the problem broadcasts his solution to the network, adding it to the ever-growing *blockchain*; the miners then start over, with the appended blockchain and the set of transactions that were not added as part of the previous block. When transactions appear in a block, they are considered *confirmed*, and each subsequent

[1]Internally, Bitcoins exist only as "unspent transaction outputs" (UTXO), but users of the system think of them as balances in addresses, and that view does not affect the results of this paper.

block provides additional confirmation. The miners' incentive for discovering a block is a reward of coins, called the *coinbase*, consisting of a predetermined *block reward* (currently 12.5 BTC) and fees from transactions included in the block.

*Full nodes* are peers in the network that do not mine, but do generate, validate, and propagate transactions and blocks to other nodes including miners. Consumers (i.e., those who purchase goods or services) typically have no need to process and validate all transactions, so they can instead operate *simple payment verification* (SPV) nodes that process, store, and transmit data involving only addresses-of-interest, which are typically addresses they control, make payments to, or receive payments from. SPV nodes rely on full nodes to relay transactions-of-interest.

**Bitcoin transaction consistency.** The main goal of the Bitcoin p2p network is to provide a consistent view of blocks and unconfirmed transactions across all network peers. Each peer maintains a local snapshot of the transactions in a memory pool dubbed the *mempool*. Blocks consist of a list of transactions that have already (almost always) been broadcast to miners and full nodes in the network.

To announce a new block, a miner lists all transactions contained in the new block along with a header that provides an easily verifiable *proof-of-work* (POW) solution. When a full node or miner receives a new block, it validates each transaction in the block and the proof of work.

Due to propagation delays in the network, it is possible for the miners to receive competing (but valid) block announcements, which bifurcates the chain, until one of the two forks is appended to first. It is also possible and valid for a miner to receive a set of blocks that retroactively rewrites many blocks; doing so is a demonstration of computational work that miners accept despite the age or depth[2] of a rewritten block.

[2] The *depth* of a block refers to the number of blocks that follow it; the *height* of a block is the number of blocks that precede it.

**Topology and flooding.** Bitcoin propagates new transaction and block announcements by flooding throughout a p2p random graph of full nodes and miners. Each peer in the graph requests direct connections to 8 other peers, and accepts requests for connections from up to 117 other peers. A peer will offer a newly created transaction to each neighbor via an `inv`message, which reports the hash of the transaction content as its ID. If a peer does not already possess the transaction, it will request it using a `getdata` message. Blocks are handled similarly: `inv`messages describe a block by its ID, which is created from the hash of the block's contents. Upon receiving the `inv`, peers will request the block if they do not already have it. Hence, in today's topology, `inv`messages cross every edge in the random graph once, while the actual transaction and block data typically propagate along only a spanning tree of the graph (more edges will be traversed if there are propagation delays). For convenience, in this paper, we refer to the set of (unconfirmed) transaction IDs that a peer knows about as the *IDpool*. Actual transaction contents are placed in the mempool.

# CHAPTER 2

# DIFFICULTY ESTIMATION

## 2.1 POW in Blockchain Systems

Bitcoin uses a simple POW algorithm based on cryptographic hashing, proposed earlier by Douceur [7]. Specifically, miners apply a 256-bit cryptographic hash algorithm [15] to an 80-byte *block header*, and the puzzle is solved if the resulting value is less than a known *target*, $0 < t < 2^{256}$. The header in Bitcoin consists of the Merkle root of the set of transactions, a timestamp, the target (stored as $2^{224}/t$), a *nonce*, and the hash of the prior block's header. If the hash is not less than the target, then a new nonce is selected to generate a new hash (the Merkle root can be adjusted as well). This process repeats until some miner finds a solution.

Each time a nonce is selected and the block header is hashed, the miner is sampling a value from a discrete uniform distribution with range $[0, 2^{256}-1]$. The probability of solving the POW and discovering a block is the cumulative probability of selecting a value from $[0, t]$, which is $t/2^{256}$. Hence, in expectation, the number of samples needed to discover a block is $2^{256}/t$. Bitcoin adjusts the target so that on average it takes about 600 seconds to find a block. Typically, the target is described for convenience as a *difficulty*, defined to be $D = 2^{224}/t$. Bitcoin's difficulty is set once every two weeks.

**Ethereum.** Ethereum operates very similarly to Bitcoin. The following differences are relevant to the context of this paper. Ethereum miners solve a POW problem that is more complicated than Bitcoin in an attempt to disadvantage miners with custom ASICs. However, in the end, a miner still compares a hash value to the target. Specifically, the number of values in the block header is larger, resulting in a 508-byte header. It's not the hash of the header that is compared against the target, but the hash resulting from an Ethereum-specific algorithm called ETHASH [9], for which the hash of the block header is the primary input. In the end, the POW hash value is a sample from a discrete uniform distribution with range $[0, 2^{256} - 1]$, and the probability of block discovery is $t/2^{256}$.

A major difference of Ethereum is that the target is set such that the expected time between blocks is 15 seconds. This setting results in quicker confirmation times, but as a result, the probability that two miners announce blocks within the propagation time of a block announcement is much higher. Therefore, there are many abandoned forks in the chain. Ethereum uses a modified version of the GHOST [21] protocol for selecting the main fork of the blockchain: the main chain follows the block at each level with the most POW on its subtree. These differences do not affect the application of our algorithms; in fact, the presence of ommers is additional data which improves our estimates.

## 2.2 Problem Statement

Developers have resorted to ad-hoc methods for updating the difficulty in many blockchain systems. So far, there has been no previous work on analyzing the efficiency and correctness of these methods. In fact, because some blockchain systems do not accurately update difficulty, networks see enormous variance in inter-block delay. If mining power were constant in these networks, then difficulty could be kept constant. However, as seen with many blockchain systems, including a recent example with Bitcoin Cash, mining power fluctuates within these networks, requiring a readjustment of network parameters, the most important being difficulty. Therefore, in this chapter, I use the most prominent cryptocurrency, Bitcoin, as a testbed for analyzing difficulty, and then propose alternative methods that could potentially increase efficiency.

