Optimizing Governed Blockchains for Financial Process Authentications

Leif-Nissen Lundbæk, Andrea Callia D'Iddio, and Michael Huth
Department of Computing, Imperial College London
London, SW7 2AZ, United Kingdom
{leif.lundbaek, a.callia-diddio14, m.huth}@imperial.ac.uk

Abstract

We propose the formal study of governed blockchains that are owned and controlled by organizations and that neither create cryptocurrencies nor provide any incentives to solvers of cryptographic puzzles. We view such approaches as frameworks in which system parts, such as the cryptographic puzzle, may be instantiated with different technology. Owners of such a blockchain procure puzzle solvers as resources they control, and use a mathematical model to compute optimal parameters for the cryptographic puzzle mechanism or other parts of the blockchain. We illustrate this approach with a use case in which blockchains record hashes of financial process transactions to increase their trustworthiness and that of their audits. For Proof of Work as cryptographic puzzle, we develop a detailed mathematical model to derive MINLP optimization problems for computing optimal Proof of Work configuration parameters that trade off potentially conflicting aspects such as availability, resiliency, security, and cost in this governed setting. We demonstrate the utility of such a *mining calculus* by solving some instances of this problem. This experimental validation is strengthened by statistical experiments that confirm the validity of random variables used in formulating our mathematical model. We hope that our work may facilitate the creation of *domain-specific* blockchains for a wide range of applications such as trustworthy information in Internet of Things systems and bespoke improvements of legacy financial services.

1 Introduction

There is little doubt that modern accounting systems have benefitted, ever since the advent of commercial computing machines, from the digitization of the processing and recording of financial transactions. The automated processing of payroll information in the 1950ies was perhaps one of the earliest examples of such benefits: IBM introduced its *702 Data Processing System* for businesses in 1953. And the use of RFID technology or smart phones for contactless payment of small items such as coffees is a more recent example thereof.

It is then striking that the mechanisms used for managing the integrity of accounts are, in essence, those developed at least a thousand years ago. What we call the modern *double-entry bookkeeping* was already used by Florentine merchants in the 13th century, for example. Without going into great detail, the key idea is in simplified terms that each account has an associated *dual* account and that each credit in one account is recorded as a debit in that dual account. This allows for the formulation and verification of an important *financial invariant*: no matter how complex financial transactions may be, or how many transactions may occur, it must always be the case that over the totality of accounts

> "Assets equal liabilities plus capital."

Modern realizations of this method may enrich account entries with time stamps and other contextual data so that the flow of assets can be better understood, for example to support an audit. The above invariant may be quite simple to verify, and its verification may give us reassurance that every debit has an associated credit. But it does not prevent the recording of transactions that may be unauthorized, fraudulent, or that may be incorrect due to human error. For example, transaction records within accounting books may be manipulated to commit fraud whilst these manipulations still satisfy the above invariant.

One may say that processing of transactions is governed by a form of *legal code* that is informed by policy on fraud prevention and detection, regulation, compliance, risk, and so forth. But the enforcement of such legal code within the *technical code* that operationalizes modern financial processes has been difficult at best, and too costly or impossible at worst.

Digitized financial processes can, of course, utilize cryptographic primitives to help with narrowing this gap between legal and technical code: digital signatures can be associated to transactions (for example embedded within transaction objects), and commitment schemes can be used to realize consistent distributed storage whose consistency is resilient to adversarial manipulation; see for example the discussion of Byzantine Agreement Protocols in [16]. But the advent of de-centralized, *eventual consistency* storage protocols, as pioneered in the cryptocurrency Bitcoin [12], opened up a new way of thinking about the processing of financial transactions, even of creating and managing a currency as a unit of account. There is little doubt that cryptocurrencies are one of the most important innovations [1, 2], along with the invention and introduction of central banks, in financial services since the advent of the double-entry bookkeeping.

In Bitcoin, transactions are grouped into blocks, and blocks are recorded and linked in a chain of blocks – the blockchain. Currency units are created through the solution of a cryptographic mining puzzle, a process in which network nodes (called miners) compete in determining the next block to be added to the chain and where the winner will become the owner of the currency created in that new block. These solutions causally link this new block to the last one, using cryptographic hash functions, creating thus the resiliency of this chain against manipulation. These acts of creating currency are treated as (special) transactions and their outputs are associated with their owners in a (pseudo)anonymous manner – using public-key cryptography. The system dynamics is adjusted so that, on average, a new block is added to the blockchain about every 10 minutes.

In Bitcoin, transactions are now technical code. Without going into the technical details, a transaction can be seen as a program that has a number of inputs and a number of outputs, each representing a unit of currency associated with some owner. That program also contains a *proof* that the program is authorized to rewire inputs to outputs in this manner. For example, this proof may be a cryptographic demonstration that the program *owns* all inputs. Transactions enjoy an important invariant: the sum of currency units of all transaction outputs cannot be larger than the sum of currency units of all its inputs – and any positive difference becomes a credit for the miner that created the block within which the transaction is found.

The Bitcoin network has many nodes that contain a full copy of the blockchain. More accurately, each node contains a tree of possible chains and identifies one of those chains deterministically as the *real* blockchain. The propagation of possible winning blocks and consensus protocols between these nodes ensure that nodes eventually agree on which of the

chains they each store is the *real* chain. This does have its problems. For example, miners may collude to gain more control of mining competitions and so may force all network nodes to accept an alternative blockchain as the real one – allowing them to rewrite the transaction history. Another problem is that nodes need to keep a record of which outputs of transactions recorded on the *real* blockchain have not been spent yet in new transactions on the blockchain; and the above consensus mechanisms for *eventual consistency* don't give hard guarantees, meaning that transaction outputs may be double spent and that transactions in a blockchain may only be trusted if their block has been linked with a certain number of blocks added to the chain subsequently. The incentive mechanism of Bitcoin also led to the creation of puzzle-solving pools of miners and a complex and risky dynamics between such pools, the development of hardware for solving such puzzles and other factors. Understanding such dynamics in predictable ways is one research challenge for cryptocurrencies [5].

Another risk resides in the verification of transactions, done by a run-time system that executes a verification script. Whilst the designer(s) of Bitcoin deliberately chose a simple scripting language that makes its execution more secure, early implementations of that system still had security vulnerabilities. There is also a tradeoff here between security and expressiveness of technical code. Other cryptocurrencies seek to program more complex transactions (so called *smart contracts*) to create new financial services, for example. But these require scripting languages that are Turing complete and are therefore much more at risk of security attacks – see the DAO hack of Ethereum and the way in which this was dealt with as a good example thereof [7].

The risks of cryptocurrencies discussed above suggest that there is value in also exploring alternative approaches, in which technical code for transactions is more centralized, governed or controlled by parties with specific interests or duties. For example, RSCoin [6] is proposed as a cryptocurrency in which central banks control monetary supply and where a number of distributed authorities prevent double-spending attacks. This addresses or mitigates risks of de-centralized, open cryptocurrencies but does support a more conventional model of currency control. There is also the question of whether a central bank would want to risk running any such system to scale up a currency nationally, given that such technical code may be subject to security vulnerabilities in configuration, implementation or lifecycle management. But such risks may be manageable. The oldest central bank, Sweden's Riksbank, for example, is actively considering whether or not to introduce a national cryptocurrency *ekrona* within the next two years [10].

In this paper, we investigate how governed, closed blockchains can be designed so that they can support the resilient, distributed, and *trustworthy* storage of authentication of transactions within conventional financial processes. Such governed systems with restricted access give us better control on balancing the use of energy for puzzle solving with the security of the Proof of Work algorithm when compared with open systems that rely on Proof of Work, such as Bitcoin. Specifically, we propose that transactions (in the sense of Bitcoin) within blocks are hashes of transactions (within conventional financial processes). We then define mathematical models that describe the design space of such a blockchain in terms of the cryptographic puzzle used – in this paper Proof of Work, in terms of expected availability, resiliency, security, and cost, and in terms that reflect that the system is centrally governed.

We stress that our approach is also consistent with transactions within blockchains that encode transaction history, which we don't consider in the use case of this paper. We believe

that our approach has potential. It may, for example, allow designers to minimize the need for consensus mechanism by guaranteeing that puzzles have a unique winner within a certain period of time with very large probability whilst still maintaining sufficient system resiliency and security; and this could inform the design of bespoke consensus protocols.

Outline of paper. In Section 2 we present our use case. Our mathematical model for Proof of Work in our setting is subject of Section 3. The derivation of optimization problems for these mathematical models is done in Section 4. An algorithm for solving such optimization problems, experimental results, and a statistical validation of our model are reported in Section 5, and the paper concludes in Section 6.

2 Use case

The use case we consider is one of a financial process that creates financial transactions. We would like to enhance the trustworthiness of this process through a blockchain that records hash-based authentications of transactions, as seen in Figure 1, where the interaction between the legacy process and the blockchain is conceptually simple – and consistent with the use of double-entry bookkeeping if desired. Our assumption is that the event streams of such transactions are not linearizable and so we cannot rely on techniques such as hash chains [8] to obtain immutability of transactions.

Our data model represents a transaction as a string *input* that can be authenticated with a hash $hash(input)$. String *input* may be a serialization of a transaction object that contains relevant information such as a time stamp of the transaction, a digital signature of the core transaction data and so forth. The trustworthiness of transaction *input* is represented outside of the blockchain by the triple

$$(input, hash(input), location) \tag{1}$$

where *location* is either the block height (≥ 0) of a block b in the blockchain such that $hash(input)$ occurs in block b or *location* is NULL, indicating that the transaction is not yet confirmed on the blockchain.