## 2.3 Preliminary Work

### 2.3.1 Analysis of Bitcoin's Difficulty Estimation

**Algorithm for setting difficulty.** Bitcoin's networks parameters are set such that a block is discovered every 10 minutes. Initially, difficulty starts at 1, and then for every 2016 blocks that are found, the timestamps of the blocks are compared to find out how much time it took to find 2016 blocks. Let $t$ denote the time in minutes it took to find 2016 blocks. Because the network is configured such that 2016 blocks must take 2 weeks (20160 minutes), the old difficulty is multiplied by $20160/t$. If the correction factor is greater than 4 or less than 1/4, then 4 or 1/4 are used, respectively, to prevent the change from being too abrupt.

Bitcoin's target and difficulty are related to each other as follows: target = targetmax / difficulty $= 2^{224}/D_i$, where $D_i$ is the $i$th time the difficulty is set [6]. The difficulty and target are inversely proportional: when difficulty increases, the target decreases. Therefore, a smaller target makes block creation more difficult, and as the difficulty goes up, so does the expected time needed to create a block.

**Analysis of difficulty adjustment.** Let $D_i$ denote the $i$th time the difficulty is set, and $X_k$ denote the number of minutes it took to generate the $k$th block after $D_i$ is set. Then we have

$$D_i = D_i \frac{10n}{\sum_{k=1}^{n} X_k}. \tag{2.1}$$

Let D be a sequence generated by the last equation where the first element of the sequence is $D_0 = 1$. Mining is an example of a Poisson process because, under constant mining power, blocks are mined continuously and independently at a constant average rate. Therefore, $X_k \sim \text{Exp}(\beta)$ with $\beta = 1/\lambda$, assuming that the miner hash rate stays constant for the 2-week period after $D_i$ is set. In this parametrization of the exponential, $\beta$ represents the survival parameter, and hence, the ratio describing the *expected time* it takes for one block to arrive. For example, ideally in Bitcoin $\beta = 1/10$ minutes and in Ethereum $\beta = 1/15$ seconds. The difficulty for the $i+1$st time given $D_i$ and $X_1, \dots, X_n$ is

$$D_{i+1} = 10nD_i\left(\frac{1}{\sum_{k=1}^{n} X_k}\right). \tag{2.2}$$

**The relationship between hash rate and $\boldsymbol{\beta}$.** Given difficulty $D_i$, the expected number of hashes, $h$, needed to meet the target for a block is

$$\mathbb{E}[h] = \frac{2^{256}-1}{T_i} = \frac{2^{256}-1}{2^{224}/D_i} = \frac{D_i(2^{256}-1)}{2^{224}}, \tag{2.3}$$

where $T_i$ is the target set for the $i$th time. Note that $\mathbb{E}[h]$ describes the *total* number of expected hashes needed to discover a block, and I have observations regarding the *time* it takes to generate a block. Let $r$ be the hash rate of the network in minutes (or the number of hashes per time unit), and $X = X_1, \dots, X_n$, where $X \sim \text{Exp}(\beta)$, with $\beta = 1/\lambda$. $\lambda r$ is the expected number of hashes each time a block is created.

$$\mathbb{E}[h] = r\lambda = r\frac{1}{\beta} \tag{2.4}$$

$$r = \mathbb{E}[h]\beta. \tag{2.5}$$

**Adjusting difficulty correctly.** For Bitcoin, where the network is expected to solve a block every 10 minutes, we can adjust the target for the $i+1$th time as follows

$$\frac{(2^{256}-1)}{T_{i+1}} = 10r \tag{2.6}$$
$$\frac{(2^{256}-1)}{T_{i+1}} = \frac{10(2^{256}-1)\beta}{T_i} \tag{2.7}$$
$$T_{i+1} = \frac{T_i}{10\beta}. \tag{2.8}$$

Therefore, by rearranging and substituting $T_i$ with its definition using $D_i$, we can adjust the difficulty for the $i+1$th time as follows

$$D_{i+1} = \frac{2^{224}}{T_{i+1}} \tag{2.9}$$
$$= 10\beta D_i. \tag{2.10}$$

**Expected value of difficulty.** We can talk about the expected value of a term in sequence D, given its preceding term and the new data we see.

$$\mathbb{E}[D_{i+1}|D_i, X_1, \ldots, X_n] = \mathbb{E}\left[10nD_i\frac{1}{Y}\right] \tag{2.11}$$
$$= 10nD_i\,\mathbb{E}\left[\frac{1}{Y}\right] \tag{2.12}$$
$$= 10nD_i\left(\frac{1}{\beta(n-1)}\right) \tag{2.13}$$
$$= \frac{10nD_i}{\beta(n-1)}. \tag{2.14}$$

**Variance of difficulty.** Additionally, we can also talk about the variance of a term in sequence D, given its preceding term and the new data we see.

$$\text{Var}(D_{i+1}|D_i, X_1, \ldots, X_n) = \text{Var}\left(10nD_i \frac{1}{Y}\right) \tag{2.15}$$

$$= (10nD_i)^2 \text{Var}\left(\frac{1}{Y}\right) \tag{2.16}$$

$$= (10nD_i)^2 \left(\frac{1}{\beta^2(n-1)^2(n-2)}\right) \tag{2.17}$$

$$= \frac{(10nD_i)^2}{\beta^2(n-1)^2(n-2)}. \tag{2.18}$$

**Bias of difficulty.**

$$\text{bias}(D_{i+1}|D_i, X_1, ..., X_n) = \mathbb{E}[D_{i+1}|D_i, X_1, ..., X_n] - D_{i+1} \tag{2.19}$$

$$= \frac{10nD_i}{\beta(n-1)} - 10\beta D_i. \tag{2.20}$$

**Mean squared error (MSE) of difficulty.**

$$\text{MSE}(D_{i+1}|D_i, X_1, ..., X_n) = \text{bias}(D_{i+1}|D_i, X_1, ..., X_n)^2 + \text{Var}(D_{i+1}|D_i, X_1, ..., X_n) \tag{2.21}$$

$$= \left(\frac{10nD_i}{\beta(n-1)} - 10\beta D_i\right)^2 + \frac{(10nD_i)^2}{\beta^2(n-1)^2(n-2)}. \tag{2.22}$$

### 2.3.2 Alternative Estimation

**Estimator for $\beta$, the expected inter-arrival time between blocks.** Let $X = X_1, \ldots, X_n$ denote the inter-arrival time between $n+1$ consecutive blocks on the blockchain. Given a consecutive sequence of $n+1$ blocks, $n$ inter-arrival times can be computed by subtracting the timestamp of each block from that of its preceding block. Note that $X \sim \text{Exp}(\beta)$ with $\beta = 1/\lambda$, similar to the definition in the previous section. It it well known that the unbiased MLE estimator for $\beta$ is

$$\hat{\beta} = \frac{\sum_{k=1}^{n} X_i}{n}. \tag{2.23}$$

**Adjusting difficulty.** Using our estimation of $\beta$, we can adjust the difficulty for the $i+1$th time as follows

$$D_{i+1} = 10 D_i \hat{\beta}. \tag{2.24}$$

**Expected Value of New Difficulty.**

$$\begin{aligned}
\mathbb{E}[D_{i+1}|D_i, X_1, \ldots, X_n] &= \mathbb{E}[10 D_i \hat{\beta}] && (2.25)\\
&= 10 D_i \, \mathbb{E}[\hat{\beta}] && (2.26)\\
&= 10 D_i \beta. && (2.27)
\end{aligned}$$

**Variance of New Difficulty.**

$$\begin{aligned}
\mathrm{Var}(D_{i+1}|D_i, X_1, \ldots, X_n) &= \mathrm{Var}(10 D_i \hat{\beta}) && (2.28)\\
&= (10 D_i)^2 \mathrm{Var}(\hat{\beta}) && (2.29)\\
&= \frac{(10 D_i \beta)^2}{n}. && (2.30)
\end{aligned}$$

**Bias of New Difficulty.**

$$\begin{aligned}
\mathrm{bias}(D_{i+1}|D_i, X_1, ..., X_n) &= \mathbb{E}[D_{i+1}|D_i, X_1, ..., X_n] - D_{i+1} && (2.31)\\
&= 10 D_i \beta - 10 D_i \beta && (2.32)\\
&= 0. && (2.33)
\end{aligned}$$

**MSE of New Difficulty.**

$$\mathrm{MSE}(D_{i+1}|D_i, X_1, ..., X_n) = \mathrm{bias}(D_{i+1}|D_i, X_1, ..., X_n)^2 + \mathrm{Var}(D_{i+1}|D_i, X_1, ..., X_n) \tag{2.34}$$

$$= \frac{(10 D_i \beta)^2}{n}. \tag{2.35}$$

### 2.3.3 Comparison of Estimators

Note that our alternative estimator has zero bias compared to Bitcoin's original estimator. Additionally, under the appropriate constraints, variance and MSE is also significantly lower.

**Variance.**

$$\frac{(10D_i\beta)^2}{n} \leq \frac{(10nD_i)^2}{\beta^2(n-1)^2(n-2)} \tag{2.36}$$

$$\frac{\beta^2}{n} \leq \frac{n^2}{\beta^2(n-1)^2(n-2)} \tag{2.37}$$

Our estimator (LHS) has lower variance than the original estimator (RHS) for $n > 2$ and $0 < \beta < 1$.

**MSE.**

$$\frac{(10D_i\beta)^2}{n} \leq \left(\frac{10nD_i}{\beta(n-1)} - 10\beta D_i\right)^2 + \frac{(10nD_i)^2}{\beta^2(n-1)^2(n-2)} \tag{2.38}$$

Our estimator (LHS) has lower variance than the original estimator (RHS) for $n > 2$, $0 < \beta < 1$ and $D_i \in \mathcal{R}$.

## 2.4 Proposed Work

To extend this chapter, I propose to answer a few key questions:

1. See what window works best here, how error changes with window

Chernoff bounds (error bars) is replaced What happens when you introduce time as an error Given a required error rate, what is the shortest window of time Then set the difficulty Sliding window or weighted sliding window Emergency difficulty adjuster How often to adjust difficulty – what if it was every 30 mins instead of every 15 seconds what are potential attacks on status reports - what are the chernoff bounds how to hit the 10 min mark

Given that the mining power is changing, how can you quickly adopt to that? 2 week window of target adjustment is too much you can totally lie about your timesteps, like put the wrong timestamp on the block you create are there any attacks here you want the target to stay challenging so you can go back and forth how much of the mining power do you need to shift the target Malicious denial-of-service attacks and we quantify them here

# CHAPTER 3

# GRAPHENE

## 3.1 Background

In this section, we review the operation of IBLTs and summarize related work.

### 3.1.1 Overview of IBLTs

**Overview of IBLTs.** We make use of Invertible Bloom Lookup Tables (IBLTs) [13], which is an efficient data structure for *set reconciliation* between two peers. Like Bloom filters [3], IBLTs allow two parties to determine, with high probability, which values from a set they share in common. But unlike Bloom filters, IBLTs enable the recovery of any missing values, which are assumed to be of fixed size and encoded as binary strings. Key-value pairs can be inserted, retrieved and deleted like an ordinary hash table. An IBLT consists of $m$ entries, each storing a `count`, a `keySum`, and a `valueSum`, all initialized to zero.