The hashes $hash(input)$ of transactions that still need to be confirmed are propagated on the blockchain network, where they are picked up by miners and integrated into blocks for Proof of Work. We assume a suitable mechanism by which nodes that manage legacy accounts learn the blockheights of their transactions that have been successfully added to the blockchain. For example, such nodes may have a full copy of the blockchain and so update *location* values in its accounts if the hash of the corresponding transaction occurs in a block that was just added.

A transaction is unverified if its *location* value is NULL or if its hash does not equal the one stored externally as in (1); it is trustworthy if $0 \leq location$ and $location + k \leq currentBlockHeight$ where $k \geq 0$ is a suitable constant and $currentBlockHeight$ denotes the number of blocks added to the blockchain so far. The value of k may be a function of how fast blocks are added to the chain on average, to ensure sufficient resiliency of trustworthiness. An auditor could then inspect any transaction by examining its triple stored as in (1). If *location* equals NULL or if $location + k > currentBlockHeight$, the transaction is considered neither

valid nor trustworthy by the auditor. Otherwise, we have $0 \leq location$ and $location + k \leq currentBlockHeight$ and the auditor uses the Merkle tree hash in block $location$ to verify that $hash(input)$ is in the block of height b. If that is the case, the auditor considers the transaction to be verified; otherwise, the auditor considers the transaction not to be trustworthy.

2.1 System Architecture

A system architecture that could support such a use case is shown in Figure 1. Unverified transactions have their hashes propagated on the network. Miners pick up those hashes and integrate them into blocks for Proof of Work. We abstract away how miners manage their pools of hashes and how Proof of Work blocks are propagated and added to the blockchain; this gives us flexibility in the use of blockchain technology. Once blocks are added to the blockchain, blockheights are propagated to the legacy account. As mentioned above, these accounts could have full copies of the blockchain and thus implement their own update mechanisms for value *location* in triples stored as in (1).

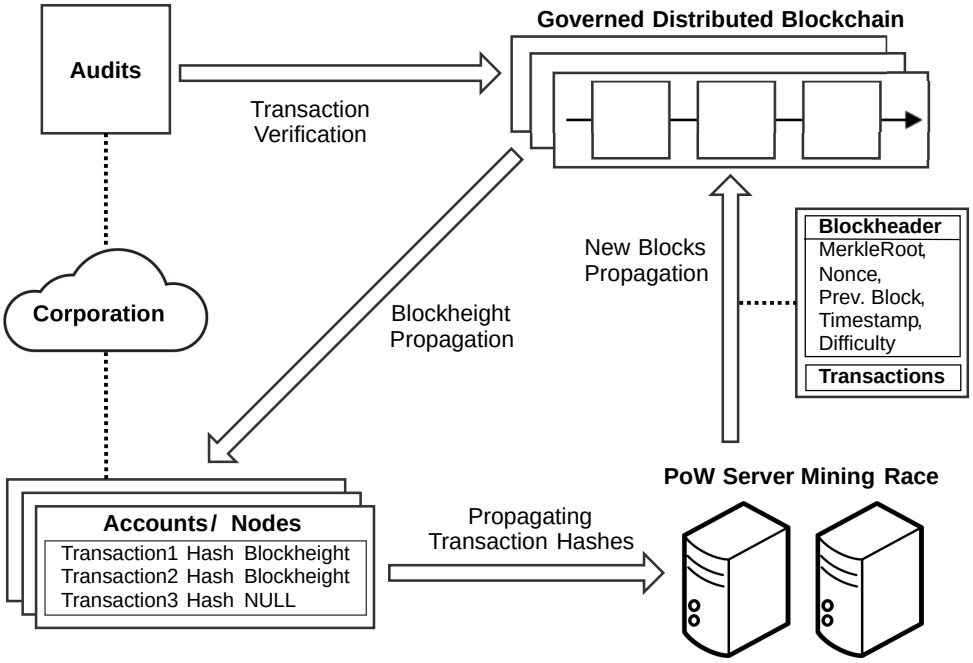

Figure 1: Governed blockchain for financial process authentications

Auditors would interface with both accounts and the blockchain to verify, in a trustworthy manner, the authenticity, that is to say the *veracity*, of transactions. Any transaction that is not verified as discussed above would be flagged up in this audit. Any pre-existing audit process – which may focus on compliance, regulations and other aspects – is consistent with such *veracity checking*; and the trustworthiness of the pre-existing audit process would be increased as it would refuse to certify any financial transaction histories that involved a transaction that is not authenticated on the blockchain.

2.2 Discussion

The approach we advocate here is pretty flexible. It seems consistent with consensus mechanisms as used in Bitcoin but it may also support 2-phase commitment schemes as proposed in [6]. Our system architecture allows for full nodes to be associated with accounts, sets of accounts or corporate boundaries. Our blockchain does not create any currency, and so there is no inherent incentive to mine. But there is an incentive for the owners of this blockchain to allocate mining resources in a manner that establishes trustworthiness of transactions as recorded in this blockchain. We think that the elimination of incentives and their game-theoretic implications are a benefit, as are the relatively simple ways of propagating trust through hashes of transactions. Such a blockchain may also be consulted by legacy systems to inform the authorization of further financial transactions.

Our blockchain does not spend any funds and so has no problem of *double spending*, and double spending in the legacy system would be detectable with existing mechanisms such as audits. Our approach does allow for *double authentication* though: a transaction hash may occur more than once in a blockchain, be it in the same block or in different blocks. We deem this to be unproblematic as audits would only need to establish *some*, sufficiently old, authentication of the transaction in the blockchain to establish its trustworthiness – noting that hash-based authentication is deterministic.

Let us now briefly reflect on security issues for the system shown in Figure 1. There are various actors within that system: miners, nodes that run full copies of the blockchain, nodes that propagate transaction hashes to miners, auditors (which may be insiders or external agents with limited inside access), insiders such as accountants and system administrators, external forces that seek to infiltrate these corporate networks, and so forth. Here are possible scenarios of interest:

S1 internal auditors may want to corrupt the state of the blockchain in order to cover up the traces of internal fraudulent activity

S2 external forces may want to corrupt transaction hashes propagated in the internal network as a denial of service attack on the blockchain itself

S3 control over a set of miners or nodes that propagate transaction hashes may be obtained so that certain transaction hashes have priority for mining

S4 classical security attacks such as those on key management may be launched.

Scenario S1 may be part of an insider attack in which skilled and sufficiently authorized insiders collude to commit fraud. The blockchain cannot ensure that only legitimate transactions have their hash recorded within it. But a *security policiy* – external to the blockchain – could specify that certain transactions are always recorded in the blockchain, for example, the hashes of security log entries that may be triggered by transactions within legacy accounts.

Scenario S2 is typical for a denial of service attack. The service that is being denied here is the blockchain, as the mechanism that creates sufficient trustworthiness into transactions as recorded within legacy system. These threats can be mitigated against, for example, legacy systems and the entire blockchain network may be placed behind a corporate firewall.

Scenario S3 is concerned with the management of transaction hashes that have not yet been recorded in the blockchain. The extent to which this is a security problem beyond that of service availability will depend on the use context of the legacy accounts. For example, for the attack discussed in scenario S1 it might help internal attackers to have control over which transaction hashes would enter the blockchain first, with a preference of recording the fraudulent transaction that changed the recipient of payments of the originally recorded transaction.

In scenario S4, a cyber attack may get control of the part of the system that provides authenticated information about the public keys of all used miners, for example, it might be able to learn a system admin key that allows for the modification of such information to change the private/public key pairs of miners to values of machines controlled by the attacker.

Security would certainly be improved in these and other scenarios if all actions of financial processes that modify accounts have their hashes recorded onto the blockchain, be these financial transactions, actions that create log entries, actions that give authority to perform financial transaction, and so forth. From this perspective, our blockchain could also facilitate a *forensic* audit – not just one concerned with compliance and regulation.

3 Probabilistic Model

Our model assumes a cryptographic hash function $h\colon \{0,1\}^p \to \{0,1\}^n$ where $p \geq n > 0$ such that h has *puzzle friendliness* [2]. The *level of difficulty* d is an integer satisfying $0 < d < n$: Proof of Work has to produce some x where $h(x)$ has at least d many leftmost 0 bits. We write $T > 0$ for the time to compute a sole hash $h(x)$ and to decide whether it has at least d leftmost zeros. Since the range of d will be relatively small, we make T a device-dependent constant.

Our probabilistic modeling will treat h in the *Random Oracle Model* (ROM): function h is chosen uniformly at random from all functions of type $\{0,1\}^p \to \{0,1\}^n$; that is to say, h is a deterministic function such that any x for which h has not yet been queried will have the property that $h(x)$ is governed by a truly random probability distribution over $\{0,1\}^n$.

We may assume that x consists of a block header which contains some random data field – a nonce *nonce* of bitlength r, that this nonce is initialized, and that the nonce is then increased by 1 each time the hash of x does not obtain Proof of Work. In particular, this yields that $\{0,1\}^p \cong \{0,1\}^{p-r} \times \{0,1\}^r$ where $0 < r < p$: the input to h will be of form $x = \mathit{data} \,||\, \mathit{nonce}$ where *data* and *nonce* have $p-r$ and r bits, respectively. Our use of ROM will rely on the assumption that mining, be it by a sole miner or in a mining race of more than one miner, will never revisit the same input again:

Assumption 1 (Invariant) *The mining of a block with one or more miners will use an input to h at most once, be it within or across miners' input spaces.*

This assumption and appeal to ROM give us that hash values are always uniformly distributed in the output space during a mining race. We now develop the probability space for mining with a sole miner, and then adapt this to the setting of more than one miner.