A new value $v$ is inserted into location $i = h(v)$ based on the hash of its value such that $i < m$. At entry $i$, all three fields are incremented or xored. IBLTs use $k > 1$ hash functions to store each value in $k$ entries, which we collectively call a value's *entry set*. If table space is sufficient, then with high probability for at least one of the $k$ entries, `count` $\equiv 1$.

Suppose that two peers each have a list of values, $V$ and $V'$, respectively, such that the difference is expected to be small. The first peer constructs an IBLT $L$ (with $m$ entries) from $V$. The second peer constructs $V'$ from $L'$ (also having $m$ entries). Eppstein et al. [8] showed that a cell-by-cell difference operator can be used to efficiently compute the symmetric difference $L \triangle L'$. For each pair of fields $(f, f')$, at each entry in $L$ and $L'$, we compute either $f \oplus f'$ or $f - f'$ depending on the field type. When $|\texttt{count}| \equiv 1$ at any entry, the corresponding value can be recovered. Peers proceed by removing the recoverable key-value pair from all entries in the value's entry set. This process will generally produce new recoverable entries, and continues until nothing is recoverable.

### 3.1.2 Related Work

The main limitation we are addressing with Graphene is the inefficiency of blockchain systems in disseminating block data. A block announcement must be validated using the transaction content comprising the block. However, it is likely that the majority of the peers have already received these transactions, and they only need to discern them from those in their mempool. In principle, a block announcement needs to include only the IDs of those transactions, and accordingly, Corallo's *Compact Block* design [5] — which has been recently deployed — significantly reduces block size by including a transaction ID list at the cost of increasing coordination to 3 roundtrip times. We further detail Compact Block's operation in Section **??** and compare it quantitatively in Section **??**. *Xtreme Thinblocks* [23], an alternative protocol, works similarly to Compact Blocks but has greater data overhead. Specifically, if an `inv`is sent for a block that is not in the receiver's mempool, the receiver sends a Bloom filter of her IDpool along with the request for the missing block. As a result, Xtreme Thinblocks are larger than Compact Blocks but require just 2 roundtrip times. Relatedly, the community has discussed in forums the use of IBLTs (alone) for reducing block announcements [1,19], but these schemes have not been formally evaluated and are less efficient than our approach. Our novel method, which we prove and demonstrate is smaller than all of these recent works, requires just 2 roundtrip times for coordination.

## 3.2 Graphene: Efficient Block Announcements

In this section, we detail *Graphene*, where a receiver learns the set of specific transaction IDs that are contained in a (pending or confirmed) block containing $n$ transactions. Unlike other approaches, Graphene never sends an explicit list of transaction IDs, instead it sends a small Bloom filter and a very small IBLT.

**PROTOCOL 1: Graphene**

1: Sender: Sends *inv* for a block.

2: Receiver: Requests unknown block; includes count of txns in her IDpool, $m$.

3: Sender: Sends Bloom filter $\mathcal{S}$and IBLT $\mathcal{I}$(each created from the set of $n$ txn IDs in the block) and essential Bitcoin header fields. The FPR of the filter is $f = \frac{a}{m-n}$, where $a = n/(c\tau)$.

4: Receiver: Creates IBLT $\mathcal{I}'$ from the txn IDs that pass through $\mathcal{S}$. She decodes the *subtraction* [8] of the two blocks, $\mathcal{I} \triangle \mathcal{I}'$.

### 3.2.1 The Protocol

The intuition behind Graphene is as follows. The sender creates an IBLT $\mathcal{I}$from the set of transaction (txn) IDs in the block. To help the receiver create the same IBLT (or similar), he also creates a Bloom filter $\mathcal{S}$of the transaction IDs in the block. The receiver uses $\mathcal{S}$to filter out transaction IDs from her pool of received transaction IDs (which we call the IDpool) and creates her own IBLT $\mathcal{I}'$. She then attempts to use $\mathcal{I}'$to *decode* $\mathcal{I}$, which, if successful, will yield the transaction IDs comprising the block. The number of transactions that falsely appear to be in $\mathcal{S}$, and therefore are wrongly added to $\mathcal{I}'$, is determined by a parameter controlled by the sender. Using this parameter, he can create $\mathcal{I}$such that it will decode with very high probability.

A Bloom filter is an array of $x$ bits representing $y$ items. Initially, the $x$ bits are cleared. Whenever an item is added to the filter, $k$ bits, selected using $k$ hash functions, in the bit-array are set. The number of bits required by the filter is $x =$

$y\frac{-\ln(f)}{\ln^2(2)}$, where $f$ is the intended false positive rate (FPR). For Graphene, we set $f = \frac{a}{m-n}$, where $a$ is the expected difference between $\mathcal{I}$and $\mathcal{I}'$. Since the Bloom filter contains $n$ entries, and we need to convert to bytes, its size is $\frac{-\ln(\frac{a}{m-n})}{\ln^2(2)}\frac{1}{8}$. It is also the case that $a$ is the primary parameter of the IBLT size. IBLT $\mathcal{I}$can be decoded by IBLT $\mathcal{I}'$with very high probability if the number of cells in $\mathcal{I}$is $d$-times the expected symmetric difference between the list of entries in $\mathcal{I}$and the list of entries in $\mathcal{I}'$. In our case, the expected difference is $a$, and we set $d = 1.5$ (see Eppstein et al. [8], which explores settings of $d$). Each cell in an IBLT has a *count*, a *hash* value, and a stored *value*. (It can also have a key, but we have no need for a key). For us, the count field is 2 bytes, the hash value is 4 bytes, and the value is the last 5 bytes of the transaction ID (which is sufficient to prevent collisions). In sum, the size of the IBLT with a symmetric difference of $a$ entries is $1.5(2 + 4 + 5)a = 16.5a$ bytes. Thus the total cost in bytes, $T$, for the Bloom filter and IBLT are given by $T(a) = n\frac{-\ln(f)}{c} + a\tau = n\frac{-\ln(\frac{a}{m-\mu})}{c} + a\tau$, where all Bloom filter constants are grouped together as $c = 8\ln^2(2)$, and we let the overhead on IBLT entries be the constant $\tau = 16.5$.