3.1 Basic Probability Space for One Miner

Our basic probability space has *data* and d as implicit parameters, and assumes the enumeration $0 \dots 2^r - 1$ of values of *nonce* without loss of generality. The set of basic events E of this probability space is

$$E = \{\otimes^k \cdot \checkmark \mid 0 \leq k \leq 2^r - 1\} \cup \{\mathsf{failure}\} \tag{2}$$

where failure denotes the event that all 2^r nonce values failed to obtain Proof of Work for *data* at level of difficulty d, and $\otimes^k \cdot \checkmark$ models the event in which the first k such nonce values failed to obtain Proof of Work for *data* at level d but the $k+1$th value of *nonce* did render such Proof of Work for *data*. We next want to define a discrete probability distribution $prob\colon E \to [0,1]$ with mass 1. Now, we have $prob(\otimes^0 \cdot \checkmark) = 2^{n-d}/2^n = 2^{-d}$ since this is the fraction between the number of possible outputs that do Proof of Work at level of difficulty d and the number of all possible outputs of h.

To define $prob(\otimes^k \cdot \checkmark)$ for the case when $k > 0$, we first need to understand the probability $\tilde{p}(\otimes^k)$ of not obtaining Proof of Work for the first k values of the nonce. For each value of *nonce*, the probability that $h(data \mid\mid nonce)$ does not have d or more leading zeros is $1 - prob(\otimes^0 \cdot \checkmark) = 1 - 2^{-d}$. By ROM and Assumption 1, these probabilities are independent from each other for each of these k different values of *nonce* and so $\tilde{p}(\otimes^k) = (1-2^{-d})^k$ follows. Similarly, the probability that the $k+1$th hash attempt proves work is independent of whether or not any of the previous k attempts did that – mining never tries that same nonce value again. But then $prob(\otimes^k \cdot \checkmark)$ is $(1-2^{-d})^k \cdot 2^{-d}$. We thus defined $prob(e)$ for events e in $E \setminus \{\mathsf{failure}\}$ such that $0 < prob(e) < 1$. Also, the mass of all those probabilities is less than 1:

$$\sum_{e \in E \setminus \{\mathsf{failure}\}} prob(e) \quad = \quad \sum_{0 \leq k \leq 2^r - 1} (1-2^{-d})^k \cdot 2^{-d}$$

but this equals $1 - (1-2^{-d})^{2^r}$ and is therefore in $(0,1)$ since $0 < d, r$. Thus, we obtain a probability distribution *prob* by setting

$$prob(\mathsf{failure}) = 1 - \sum_{0 \leq k \leq 2^r - 1} prob(\otimes^k \cdot \checkmark)$$

which equals $1 - (1 - (1-2^{-d})^{2^r}) = (1-2^{-d})^{2^r}$.

3.2 Probability Space for $s > 1$ Miners

Consider having $s > 1$ many miners that run in parallel to find Proof of Work, engaging thus in a *mining race*. We assume these miners run with the same configurations and hardware. In particular, the hash function h and the values n, p, d, r, and T will be the same for each of these miners. As already discussed, miners do not get rewarded:

Assumption 2 (Miners) *Miners are a resource controlled by the governing organization and have identical hardware. In particular, miners are not rewarded nor have the need for incentive structures.*

But miners may be corrupted and misbehave, for example they may refuse to mine. To simplify our analysis, we assume miners begin the computation of hashes in approximate synchrony:

Assumption 3 (Approximate Synchrony) *Miners start a mining race at approximately the same time.*

For many application domains, this is a realistic assumption as communication delays to miners would have a known upper bound that our models could additionally reflect if needed.

Next, we want to model the *race* of getting a Proof of Work where each miner j has some data $data_j$. To realize Assumption 1, it suffices that each miner j have a nonce $nonce_j$ in a value space of size $\lfloor 2^r/s \rfloor$ such that these nonce spaces are mutually disjoint across miners. To model this mining race between s miners, we take the product $\prod_{j=1}^{s} E^j$ of s copies E^j of our event space E for mining with a sole miner, and quotient it via an equivalence relation $\equiv$ on that product $\prod_{j=1}^{s} E^j$. The s-tuple $(\mathsf{failure}, \ldots, \mathsf{failure})$ models failure of this mining race, it is $\equiv$ equivalent only to itself.

All s-tuples $a = (a_j)_{1 \leq j \leq s}$ other than tuple $(\mathsf{failure}, \ldots, \mathsf{failure})$ model that the mining race succeeded for at least one miner. For such an s-tuple a, the set of natural numbers k such that $\otimes^k \cdot \checkmark$ is a coordinate in a is non-empty and therefore has a minimum $\min(a)$. Given two s-tuples $a = (a_j)_{1 \leq j \leq s}$ and $b = (b_j)_{1 \leq j \leq s}$ both different from $(\mathsf{failure}, \ldots, \mathsf{failure})$, we can then define a and b as $\equiv$ equivalent iff $\min(a) = \min(b)$. So two non-failing tuples are equivalent if they determine a first (and so final) Proof of Work at the same round of the race. This defines an equivalence relation $\equiv$ and adequately models a synchronized mining race between s miners.

In the setting of $s > 1$ miners, the interpretation of events $\otimes^k \cdot \checkmark$ of E in (2) is then the equivalence class of all those tuples a for which $\min(a)$ is well defined and equals k: all mining races that succeed first at round k. The meaning of $\mathsf{failure}$ is still overall failure of the mining race, the equivalence class containing only tuple $(\mathsf{failure}, \ldots, \mathsf{failure})$.

Next, we set $\lambda = \lfloor 2^r/s \rfloor$ as the size of the nonce space for each of the s miners, and define accordingly the set of basic events for s miners as

$$E^s = \{\otimes^k \cdot \checkmark \mid 0 \leq k \leq \lambda\} \cup \{\mathsf{failure}\} \tag{3}$$

In (3), expression $\otimes^k \cdot \checkmark$ denotes an element of the quotient $\left(\prod_{j=1}^{s} E^j\right)/\equiv$, the equivalence class of tuple $(\otimes^k \cdot \checkmark, \mathsf{failure}, \mathsf{failure}, \ldots, \mathsf{failure})$. Also, E^s restricts the set of non-failure events from E in (2) to those with $k \leq \lambda$.

Next, we define a probability distribution $prob^s$ over E^s, consistent with the definition of $prob$ over E when s equals 1. To derive the probability $prob^s(\otimes^k \cdot \checkmark)$, recall $\tilde{p}(\otimes^k) = (1-2^{-d})^k$ as the probability that a given miner does not obtain Proof of Work at level d in the first k rounds. By Assumption 1, these miners work independently and over disjoint input spaces. By ROM, the expression $\left[(1-2^{-d})^k\right]^s = (1-2^{-d})^{k \cdot s}$ therefore models the probability that none of the s miners obtains Proof of Work in the first k rounds. Appealing again to ROM and Assumption 1, the behavior at round $k+1$ is independent of that of the first k rounds. Therefore, we need to multiply the above probability with the one for which at least one of the s miners will obtain a Proof of Work in a single round. The latter probability is the

complementary one of the probability that none of the s miners will get a Proof of Work in a sole round, which is $(1-2^{-d})^s$ due to the ROM independence. Therefore, we get

$$prob^s(\otimes^k \cdot \checkmark) = (1-2^{-d})^{k\cdot s} \cdot [1-(1-2^{-d})^s] \tag{4}$$

This defines a probability distribution with a non-zero probability of failure. Firstly, $\sum_{k=0}^{\lambda}(1-2^{-d})^{k\cdot s} \cdot [1-(1-2^{-d})^s]$ is in $(0,1)$: that sum equals

$$[1-(1-2^{-d})^s] \cdot \frac{1-[(1-2^{-d})^s]^{\lambda+1}}{1-(1-2^{-d})^s} = 1-(1-2^{-d})^{s\cdot(\lambda+1)}$$

And since $0 < d, s$, the real $1-2^{-d}$ is in the open interval $(0,1)$, and the same is true of any integral power thereof. Secondly, $prob^s$ becomes a probability distribution with the non-zero probability $prob^s(\mathsf{failure})$ being $1 - prob^e(E^s \setminus \{\mathsf{failure}\})$, that is

$$prob^s(\mathsf{failure}) = (1-2^{-d})^{s\cdot(\lambda+1)} \tag{5}$$

This failure probability basically equals that for $s = 1$, an artefact of our parameter representation: for example, if each miner has 64 bits of nonce space, then our model would have $r = 64 \cdot s$, so failure probabilities do decrease as s increases.

4 Mathematical Optimization in Mining Design Space

4.1 Generality of Approach

We want to optimize the use of $s > 1$ miners using a level of difficulty d, and a bit size r of the global nonce space with respect to an objective function. The latter may be a cost function, if containing cost is the paramount objective or if a first cost estimate is sought that can then be transformed into a constraint to optimize for a security objective, as seen further below.

Higher values of d add more security: it takes more effort to mine a block and so more effort to manipulate the mining process and used consensus mechanism. But lower values of d may be needed, for example, in high-frequency trading where performance can become a real issue. We want to understand such trade-offs.

Moreover, we want to explore how the corruption of a number of miners or inherent uncertainty in the number of deployed miners or in the level of difficulty across the lifetime of a system may influence the above tradeoffs. We will use tools from robust optimization [3] and functional programming to analyze such issues.

4.2 Optimizing Cost and Security

The flexibility of our approach includes the choice of objective function for optimization. Let us first consider an objective function

$$\mathrm{Cost}(s,r,d) = \mathrm{TVC} \cdot E^s(noR) \cdot s + \mathrm{TFC} \cdot s \tag{6}$$

$$
\begin{array}{rcl}
0 & < & s_l \leq s \leq s_u \qquad 0 < d_l \leq d \leq d_u \qquad 0 < r_l \leq r \leq r_u \qquad \epsilon \geq prob^s(\mathsf{failure}) \\
\tau_u & \geq & T \cdot E^s(noR) \geq \tau_l \qquad\qquad \delta_2 \geq prob^s(disputes\ within\ \mu) \\
\delta & \geq & prob^s(PoWTime > th) \qquad \delta_1 \geq prob^s(PoWTime < th')
\end{array}
$$

Figure 2: Constraint set $\mathcal{C}$ for optimization problems: (a) *minimize* $\mathrm{Cost}(s, r, d)$ as in (6) subject to constraints in $\mathcal{C}$; and (b) *maximize* d subject to $\mathcal{C} \cup \{\mathrm{Cost}(s, r, d) \leq budget\}$ for cost bound *budget*. This is parameterized by constants $0 \leq \delta, \delta_1, \delta_2, \epsilon, th, th', \tau_l, \mathrm{TVC}, \mathrm{TFC}$ and $0 < T, s_l, r_l, d_l$. Variables or constants $s_l, s_u, s, d_l, d_u, d, r_l, r_u, r$ are integral

that models cost as a function of the number of miners s, the bit size of the nonce r – implicit in random variable $E^s(noR)$, and the level of difficulty d; where we want to *minimize* cost.

The real variable TVC models the *variable* cost of computing *one* hash for *one* miner, reflecting the device-dependent speed of hashes and the price of energy. The real variable TFC models the *fixed* costs of *having one miner*; this can be seen as modeling procurement and depreciations. Variables s, r, and d are integral, making this a *mixed integer* optimization problem [9]. The expression $E^s(noR)$ denotes the *expected number of rounds* needed to mine a block in a mining race that uses s miners, level of difficulty d, and nonce bitsize r. The derivation of this expression below shows that it is non-linear, making this a MINLP optimization problem [14, 9]. We chose not to include in TVC a constant that reflects how many blocks may be mined within a system horizon of interest but this can easily be done within our proposed approach, for example when there is a constraint on the carbon footprint of a system during its lifetime.

We may of course use other objective functions. One of these is simply the expression d, which we would seek to *maximize*, the intuition being that higher values of d give us more trust into the veracity of a mined block and the blockchains generated in the system. Figure 2 shows an example of a set of constraints and optimizations of security and cost for this.

Integer constants s_l and s_u provide bounds for variable s, and similar integer bounds are used to constrain integer variables r and d. The constraint for ϵ uses it as upper bound for the probability of a mining race failing to mine a block. The next two inequalities stipulate that the expected time for mining a block is within a given time interval, specified by real constants τ_l and τ_u.

The real constant δ_2 is an upper bound for $prob^s(disputes\ within\ \mu)$, the probability that more than one miner finds PoW within μ seconds in the same, synchronous, mining race. The constraint for real constant δ says that the probability $prob^s(PoWTime > th)$ of the *actual* time for mining a block being above a real constant th is bounded above by δ. This constraint is of independent interest: knowing that the expected time to mine a block is within specified bounds may not suffice in systems that need to assure that blocks are *almost always* (with probability at least $1 - \delta$) mined within a specified time limit. Some systems may also need assurance that blocks are almost always mined in time *exceeding* a specified time limit th'. We write $prob^s(PoWTime < th')$ to denote that probability, and add a dual constraint specifying that the actual time for mining a block has a sufficiently small probability $\leq \delta_1$ of being faster than some given threshold th'.

4.3 Constraints as Analytical Expressions

We derive analytical expressions for random variables occurring in Figure 2. Beginning with $E^s(noR)$, we have $E^s(noR) = \sum_{0 \leq k \leq \lambda} prob^s(\otimes^k \cdot \checkmark) \cdot (k+1)$ which we know to be equal to $\sum_{0 \leq k \leq \lambda} (1-2^{-d})^{k \cdot s} \cdot [1-(1-2^{-d})^s] \cdot (k+1)$. We may rewrite the latter expression so that summations are eliminated and reduced to exponentiations: concretely, we rewrite $\sum_{0 \leq k \leq \lambda} prob(\otimes^k \cdot \checkmark) \cdot (k+1)$ to $\lambda + 1$ summations, each one starting at a value between 0 and λ, where we exploit $\sum_{k=a}^{b} x^k = \frac{x^a - x^{b+1}}{1-x}$. This renders

$$E^s(noR) = \frac{1 - y^{\lambda+1} - (\lambda+1) \cdot (1-y) \cdot y^{\lambda+1}}{1-y} \tag{7}$$

where we use the abbreviation

$$y = (1-2^{-d})^s \tag{8}$$

The expected time needed to get a proof of work for input *data* is then given by

$$E^s(poW) = T \cdot E^s(noR) \tag{9}$$

We derive an analytical expression for the above probability $prob^s(PoWTime > th)$ next. Note that $(th/T) - 1 < k$ models that the actual time taken for $k+1$ hash rounds is larger than th. Therefore, we capture $prob^s(PoWTime > th)$ as

$$\begin{aligned} \sum_{\lceil (th/T)-1 \rceil < k \leq \lambda} prob^s(\otimes^k \cdot \checkmark) &= \\ \sum_{\lceil (th/T)-1 \rceil < k \leq \lambda} (1-2^{-d})^{k \cdot s} \cdot [1-(1-2^{-d})^s] &= \\ (1-2^{-d})^{s \cdot (\lceil (th/T)-1 \rceil + 1)} - (1-2^{-d})^{s \cdot (\lambda+1)} &= \\ y^{\lceil (th/T)-1 \rceil + 1} - y^{\lambda+1} & \end{aligned} \tag{10}$$

assuming that $\lceil (th/T) - 1 \rceil < \lambda$, the latter therefore becoming a constraint that we need to add to our optimization problem. One may be tempted to choose the value of δ based on the Markov inequality, which gives us

$$prob^s(PoWTime \geq th) \leq T \cdot E^s(noR)/th$$

But we should keep in mind that upper bound $T \cdot E^s(noR)/th$ depends on the parameters s, r, and d; for example, the analytical expression for $E^s(noR)$ in (7) is dependent on λ and so dependent on r as well. The representation in (10) also maintains that expression

$$y^{\lceil (th/T)-1 \rceil + 1} - y^{\lambda+1}$$

is in $[0,1]$, i.e. a proper probability. Since $y = (1-2^{-d})^s$ is in $(0,1)$, this is already guaranteed if $\lceil (th/T) - 1 \rceil + 1 \leq \lambda + 1$, i.e. if $\lceil (th/T) - 1 \rceil \leq \lambda$. But we already added that constraint to our model. Similarly to how we proceded for $prob^s(PoWTime > th)$, we get

$$prob^s(PowTime < th') = 1 - (1-2^{-d})^{s \cdot (\lfloor (th'/T)-1 \rfloor + 1)} = 1 - y^{\lfloor (th'/T)-1 \rfloor + 1} \tag{11}$$

$$
\begin{array}{rcl}
s_l \leq s \leq s_u \qquad d_l \leq d \leq d_u \qquad r_l \leq r \leq r_u \qquad \lambda = \lfloor 2^r/s \rfloor \\
y = (1-2^{-d})^s \qquad z = (1-2^{-d})^{\lfloor \mu/T \rfloor} \qquad 0 \leq \lfloor \mu/T \rfloor \\
\epsilon \geq y^{\lambda+1} \qquad \lceil (th/T) - 1 \rceil < \lambda \qquad 0 < \lfloor (th'/T) - 1 \rfloor \\
E^s(noR) = \dfrac{1 - y^{\lambda+1} - (\lambda+1) \cdot (1-y) \cdot y^{\lambda+1}}{1-y} \\
\tau_u \geq T \cdot E^s(noR) \geq \tau_l \qquad \delta_1 \geq 1 - y^{\lfloor (th'/T)-1 \rfloor + 1} \\
\delta \geq y^{\lceil (th/T)-1 \rceil + 1} - y^{\lambda+1} \\
\delta_2 \geq 1 - (1 - 2^{-d} \cdot [1-z])^{s-1} \cdot \Big[1 + (s-1) \cdot 2^{-d} \cdot [1-z]\Big]
\end{array}
\tag{17}
$$

Figure 3: Arithmetic version of set of constraints $\mathcal{C}$ from Figure 2, with additional soundness constraints for this representation. Feasibility of (s, r, d) and $r_u \geq r' > r$ won't generally imply feasibility of (s, r', d) due to the constraint in (17)

which needs $0 < \lfloor (th'/T) - 1 \rfloor$ as additional constraint.