To set the Bloom filter as small as possible, we must ensure that the FPR of the filter is as high as permitted. If we assume that all `inv`messages are sent ahead of a block, we know that the receiver already has all of the transactions in the block in her IDpool (they need not be in her mempool). Thus, $\mu = n$; i.e., we allow for $a$ of $m - n$ transactions to become false positives, since all transactions in the block are already guaranteed to pass through the filter. It follows that

$$T(a) = n\frac{-\ln(\frac{a}{m-n})}{c} + a\tau. \tag{3.1}$$

Taking the derivative w.r.t. $a$, Eq. 3.1 is minimized[1] when when $a = n/(c\tau)$.

Due to the randomized nature of an IBLT, there is a non-zero chance that it will fail to decode. In that case, the sender resends the IBLT with double the number of cells (which is still very small). In our simulations, presented in the next section, this doubling was sufficient for the incredibly few IBLTs that failed.

**PROTOCOL 2: CompactBlocks**

1: Sender: Sends `inv`for a block that has $n$ txns.

2: Receiver: If block is not in mempool, requests compact block.

3: Sender: Sends the block header information, all txn IDs in the block and any full txns he predicts the sender hasn't received yet.

4: Receiver: Recreates the block and requests missing txns if there exist any.

### 3.2.2 Comparison to Compact Blocks.

Compact Blocks [5] is to our knowledge the best-performing related work. It has several modes of operation. We examined the *Low Bandwidth Relaying* mode due to its bandwidth efficiency, which operates as follows. After fully validating a new block, the sender sends an `inv`, for which the receiver sends a *getdata* message if she doesn't have the block. The sender then sends a *compact block* that contains block header information, all transaction IDs (shortened to 5 bytes) in the block, and any transactions that he predicts the receiver does not have (e.g., the coinbase). If the receiver still has missing transactions, she requests them via an `inv`message.

[1]Actual implementations of Bloom filters and IBLTs involve several (non-continuous) ceiling functions such that we can re-write:

$$T(a) = \left( \lceil \ln(\frac{m-n}{a}) \rceil \left\lceil \frac{n \ln(\frac{m-n}{a})}{\lceil \ln(\frac{m-n}{a}) \rceil \ln^2(2)} \right\rceil \right) \frac{1}{8} + \lceil a \rceil \tau. \quad (3.2)$$

The optimal value of Eq. 3.2 can be found with a simple brute force loop. We compared the value of $a$ picked by using $a = n/(c\tau)$ to the cost for that $a$ from Eq. 3.2, for valid combinations of $50 \leq n \leq 2000$ and $50 \leq m \leq 10000$. We found that it is always within 37% of the cost of the optimal value from Eq. 3.2, with a median difference of 16%. In practice, a for-loop brute-force search for the lowest value of $a$ is almost no cost to perform, and we do so in our simulations.

Protocol 2 outlines this mode of Compact Blocks. The main difference between Graphene and Compact Blocks is that instead of sending a Bloom filter and an IBLT, the sender sends block header information and all shortened transaction IDs to the receiver.

A detailed example of how to calculate the size of each scheme is below; but we can state more generally the following result. For a block of $n$ transactions, Compact Blocks costs $5n$ bytes. For both protocols, the receiver needs the `inv`messages for the set of transactions in the block before the sender can send it. Therefore, we expect the size of the IDpool of the receiver, $m$, to be constrained such that $m \geq n$. Assuming that $m > 0$ and $n > 0$, the following inequality must hold for Graphene to outperform Compact Blocks:

$$n\frac{-\ln(\frac{a}{m-n})}{c} + a\tau \quad < \quad 5n \tag{3.3}$$

$$n \quad > \quad {}^{m}/_{1287670} \tag{3.4}$$

In other words, Graphene is strictly more efficient than Compact Blocks *unless* the set of unconfirmed transactions held by peers is 1,287,670 times larger than the block size (e.g., over 22 billion unconfirmed transactions for the current block size.) Finally, we note that Xtreme Thinblocks [23] are strictly larger than Compact Blocks since they contain all IDs and a Bloom filter, and therefore Graphene performs strictly better than Xtreme Thinblocks as well. In Section **??**, we provide specific empirical results from network simulation, where we use real IBLTs and Bloom filters to evaluate Graphene and Compact Blocks.

**Example.** A receiver with an IDpool of $m = 4000$ transactions makes a request for a new block that has $n = 2000$ transactions. The value of $a$ that minimizes the cost is $a = n/(c\tau) = 31.5$. The sender creates a Bloom filter $\mathcal{S}$with $f = \frac{a}{m-n} = 31.5/2000 = 0.01577$, with total size of $2000 \times \frac{-ln(0.01577)}{c} = 2.1$ KB. The sender also creates an IBLT with $a$ cells, totaling $16.5a = 521B$. In sum, a total of $2160B + 521B = 2.6$ KB bytes are sent. The receiver creates an IBLT of the same size, and using the technique introduced in Eppstein et al. [8], the receiver subtracts one IBLT from the other before decoding. In comparison, for a block of $n$ transactions, Compact Blocks costs $2000 \times 5B = 10$ KB, over 3 times the cost of Graphene.

**Ordered blocks.** Graphene does not specify an order for transactions in the blocks, and instead assumes that transactions are sorted by ID. Bitcoin requires transactions depending on another transaction in the same block to appear later, but a canonical ordering is easy to specify. If a miner would like to order transactions with some proprietary method (e.g., [14]), that ordering would be sent alongside the IBLT. For a block of $n$ items, in the worst case, the list will be $n \log_2(n)$ bits long. Even with this extra data, our approach is much more efficient than Compact Blocks. In terms of the example above, if Graphene was to impose an ordering, the additional cost for $n = 2000$ transactions would be $n \log_2(n)$ bits $= 2000 \times log_2(2000)$ bits $= 2.74$ KB. This increases the cost of Graphene to 5.34 KB, still almost half of Compact Blocks.