To derive an analytical expression for $prob^s(disputes\ within\ \mu)$, each miner can perform $\lfloor \mu/T \rfloor$ hashes within μ seconds. Let us set $z = (1-2^{-d})^{\lfloor \mu/T \rfloor}$. The probability that a given miner finds PoW within μ seconds is

$$
\sum_{k=0}^{\lfloor \mu/T \rfloor} (1-2^{-d})^k \cdot 2^{-d} = 2^{-d} \cdot [1 - (1-2^{-d})^{\lfloor \mu/T \rfloor}] = 2^{-d} \cdot [1-z] \tag{12}
$$

Therefore, the probability that no miner finds PoW within μ seconds is

$$
prob^s(0\ PoW\ within\ \mu) = \Big(1 - 2^{-d} \cdot [1 - (1-2^{-d})^{\lfloor \mu/T \rfloor}]\Big)^s = \Big(1 - 2^{-d} \cdot [1-z]\Big)^s \tag{13}
$$

The probability that exactly one miner finds PoW within μ seconds is

$$
prob^s(1\ PoW\ within\ \mu) = s \cdot \Big(1 - 2^{-d} \cdot [1-z]\Big)^{s-1} \cdot 2^{-d} \cdot [1-z] \tag{14}
$$

Thus, the probability that more than one miner finds PoW within μ seconds is

$$
prob^s(disputes\ within\ \mu) = 1 - prob^s(0\ PoW\ within\ \mu) - prob^s(1\ PoW\ within\ \mu) \tag{15}
$$

Then we can calculate $prob^s(disputes\ within\ \mu)$ to be equal to

$$
prob^s(disputes\ within\ \mu) = 1 - (1 - 2^{-d} \cdot [1-z])^{s-1} \cdot \Big[1 + (s-1) \cdot 2^{-d} \cdot [1-z]\Big] \tag{16}
$$

Figure 3 shows the set of constraints $\mathcal{C}$ from Figure 2 with analytical expressions and their additional constraints, we add constraint $0 \leq \lfloor \mu/T \rfloor$ to get consistency for the analytical representation of $prob^s(disputes\ within\ \mu)$.

4.4 Robust Design Security

Our model above captures design requirements or design decisions as a set of constraints, to optimize or trade off measures of interest subject to such constraints. We can extend this model to also *manage uncertainty* via robust optimization [3]. Such uncertainty may arise during the lifetime of a system through the possibility of having corrupted miners, needed flexibility in adjusting the level of difficulty, and so forth. For example, corrupted miners may refuse to mine, deny their service by returning invalid block headers, pool their mining power to get more mining influence or they may simply break down.

Consider $1 \leq l < s$ corrupted miners. We can model their *pool power* by appeal to ROM and the fact that the mining race is roughly synchronized: the probability that these l miners win $c > 0$ many subsequent mining races is then seen to be $(l/s)^c$. We can therefore bound this with a constant δ_3, or equivalently we can add to the set of constraints $\mathcal{C}$ from Figure 3 the constraint $l^c \leq \delta_3 \cdot s^c$.

We model uncertainty in the number of miners available by an integer constant u_s as follows: if s miners are deployed, then we assume that at least $s - u_s$ and at most s many miners participate reliably in the mining of legitimate blocks. Therefore, u_s allows us to model aspects such as denial of service attacks or a combination of such attacks with classical faults: for example, $u_s = 3$ subsumes the scenario in which one miner fails and two miners mine invalid blocks.

Furthermore, an integer constant u_d models the uncertainty we have in the deployed level of difficulty d: the intuition is that our analysis should give us results that are robust in that they hedge against the fact that any of the values d' satisfying $|\, d - d' \,| \leq u_d$ may be the actually running level of difficulty. This enables us to understand a design if we are unsure about which level of difficulty will be deployed or if we want some flexibility in dynamically adjusting the value of d in the running system.

The corresponding robust optimization problem for cost minimization is seen in Figure 4. It adds to the constraints we already consider the requirements on constants l, c, and δ_3 as well as the constraint $l^c \leq \delta_3 \cdot s^c$. The robustness of analysis is achieved by a change of the objective function from $\mathrm{Cost}(s, r, d)$ to

$$\mathrm{Cost}^{u_s}_{u_d}(s, r, d) = \max_{s - u_s \leq s' \leq s,\, |d - d'| \leq u_d} \mathrm{Cost}(s', r, d') \tag{18}$$

The latter computes a worst-case cost for triple (s, r, d) where s and d may vary independently subject to the strict uncertainties u_s and u_d, respectively. We call a triple (s, r, d) *feasible* if it satisfies all constraints of its optimization problem. Costs such as the one in (18) for a triple (s, r, d) are only considered for optimization if all triples (s', r, d') used in the robust cost computation in (18) are feasible – realized with predicate $\mathit{feasible}^{u_s}_{u_d}$: robust optimization guarantees [3] that the feasibility of solutions is invariant under the specified uncertainty (here u_s and u_d).

5 Experiments on Mathematical Models

We submitted simple instances of the optimization problem in Figure 4 to state of the art MINLP solvers. All these solvers reported, erroneously, in their preprocessing stage that the

$$\min\{\mathrm{Cost}^{u_s}_{u_d}(s,r,d) \mid \mathit{feasible}^{u_s}_{u_d}(s,r,d)\}$$
subject to the set of constraints $\mathcal{C}$ from Figure 3 together with

$$4 = l < s \qquad c = 6 \qquad 0.001 = \delta_3$$
$$l^c \leq s^c \cdot \delta_3 \qquad u_s = 5 \qquad u_d = 3$$

Figure 4: Robust optimization that minimizes cost for the set of constraint from Figure 3, where up to $u_s = 5$ miners may be either non-functioning, refusing to mine or mining invalid blocks; where the level of difficulty may vary by up to $+3$ or -3; and where we want that the probability of any mining *pool* of size $l = 4$ winning $c = 6$ consecutive mining races is sufficiently small (here $\delta_3 = 0.001$). Predicate $\mathit{feasible}^{u_s}_{u_d}(s,r,d)$ characterizes *robustly feasible* triples and is true iff all triples (s',r,d') with $s - u_s \leq s' \leq s$ and $|d - d'| \leq u_d$ are feasible

problem is infeasible. These solvers were not designed to deal with problems that combine such small numbers and large powers, and rely on standard floating point implementations. Therefore, we wrote a bespoke solver in Haskell that exploits the fact that we have only few integral variables so that we can explore their combinatorial space completely to determine feasibility.

5.1 Experimental Setup

We solve the robust optimization problem for the analytical expressions we derived above with the algorithm depicted in Figure 5. This algorithm has as input the set of constraints, a parameter p and a parameter α. It will output at most p robustly feasible tuples (s,r,d,cost) from a list of all robustly feasible such tuples as follows: it will identify the maximal values of d for which such tuples are robustly feasible, and it will report exactly one such tuple for each value of d where r is minimal, and *cost* is minimal whilst also bounded above by $\alpha \cdot c_m$ where c_m is the globally minimal cost. This also determines the values of s in these tuples and so the algorithm terminates.

Now, having defined the required analytical expressions and the algorithm to report the best p robustly feasible tuples in Figure 5, we also want to validate these expressions and the algorithm experimentally. Our setup for this is based on pure Haskell code, as functional – and in particular – Haskell programs offer the advantages of being modular in the dimension of functionality, being strongly typed as well as supporting an easy deconstruction of data structures, particularly lists [4]. Furthermore, the arbitrary-precision verification is handled by the external `Data.BigFloat` package, which is also written in Haskell. Further verification and validation of the received results is pursued by unit testing using an arbitrary precision calculator. Moreover, our experiments ran on a machine with the following specifications: Intel(R) Xeon(R) CPU E5-4650 with 64 cores and 2.70GHz and 64 GB RAM on each core.

We instantiate the model in Figure 4 with the constants shown in Table 1. We choose T to be $1/(50 \cdot 10^9) = 0.02 \cdot 10^{-9}$ for a mining ASIC from early 2016 with an estimated cost of 2700 USD at that time, so a fixed cost of TFC = 3000 USD seems reasonable. Let us now explain the value $2 \cdot 10^{-12}$, which models the energy cost of a sole hash (we can ignore other

input : p, α, and values for all constants in Figure 4
invariant: *list* lists tuples $(s, r, d, cost)$ in descending order for d

begin
  define all constants for constraints in Figure 4;
  $list = [(s, r, d, cost) \mid cost = Cost(s, r, d), feasibleFloat(s, r, d)\ is\ true]$;
  $list = [(s, r, d, cost) \in list \mid feasibleFloat^{u_s}_{u_d}(s, r, d)\ is\ true]$;
  while $(\exists (s, r, d, cost) \neq (s', r', d', cost') \in list)$ **do**
    if $d' = d$ ***and*** $cost' < cost$ **then**
      remove $(s, r, d, cost)$ *from list*;
    else
      if $cost' = cost$ ***and*** $r' < r$ **then**
        remove $(s, r, d, cost)$ *from list*;
      else
        remove $(s', r', d', cost')$ *from list*;
      end if
    end if
  end while
  $c_m = \min\{c \mid \exists (s, r, d, cost) \in list\}$;
  while $(\exists (s, r, d, cost) \in list\colon cost > \alpha \cdot c_m)$ **do**
    remove $(s, r, d, cost)$ *from list*;
  end while
  $results = list\ of\ first\ p\ tuples\ from\ list$;
  $results = [(s, r, d, cost) \in results \mid feasibleBigFloat^{u_s}_{u_d}(s, r, d)\ is\ true]$;
  return *results;*
end

Figure 5: Algorithm, written in imperative style of list processing, for reporting the best p robustly feasible tuples $(d, r, s, cost)$ such that d is maximal subject to the cost $cost = Cost(s, r, d)$ satisfying $cost \leq \alpha \cdot c_m$ where c_m is the minimal cost for all robustly feasible tuples (s, r, d) and $\alpha \geq 1$ is a tolerance factor for increasing cost beyond c_m. Predicate $feasibleFloat(s, r, d)$ is true iff all constraints in Figure 3 are true for this choice of s, r, and d under normal precision floats. Predicates $feasibleBigFloat$ and $feasibleBigFloat^{u_s}_{u_d}$ are true iff their mathematical definition is true under arbitrary-precision floating points (applying `Data.BigFloat` version 2.13.2).

costs on that time scale). A conservative estimate for the power consumption of an ASIC is 10 watts per Gigahashes per second, i.e. 10 watts per Gh/s. We estimate the cost of one kilowatt hour kWh to be about 10 cents. A kWh is $3600s$ times kW and one kW is 1000 watts. So 10 watts per Gh/s equals $10 \cdot 3600$ watts, which amounts to $36kWh$. So the cost for this is $36 \cdot 10$ cents per hour, i.e. 360 cents per hour. But then this costs $360/3600 = 0.1$ cents per second. The price for a sole hash is therefore 0.1 divided by $50 \cdot 10^9$, which equals $\mathrm{TVC} = 2 \cdot 10^{-12}$.

We insist on having at least 4 miners and cap this at 80 miners. The *shared* nonce space for miners is assumed to be between 24 and 64 bits. The level of difficulty is constrained to

$s_l = 4$	$s_u = 80$	$r_l = 24$	$r_u = 64$
$d_l = 4$	$d_u = 64$	$\text{TVC} = 2 \cdot 10^{-12}$	$\text{TFC} = 3000$
$\alpha = 1.5$	$T = 0.002 \cdot 10^{-9}$	$th = 300$	$th' = 300$
$\delta = 10^{-9}$	$\delta_1 = 1$	$\delta_3 = 0.001$	$\delta_2 = 10^{-9}$
$\tau_l = 0$	$\mu = 300$	$\epsilon = 2^{-64}$	$k = 5$
$u_d = 3$	$u_s = 5$	$c = 6$	$l = 4$

Table 1: Constants for our experiments. This does not specify the values of τ_u which will vary in experiments. Some experiments will also vary the values of δ, δ_2 or δ_3

be between 4 and 64. We list optimal tuples that are within a factor of $\alpha = 1.5$ of the optimal cost. We make the value th' irrelevant by setting $\delta_1 = 1$ which makes the constraint for th' vacuously true. The probability for mining failure is not allowed to exceed $\epsilon = 2^{-64}$. Setting $\tau_l = 0$ means that we don't insist on the average mining time to be above any particular positive time. The probability that mining a block takes more than $th = 300$ seconds is bounded by 10^{-9}. And the probability that more than one miner finds PoW within $\mu = 300$ seconds is also bounded by 10^{-9}. The algorithm reports the top $k = 5$ optimal tuples – and reports fewer if there are no 5 feasible tuples. The remaining constants for robustness are as given in Figure 4.

Let us now specify some values of τ_u of interest. As reported in [6], Bitcoin is believed to handle up to 7 transactions per second, Paypal at least 100 transactions per second (which we take as an average here), and Visa anywhere between 2000 and 7000 transactions per second on average. By *transactions per second* we mean that blocks are mined within a period of time consistent with this. Of course, this depends on how many transactions are included in a block. For sake of concreteness and illustration, we take an average number of transactions in a Bitcoin block, as reported for the beginning of April 2016, that is 1454 transactions.

For a Bitcoin style rate, but in our *governed* setting, this means that a block is mined in about $1454/7 \sim 207.71$ seconds. Since $T \cdot E^s(noR)$ is the expected (average) time to mine a block, we can model that we have 7 transactions per second on average by setting $\tau_u^{Bitcoin}$ to be $1454/7$. Similarly, we may compute τ_u^{PayPal} and τ_u^{Visa} based on respective 100 and 7000 transactions per second:

$$\tau_u^{Bitcoin} = 1454/7 \qquad \tau_u^{PayPal} = 1454/100 \qquad \tau_u^{Visa} = 1454/7000 \tag{19}$$

5.2 Experimental Results

Transactions per second as in Bitcoin, PayPal, and Visa. We show in Table 2 the top 5 optimal robustly feasible tuples for the various values of τ_u in (19). We see that all three transaction rates per second can be realized with 18 miners and a 50-bit shared nonce space in our governed setting. The achievable level of difficulty (within the uncertainty in u_s and u_d) drops from 43 for the Bitcoin style rate to 41 for a PayPal style rate, and down to 35 for a Visa style rate.

Let us see what changes if we keep all constants as above but now

$$\delta = \delta_2' = 2^{-64} \tag{20}$$

$\tau_u^{Bitcoin}$ $(s, r, d, cost)$	τ_u^{PayPal} $(s, r, d, cost)$	τ_u^{Visa} $(s, r, d, cost)$
$(18, 50, 43, 54017.593)$	$(18, 50, 41, 54004.399)$	$(18, 50, 35, 54000.069)$
$(18, 50, 42, 54008.797)$	$(18, 50, 40, 54002.200)$	$(18, 50, 34, 54000.035)$
$(18, 50, 41, 54004.399)$	$(18, 50, 39, 54001.100)$	$(18, 50, 33, 54000.018)$
$(18, 50, 40, 54002.200)$	$(18, 50, 38, 54000.550)$	$(18, 50, 32, 54000.009)$
$(18, 50, 39, 54001.010)$	$(18, 50, 37, 54000.275)$	$(18, 50, 31, 54000.005)$

Table 2: Top 5 optimal tuples for our robust optimization problem with constants as in Table 1 and values τ_u as listed in (19). Costs are rounded up for three decimal places

This models that the probability that more than two miners find PoW within 300 seconds, as well as the probability of mining to take more than 300 seconds, are very small. We now only report the changes to the results shown in Table 2 for the top rated, optimal tuple. For $\tau_u^{Bitcoin}$, the level of difficulty drops from 43 to 39 but there are still 18 miners and a shared nonce space of 50 bits. This tuple $(s, r, d) = (18, 50, 39)$ is also optimal for τ_u^{PayPal} now, whereas the problem now becomes infeasible for τ_u^{Visa}. We may explore the *feasibility boundary* for τ_u given the values of all other constants as just discussed: the optimization problem is infeasible for $\tau_u = 0.2748$ but becomes feasible when τ_u equals 0.2749.

Next, let us keep the constants as in (20) but where we now change δ_3 to 0.0001, decreasing the probability that corrupt miners can win 6 consecutive mining races. The optimal robustly feasible tuple for $\tau_u^{Bitcoin}$ is $(s, r, d, cost) = (24, 51, 39, 72001.010)$ where cost is again rounded up for three decimal places. This increase of δ_3 therefore requires 8 more miners, 1 more bit of shared nonce space, but can still realize 39 as level of difficulty. This same tuple is also optimal for τ_u^{PayPal} whereas the problem remains infeasible for τ_u^{Visa}.

Larger transaction rates per second. In our next experiment we want to vary the average number of transactions in a block from 1454 to larger values. This is sensible for our use case as transactions only record a hash, which may be 8 bytes each. These results are seen in Table 3 for 50000 transactions on average in a block with the constants as in Table 1 again. Let us discuss the impact of changing the average number of transactions in a block from 1454 to 50000. This has no impact when 7 transactions per second are desired, the level of difficulty increases by 2 from 41 to 43 for 100 transactions per second, and the level of difficulty increased by 5 from 35 to 40 for 7000 transactions per second. This quantifies the security benefits of packing more transactions into a block for mining throughput.

Let us now see how these results change when we set δ and δ_2 to 2^{-64} as in (20). The optimal tuples still have 18 miners and a shared nonce space of 50 bit. But for Bitcoin style rate 7 as well as for PayPal style rate 100, the level of difficulty drops by 4 from 43 to 39 whereas it drops by 1 from 40 to 39 for a Visa style rate of 7000. Note that these are the same results as for $\tau_u^{Bitcoin}$ and τ_u^{PayPal} but for τ_u^{Visa} this problem was infeasible.

If we now change δ_3 to 0.0001 and keep the values for δ and δ_2 as in (20), then the optimal tuples are now the same ones as for $\tau_u^{Bitcoin}$ and τ_u^{PayPal} reported above (respectively). This problem was infeasible for τ_u^{Visa} but is feasible now for $\tau_u = 50000/7000$ with optimal tuple $(s, r, d, cost) = (24, 51, 39, 72001.010)$ as for the other two cases.