### 3.2.3 Empirical Evaluation

# CHAPTER 4

# RL APPLIED TO BITCOIN

## 4.1 Background

### 4.1.1 Selfish Mining and Double-Spend Attacks

Blockchain systems [17] provide probabilistic consensus [24] among a set of peers on the order and validity of a set of blocks, each containing transactions. The blockchain encodes the consensus as the *main chain* among all possible forks. As the *depth* of a block on the main chain increases, there is an exponentially decreasing probability that network consensus will switch the main chain to a fork that does not include the block [11,17]. This result assumes that the attacker never stops mining on the alternative fork, regardless of cost. A more realistic model assumes that attackers have finite time and resources, and would not expend resources on an attack that isn't expected to be profitable.

Recently, Gervais et al. [12] evaluated the security of blockchains in terms of an attacker's economic profitably and assuming finite resources. They modeled a double-spending attacker's strategy as Markov Decision Process. MDPs are defined by a finite set of discrete states, a set of actions, a transition function, and a reward function. They defined each state in the MDP as a tuple representing the status of the fork, and the number of blocks mined by an attacker, the honest miners, and *eclipse attack* victim [16], respectively. Gervais et al. encoded several factors into the MDP that affect attacker strategy, including mining power, block depth, connectivity, and the impact of eclipse attacks. The MDP was implemented with a cutoff value of 20 blocks, representing an attack of finite duration. Via a search algorithm over the space defined by the MDP, and in combination with a simulation of a blockchain system that determined for example block propagation times and throughput, they determined the maximum transaction value that would be safe from double spending by an economically rational attacker. This approach is rich, capturing the optimal strategy for double-spending (as well as selfish mining [10, 20]) given network conditions and blockchain parameters. Gervais et al. were able to reach many fascinating conclusions about the comparative performance and security of several widely used blockchains.

**Double spending.** A fundamental attack against Bitcoin is the *double-spend* attack [17], which works as follows. An attacker creates a transaction that moves funds to a merchant's address. After the transaction appears in the newest block on the main branch, the attacker takes possession of the purchased goods. Using his mining power, the attacker then immediately releases two blocks, with a transaction in the first that moves the funds to a second attacker-owned address. Now the attacker has the goods and his coin back. To defend against the attack, a merchant can refuse to release goods to a Bitcoin-paying customer until $z$ blocks have been added to the blockchain including the first block containing a transaction moving coin to the merchant's address. Nakamoto calculated the probability of the attack succeeding assuming that the miner controlled a given fraction of the mining power [17]; for a given fraction, the probability of success decreases exponentially as $z$ increases.

In general, a merchant may wait $z$ blocks before releasing goods, which can thwart an attacker. But choosing the minimum value of $z$ that secures a transaction is an unresolved issue. The core Bitcoin client shows that a transaction is unconfirmed until it is 6 blocks deep in the blockchain [2], and advice from others is necessarily vague; e.g., "for very large transactions, coin owners might want to wait for a larger number of block confirmations" [4].

### 4.1.2 Related Work

The standard Bitcoin protocol requires miners to broadcast a block they mined immediately. However, in the case of selfish mining, a miner purposefully withholds his blocks. The motivation is to bifurcate the chain and waste the computational resources of the honest miners should the network decide to build on the attacker's chain. An attacker can not profit economically since the number of blocks that can be created by a miner depends on the fraction of the mining power he has. However, selfish mining discards the honest miners' blocks, by releasing an alternative chain that takes over the current longest chain. If a selfish mining attack is successful, the selfish miners own a higher fraction of the blocks on the main chain because some portion of the blocks created by the honest network go to waste.

Eyal et al. [10] modeled selfish mining using a Markov chain. Then Sapirshtein et al. [20] created a more complex model using a Markov decision process (MDP) for selfish mining and computed the $\epsilon$-optimal policy that increases a selfish miner's revenue. Recently, Gervais et al. [12] incorporated additional parameters such as network conditions and Bitcoin settings into the MDP to study the affects of such parameters on the attacker's policy.

Sapirshtein et al. [20] first observed that some double-spend attacks can be carried out essentially cost-free in the presence of a concurrent selfish mining [10] attack. More recent work extends the scope of double-spends that can benefit from selfish mining to cases where the attacker is capable of *pre-mining* blocks on a secret branch at little or no opportunity cost [22]. The papers identify the optimal mining strategy for an attacker and quantify the advantage that he can expect to have over the merchant in terms of pre-mined blocks. This analysis is complementary to ours; it is possible to relatively easily incorporate the pre-mining advantage into our model by simply changing the attacker's block target from $z$ to $z - c$. We note that pre-mining in the context of the eclipse attack may not be feasible since an eclipse cannot generally be carried out for an indefinite period of time. Nevertheless, we intend to update both of our double-spend analyses to account for cost-free pre-mining in future work.

Gervais et al. [12] is the work most directly related to our objectives. Rosenfeld [18] also has the same economic objective. As we discuss in Section **??**, in general, the approach taken by past work, including Gervais et al., Rosenfeld, and the works cited above, models only the *order* of block creation, which is a discrete process; they do not model block mining time, which is a continuous process. As a result, it is difficult to extend those results to model cost in circumstances where the attacker is given a specific deadline in time (as we have done in our eclipse attack analysis) or where an attacker drops out (because the honest miners have already won) but has spent time mining. We develop a richer, continuous-time model that explicitly accounts for attacker cost as a function of mining duration.