$Bitcoin \equiv 7$ $(s, r, d, cost)$	$PayPal \equiv 100$ $(s, r, d, cost)$	$Visa \equiv 7000$ $(s, r, d, cost)$
$(18, 50, 43, 54017.593)$	$(18, 50, 43, 54017.593)$	$(18, 50, 40, 54002.200)$
$(18, 50, 42, 54008.797)$	$(18, 50, 42, 54008.797)$	$(18, 50, 39, 54001.100)$
$(18, 50, 41, 54004.399)$	$(18, 50, 41, 54004.399)$	$(18, 50, 38, 54000.550)$
$(18, 50, 40, 54002.200)$	$(18, 50, 40, 54002.200)$	$(18, 50, 37, 54000.275)$
$(18, 50, 39, 54001.100)$	$(18, 50, 39, 54001.100)$	$(18, 50, 36, 54000.138)$

Table 3: Top 5 optimal tuples for our robust optimization problem with constants as in Table 1 and values τ_u given as 50000/7, 50000/100, and 50000/7000 (respectively). Costs are rounded up for three decimal places

Feasibility boundary for transaction rates per second. We now repeat the last experiment by varying the average number of transaction blocks from 50000 to half a million, in increments of 50000. For 100000 as average number of transactions, we redo the three experiments (original constants in as Table 1, larger $\delta = \delta_2$ as in (20), and larger $\delta_3 = 0.0001$ with $\delta = \delta_2$ as in (20)) and can note only one change: for $\tau_u = 100000/7000$ and the constants in Table 1 the level of difficulty increases by 1 from 40 to 41. All other values of $(s, r, d, cost)$ stay the same in these three experiments; there is no change whatsoever for $q/7$ and $q/100$ when q is between 10000 and 500000 in increments of 50000. When we increase the average number of transactions in a block to 150000, we similarly see only one change for 150000/7000 for the experiment with the constants in Table 1 such that the level of difficulty increases now by 1 from 41 to 42. Increasing the average number of transactions in a block to 200000 does not change these results, but changing this number to 250000 also results in only one change: for the experiment with the constants in Table 1 the level of difficulty increases now by 1 from 42 to 43. None of the three experiments then result in any more changes when the average number of transactions per block changes to 30000, 350000, 400000, 450000 or 500000.

Transactions per second for 500000 **transactions on average in blocks.** Now we explore how many transactions per second can be processed when we have $\tau_u = 500000/rate$ where $rate$ is the desired number of transactions processed per second. Table 4 shows the optimal robustly feasible tuple for the experiment with constants as in Table 1 as a function of $\tau_u = 500000/rate$, except that $\delta = \delta_2 = 2^{-64}$ as in (20) and $\delta_3 = 0.0001$. This shows that for this higher average number of hashes recorded in a block, the rate of such hashes processed on the blockchain per second can span 4 orders of magnitude without changing the optimal tuple of 24 miners, 51 bits of shared nonce space, and level of difficulty 39.

Range of feasible sizes for nonce space. Our tools can also compute and validate whether a robustly feasible tuple $(s, r, d, cost)$ has any other values r' for which $(s, r', d, cost)$ is robustly feasible. For example, for all the optimal tuples $(s, r, d, cost)$ we computed above, we conclude that we may change r to any r' satisfying $r < r' \leq 64$.

rate	*optimal tuple* $(s, r, d, cost)$
10	$(24, 51, 39, 72001.09951162778)$
100	$(24, 51, 39, 72001.09951162778)$
10000	$(24, 51, 39, 72001.09951162778)$
100000	$(24, 51, 39, 72001.09951162778)$
10^6	*infeasible*
10^7	*infeasible*
10^8	*infeasible*

Table 4: Optimal robustly feasible tuples with cost rounded up for three decimal points where $\tau_u = 500000/rate$. The experiment uses the constants from Table 1, expect that $\delta = \delta_2 = 2^{-64}$ as in (20) and $\delta_3 = 0.0001$

5.3 Statistical evaluation

We will evaluate our model by comparing its random variables, represented in analytical form, with empirical counterparts generated through experiments. We make this comparison in terms of both absolute and relative error.

We ran experiments consistent with Assumptions 1-3 to generate data for testing the random variables used in our model. Specifically, for a triple (s, r, d) we generated 10000 mining races with hash function *Double SHA-256* and recorded their outcome: either a failure of all s miners to proof work for level of difficulty d or an integer rds saying that mining took rds many rounds for proof of work with level of difficulty d. The triples we studied where (s, r, d) in

$$\{4i \mid 1 \leq i \leq 8\} \times \{2^i \mid 3 \leq i \leq 7\} \times \{4, 8, 12, 16, 17, 18, 19, 20\}$$

The data we thus generated amount to three million and two-hundred thousand mining races.

Suppose, for a given (s, r, d), that l of these 10000 outcomes where integers rds. Then the *empirical failure probability* for triple (s, r, d) equals $\frac{10000-l}{10000}$.

For the evaluation of $E^s(noR)$, we compute the arithmetic average $\overline{rds}$ of all rds values for a triple (s, r, d) and compare this with the exact value of $E^s(noR)$ by computing the *absolute* error in (21) and the *relative* error in (22)

$$|\, E^s(noR) - \overline{rds}\,| \tag{21}$$
$$|\, E^s(noR) - \overline{rds}\,|/E^s(noR) \tag{22}$$

We analyze the absolute error first: for each threshold value x from $\{0.01 + b \cdot 0.05 \mid 0 \leq b \leq 20\}$, let $\mathcal{R}_x$ be the set of all outcomes rds for all triples (s, r, d) whose empirical or analytical failure probability (i.e. $prob^s(\mathsf{failure})$ or $\frac{10000-l}{10000}$) is strictly less than x. Then for the set

$$S_x = \{|\, E^s(noR) - \overline{rds}\,| \mid rds \in R_x\} \tag{23}$$

function $x \mapsto \max(S_x)$, seen in Figure 6, renders a discrete graph of the maximal absolute error from set S_x; whereas function $x \mapsto \frac{1}{|S_x|} \cdot \sum_{r \in S_x} r$, named $\overline{S_x}$ in the same figure, denotes the arithmetic average of the elements in S_x. For the worst-case absolute errors, we see that

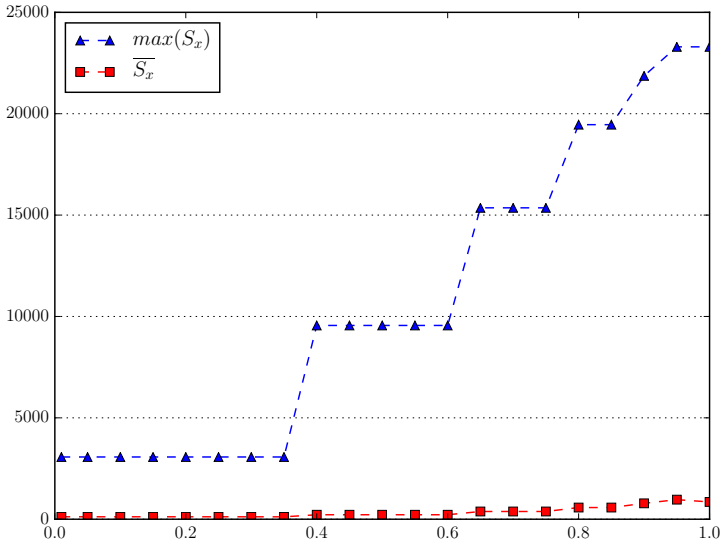

Figure 6: Discrete functions $x \mapsto \max(S_x)$ and $x \mapsto \overline{S_x}$ where $\overline{S_x}$ is arithmetic average of S_x in (23)

they remain well below 5000 rounds, namely at about 3070 rounds for failure probabilities < 0.35. In that range of failure probabilities, the average absolute error is about 119 rounds which is a very small difference. To illustrate, for the time $T = 0.02 \cdot 10^{-9}$ to compute a sole hash used above, this would mean that the average absolute error for failure probability < 0.35 amounts to a time difference of about $2.38 \cdot 10^{-9}$ seconds. We did the same analysis for the relative error in (22). For example, the relative maximal error is then about 0.0323 and the arithmetic average of relative errors is about 0.0069.

In summary, we could show that the random variable $E^s(noR)$ accounts really well for our experimental data, and does so for failure probabilities that even extend into significant ranges such as a 30% failure probability. But it also highlights that modelers need to be mindful of choosing values for ϵ in Figure 2 that won't invalidate the predictive value of random variables such as $E^s(noR)$. Clearly, for values of ϵ considered above, for example 2^{-64}, this is a non-issue.

For the evaluation of $prob^s(\mathsf{failure})$, the absolute error is

$$|\, prob^s(\mathsf{failure}) - \frac{10000 - l}{10000} \,| \tag{24}$$

where l is as above the number of successful mining races of 10000 overall such races for triple (s, r, d) that defines $prob^s(\mathsf{failure})$. The worst-case, maximal, value of all absolute errors in (24) ranging over all 320 combinations of (s, r, d) is about 0.0118. The average of the absolute errors in (24) for all these 320 combinations is about 0.00057. This shows that our model of failure probabilities is very precise, both in a worst-case and in a statistical average sense when compared to our experimental data.

For the evaluation of $prob^s(PowTime > th)$, let the experiment for triple (s, r, d) have been run on a machine which takes T time (in seconds) to compute a sole hash. We use $T \cdot 2^d$

as a rough approximation of how long it takes to mine a block. Then we set $th = 1.2 \cdot T \cdot 2^d$ as a reasonable test value of th, an increase of twenty percent. Let q be the number of times that a reported outcome rds for triple (s, r, d) – which defines $prob^s(PowTime > th)$ – satisfies $T \cdot rds > th$. Then we want to compute $prob^s(PowTime > th)$ exactly as above, and compute the absolute error

$$|prob^s(PowTime > th) - \frac{q}{l}| \tag{25}$$

as the difference between the formal and the empirical probability. However, we only consider data for tuples (s, r, d) for which the constraint $\lceil (th/T) - 1 \rceil < \lambda$ is met; otherwise, our computation of $prob^s(PowTime > th)$ would result in negative and therefore unsound absolute errors. This is a valid evaluation approach since triples (s, r, d) that violate $\lceil (th/T) - 1 \rceil < \lambda$ would never contribute to optimizations in our models. The worst-case, maximal, absolute error ranging over all triples (s, r, d) that satisfy $\lceil (th/T) - 1 \rceil < \lambda$ is about 0.003. The arithmetic average of all these absolute errors is about 0.0001 – which suggests a very good fit of our model with these experimental data.