## 4.2 Problem Statement

In this project, we study a form of adversarial behavior in cryptocurrencies. Known as selfish mining, this behavior involves the deliberate withholding of transaction information. We study whether an attacker who wants to perform selfish mining could use a reinforcement learning method to search for the optimal strategy. Furthermore, we compare the optimal policy found by RL algorithms to those cited in previous work.

**Parametrization of $\beta$.** Let $X = X_1, \ldots, X_n$, where $X \sim \mathrm{Exp}(\beta)$, where $\beta = 1/\lambda$. The MLE estimator for $\beta$ is

$$\hat{\beta} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{4.1}$$

The expected value of the estimator is

$$\mathbb{E}[\hat{\beta}|\beta] = \frac{1}{n}\,\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] \tag{4.2}$$

$$= \frac{1}{n} n\beta \tag{4.3}$$

$$= \beta \tag{4.4}$$

This estimator *not* biased. The bias is

$$\text{bias}(\hat{\beta}) = \mathbb{E}[\hat{\beta}|\beta] - \beta \tag{4.5}$$

$$= \beta - \beta \tag{4.6}$$

$$= 0 \tag{4.7}$$

The variance of the estimator is

$$\text{Var}(\hat{\beta}) = \text{Var}\left(\frac{\sum_{i=1}^{n} X_i}{n}\right) \tag{4.8}$$

$$= \frac{1}{n^2}\text{Var}\left(\sum_{i=1}^{n} X_i\right) \tag{4.9}$$

$$= \frac{1}{n^2} n\beta^2 \tag{4.10}$$

$$= \frac{1}{n}\beta^2 \tag{4.11}$$

Ideally, in Bitcoin $\beta = 1/10$ mins and in Ethereum $\beta = 1/15$ secs. Note that the variance increases with more data (more inter-arrival times).

**Understanding the Relationship Between Hash Rate and $\beta$.** Given target $T_i$, the expected number of hashes, $h$, needed to meet the target for a block is

$$\mathbb{E}[h] = \frac{2^{256} - 1}{T_i} \tag{4.12}$$

However, $\mathbb{E}[h]$ describes the *total* number of expected hashes needed to find a block. We have observations regarding the *time* it takes to generate a block. Let $r$ be the hash rate of the network in minutes (or the number of hashes per time unit), and $X = X_1, \ldots, X_n$, where $X \sim \text{Exp}(\beta)$, where $\beta = 1/\lambda$. Note that $\lambda r$ is the expected number of hashes each time a block is created.

$$\mathbb{E}[h] = r\lambda = r\frac{1}{\beta} \tag{4.13}$$

$$r = \mathbb{E}[h]\beta \tag{4.14}$$

**Expected Value of Hash Rate.**

$$\begin{aligned}
\mathbb{E}[r|T_i, X_1, ..., X_n] &= \mathbb{E}\left[\frac{(2^{256}-1)\hat{\beta}}{T_i}\right] && (4.15)\\
&= \frac{(2^{256}-1)}{T_i}\,\mathbb{E}[\hat{\beta}] && (4.16)\\
&= \frac{(2^{256}-1)\beta}{T_i} && (4.17)
\end{aligned}$$

**Variance of Hash Rate.**

$$\begin{aligned}
\text{Var}(r|T_i, X_1, ..., X_n) &= \text{Var}\left(\frac{(2^{256}-1)\hat{\beta}}{T_i}\right) && (4.18)\\
&= \frac{(2^{256}-1)^2}{T_i^2}\text{Var}(\hat{\beta}) && (4.19)\\
&= \frac{(2^{256}-1)^2\beta^2}{T_i^2 n} && (4.20)
\end{aligned}$$

**Bias of Hash Rate.**

$$\begin{aligned}
\text{bias}(r|T_i, X_1, ..., X_n) &= \mathbb{E}[r|T_i, X_1, ..., X_n] - r && (4.21)\\
&= \frac{(2^{256}-1)\beta}{T_i} - \frac{(2^{256}-1)\beta}{T_i} && (4.22)\\
&= 0 && (4.23)
\end{aligned}$$

The variance for estimating the miner hash rate is huge! You'd require more blocks than those that are in the blockchain to reduce variance! Therefore, let's use RL.

**Example.** The current Bitcoin difficulty, $D_i$, is $1,590,896,927,258 \approx 2^{40}$. This means the current target, $T_i$, is

$$T_i = \frac{2^{224}}{D_i} = \frac{2^{224}}{2^{40}} \tag{4.24}$$

$$\approx 2^{184} \tag{4.25}$$

Then approximately the variance associated with the hash rate is

$$\mathrm{Var}(r|T_i, X_1, ..., X_n) = \frac{(2^{256}-1)^2\beta^2}{T_i^2 n} \tag{4.26}$$

$$\approx \frac{(2^{256})^2\beta^2}{T_i^2 n} \tag{4.27}$$

$$\approx \frac{(2^{256})^2\beta^2}{(2^{184})^2 n} \tag{4.28}$$

$$\approx \frac{2^{512}\beta^2}{2^{368}n} \tag{4.29}$$

$$\approx 2^{144}\frac{\beta^2}{n} \tag{4.30}$$

A huge number no matter what $\beta$ or $n$ is!

## 4.3 Preliminary Work

### 4.3.1 Formulating the Problem as an MDP

**States.** Using Sapirshtein et al. [20]'s model as a basis, we construct the following MDP that is a 6-tuple $\{S, A, P, R, \gamma, d_0\}$, where $S = \{(x, y, z)\}$ such that $x, y, z \in \mathbb{N}$. The state consists of a 3-tuple where $x$ denotes the number of blocks on the attacker's hidden chain, $y$ denotes the number of blocks created by the honest miners on the main chain, and $z$ denotes the number of blocks that the attacker released on the main chain. Note that this representation assumes that all blocks build on the same parent block.

**Actions.** $A = \{\texttt{adopt}, \texttt{mine}, \texttt{override}, \texttt{match}\}$. `adopt` refers to the adoption of the main chain, thereby discarding all blocks created by the attacker. The action `mine` denotes that the attacker continues to mine, waiting to see who the next block will be discovered by. `override` refers to an attacker's releasing one more block than the honest miners' blocks on the main chain. This action can be viewed as honest or selfish depending on the current state. If the honest miners have no blocks on the main chain, an addition of a block to the main chain means that the attacker is honest. However, if the honest miners already have blocks on the main chain and the attacker releases an alternative chain that is 1 block longer than that created by the honest miners, then the attacker overwrites the main chain, wasting the victim's computational resources. The `match` action means that the attacker releases as many blocks as there are on the main chain, causing a bifurcation.

**Initial state distribution.** $d_0 = \{(0, 0, 0)\}$, where $P(S_0 = (0, 0, 0)) = 1$. In other words, the start state assumes that no blocks have been mined yet. When the attacker chooses the action `adopt`, the agent goes back to the start state because a new parent block is chosen to build on top of.

**Transition Function.** We consider 3 of parameters of interest included in Gervais et al. [12]'s model. At each time step, a new block is created by the network: with probability $q$, where $q$ is the mining power of the attacker, the attacker is the winner of a new block. The honest network discovers a block with probability $1 - q$. Not all actions are available in every state. The attacker can always choose the `mine` action. The `adopt` action is only available when there are blocks on the main chain created by the honest network. Additionally, `override` and `match` are only available when the attackers have enough blocks. At cutoff, $c = 30$, we force the attacker to choose the `adopt` action in order to have episodic trials. If the number of blocks created by the honest miners exceeds those of the attacker by $c$, we force the attacker to end the episode. Our third parameter, $a$, represents network connectivity. If there is a fork of same length on the main chain, fraction $a$ of the honest miners build on the attacker's alternative chain. We set $a = 1$ to give advantage to the attacker, and to see if RL methods can learn to take advantage of the `match` action. If the attacker matches the main chain with a fork of the same length, all honest miners build on the attacker's chain.

## 4.4 Proposed Work

# CHAPTER 5

# TIMELINE

# BIBLIOGRAPHY


[1] Andresen, Gavin. O(1) Block Propagation. `https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2`, August 2014.

[2] Confirmation. `https://en.bitcoin.it/wiki/Confirmation`, February 2015.

[3] Bloom, Burton H. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM 13*, 7 (July 1970), 422–426.

[4] Bonneau, Joseph. How long does it take for a bitcoin transaction to be confirmed? `https://coincenter.org/2015/11/what-does-it-mean-for-a-bitcoin-transaction-to-be-confirmed/`, November 2015.

[5] Corallo, Matt. Bip152: Compact block relay. `https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki`, April 2016.

[6] Difficulty. `http://learnmeabitcoin.com/guide/difficulty`, Feb 2015.

[7] Douceur, J. The Sybil Attack. In *Proc. Intl Wkshp on Peer-to-Peer Systems (IPTPS)* (Mar. 2002).

[8] Eppstein, David, Goodrich, Michael T., Uyeda, Frank, and Varghese, George. What's the Difference?: Efficient Set Reconciliation Without Prior Context. In *ACM SIGCOMM* (2011).

[9] Ethash. `https://github.com/ethereum/wiki/wiki/Ethash`, Last retrieved June 2017.

[10] Eyal, Ittay, and Sirer, Emin Gün. Majority is not enough: Bitcoin mining is vulnerable. *Financial Cryptography* (2014), 436–454.

[11] Feller, William. *An Introduction to Probability Theory and its Applications: Volume I*, 3 ed. John Wiley & Sons London-New York-Sydney-Toronto, 1968.

[12] Gervais, Arthur, O. Karame, Ghassan, Wust, Karl, Glykantzis, Vasileios, Ritzdorf, Hubert, and Capkun, Srdjan. On the Security and Performance of Proof of Work Blockchains. `https://eprint.iacr.org/2016/555`, 2016.

[13] Goodrich, M.T., and Mitzenmacher, M. Invertible bloom lookup tables. In *Conf. on Comm., Control, and Computing* (Sept 2011), pp. 792–799.

[14] Hanke, Timo. A Speedup for Bitcoin Mining. http://arxiv.org/pdf/1604.00575.pdf (Rev. 5), March 31 2016.

[15] Hashcash. https://en.bitcoin.it/wiki/Hashcash, Last retrieved June 2017.

[16] Heilman, Ethan, Kendler, Alison, Zohar, Aviv, and Goldberg, Sharon. Eclipse Attacks on Bitcoin's Peer-to-peer Network. In *USENIX Security* (2015).

[17] Nakamoto, Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System, May 2009.

[18] Rosenfeld, Meni. Analysis of hashrate-based double-spending. https://bitcoil.co.il/Doublespend.pdf, December 2012.

[19] Russel, Rusty. Playing with invertible bloom lookup tables and bitcoin transactions. http://rustyrussell.github.io/pettycoin/2014/11/05/Playing-with-invertible-bloom-lookup-tables-and-bitcoin-transactions.html, Nov 2014.

[20] Sapirshtein, Ayelet, Sompolinsky, Yonatan, and Zohar, Aviv. Optimal Selfish Mining Strategies in Bitcoin. https://arxiv.org/pdf/1507.06183.pdf, July 2015.

[21] Sompolinsky, Yonatan, and Zohar, Aviv. Secure high-rate transaction processing in Bitcoin. *Financial Cryptography and Data Security* (2015).

[22] Sompolinsky, Yonatan, and Zohar, Aviv. Bitcoin's Security Model Revisited. https://arxiv.org/abs/1605.09193, May 2016.

[23] Tschipper, Peter. BUIP010 Xtreme Thinblocks. https://bitco.in/forum/threads/buip010-passed-xtreme-thinblocks.774/, Jan 2016.

[24] Vukolić, Marko. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *Wkshop Open Problems in Network Security* (2015), pp. 112–125.