For the evaluation of $prob^s(PowTime < th')$, we now set $th' = 0.8 \cdot T \cdot 2^d$ as a reasonable test value of th', a *decrease* of twenty percent. Let o be the number of times that values rds in our data for (s, r, d) – defining $prob^s(PowTime < th')$ – satisfies $T \cdot rds < th'$. We compute $prob^s(PowTime < th')$ as above, and report the worst-case absolute error

$$|prob^s(PowTime < th') - \frac{o}{l}| \tag{26}$$

over all triples (s, r, d) for which the empirical or analytical failure probability is strictly less than x. The results, depicted in Figure 7, show that for failure probabilities below 0.35 the *worst-case* absolute errors are about 0.00366 and so very small. The worst-case absolute errors for larger failure probabilities are about 0.041. The arithmetic average of all these absolute errors is more variable in x: it ranges from 0.000235 for $x = 0.01$ to 0.00176 (which is also the maximal arithmetic average over all x considered) for x equal to 1; and these are very small values. Results for the corresponding relative error are of the same quality and so omitted.

These findings above are evidence of the validity of our random variables and their use in our modelling approach. But they also highlight that failure probabilities of mining larger than 30% may require caution in using our approach. We emphasize that these experiments indirectly depended on security properties of the underlying hash function.

The results for the statistical evaluation of $prob^s(disputes\ within\ \mu)$ will be included in a future version of this paper.

6 Conclusions

In this paper we considered blockchains as a well known mechanism for the creation of trustworthiness in transactions, as pioneered in the Bitcoin system [12]. We studied how blockchains, and the choice and operation of cryptographic puzzles that drive the creation of new blocks, could be controlled and owned by one or more organizations. Our proposal for such governed and more central control is that puzzle solvers are mere resources procured by those who control or own the blockchain, and that the solution of puzzles does not provide

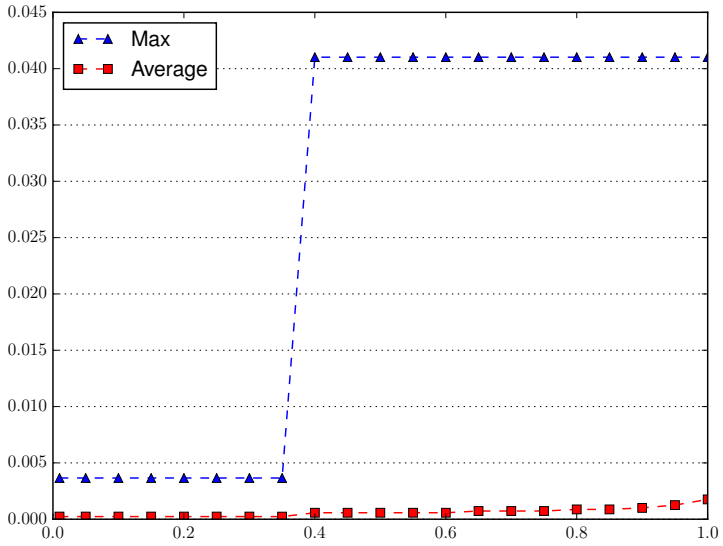

Figure 7: Graph *Max* shows worst-case absolute error in (26) for all triples (s, r, d) with failure probability $< x$. Graph *Average* shows the same result for arithmetic average of these absolute errors

any monetary or other reward. In particular, a newly solved block will not create units of some cryptocurrency and there is therefore no inherent incentive in solving puzzles.

The absence of incentives thus avoids the well known problems with game-theoretic behavior, for example that seen within and across mining pools in Proof of Work systems such as Bitcoin. Furthermore, it lends itself well to the development of proprietary or private blockchains that are *domain-specific* and whose specific purpose may determine who can access it and in what ways. We illustrated this idea with a use case in which financial transactions recorded within conventional accounts would be recorded as hashes within a governed blockchain and where it would be impractical to use hash chains, due to non-linearizability of transaction flows.

A blockchain design should of course have specifications of its desired behavior, including but not limited to the expected time for creating a new block, resiliency against corruption of some of the puzzle solvers, service level guarantees such as a negligible probability that the time needed for creating a new block exceeds a critical threshold or a negligible probability that more than one solver does solve a puzzle within a specified period of time.

We developed mathematical foundations for specifying and validating a crucial part of a governed blockchain system, the solving of cryptographic puzzles – where we focussed on Proof of Work. In our approach, owners of a blockchain system can specify allowed ranges for the size of the shared nonce space, the desired level of difficulty, and the number of miners used; and they can add mathematical constraints that specify requirements on availability, security, resiliency, and cost containment. This gives rise to MINLP optimization problems that we were able to express in analytical form, by appeal to the ROM model of cryptographic hash functions used for cryptographic puzzles.

We then wrote an algorithm that can solve such MINLP problems for sizes of practical

relevance. We illustrated this on some instances of that MINLP problem. This demonstrated that we have the capability of computing optimal design decisions for a governed Proof of Work system, where resiliency is modeled through robust optimization. This *mining calculus* also supports change management. For example, if we wanted to increase mining capacity and/or mining resiliency, our mathematical model could be used to determine how many new miners are needed to realize this – be it for the same or better hardware specifications. For another example, our tool could be used to determine optimal numbers of used miners or parameters by reacting to new energy prices.

Our approach and mathematical model are consistent with the consideration of several organizations controlling and procuring heterogeneous system resources, with each such organization having its bespoke blockchain, and with the provision of puzzle solving as an outsourced service. We leave the refinement of our mathematical models to such settings as future work. It will also be of interest to develop mathematical techniques for the real-time analysis of such blockchains, for example, to assess statistically whether the observed history of cryptographic puzzle solutions is consistent with the design specifications.

We hope that the work reported in this paper will provoke more thinking about the design, implementation, and validation of blockchains that are centrally – or in a federated manner – owned and controlled and that may fulfill domain-specific needs for the creation of trustworthiness. We believe that many domains have such needs that the approach advocated in this paper might well be able to meet: existing financial processes and payment workflows (which conventional cryptocurrencies are more likely to replace than to adequately support), trustworthiness of information in Internet of Things systems (where the difficulty of the puzzle, for example, may have to be contained), but also systems that have governed blockchains at the heart of their initial design (for example, a payment system in which the temporal and causal history of payments, logs, and audits is recorded on the blockchain).

Acknowledgements: This work was supported by the UK Engineering and Physical Sciences Research Council with a Doctoral Training Fees Award for the first author and with projects [grant numbers EP/N020030/1 and EP/N023242/1]. We expressly thank Ruth Misener for having run some of our models on state-of-the-art global MINLP solvers – the tools ANTIGONE [11], BARON [13], and SCIP [15] – to conclude that these solvers judge the models to be infeasible, although they are feasible.[1]

References

[1] Ali, R., Barrdear, J., Clews, R., and Southgate, J. Innovations in payment technologies and the emergence of digital currencies. *Quarterly Bulletin* (2014). Published by the Bank of England.

[2] Arvind Narayanan, Joseph Bonneau, E. F. A. M. S. G. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction.* Princeton University Press, 2016.

[1] In fairness, these global MINLP solvers were not built for dealing with numerical problems such as those encountered in our models.

[3] Ben-Tal, A., Ghaoui, L. E., and Nemirovski, A. *Robust Optimization*. Princeton University Press, 2009.

[4] Bird, R. *Thinking Functionally with Haskell*. Cambridge University Press, 2015.

[5] Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A., and Felten, E. W. SoK: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015* (2015), pp. 104–121.

[6] Danezis, G., and Meiklejohn, S. Centrally banked cryptocurrencies. *CoRR abs/1505.06895* (2015).

[7] del Castillo, M. The DOA Hacker is Getting Away. Online article on `coindesk.com`, 8 August 2016.

[8] Horne, D. Hash chain. In *Encyclopedia of Cryptography and Security, 2nd Ed.* Springer, 2011, pp. 542–543.

[9] Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., and Wolsey, L. A., Eds. *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. Springer, 2010.

[10] Milne, R. Sweden's Riksbank eyes digital currency. Online article of the Financial Times, 15 November 2016.

[11] Misener, R., and Floudas, C. A. ANTIGONE: Algorithms for coNTinuous Integer Global Optimization of Nonlinear Equations. *J. Glob. Optim. 59*, 2-3 (2014), 503–526.

[12] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System, May 2008. Published under pseudonym.

[13] Tawarmalani, M., and Sahinidis, N. V. A polyhedral branch-and-cut approach to global optimization. *Math. Program. 103* (2005), 225–249.

[14] Vigerske, S. MINLP Library 2. Online benchmark repository at `http://www.gamsworld.org/minlp/minlplib2/html/`.

[15] Vigerske, S. *Decomposition in Multistage Stochastic Programming and a Constraint Integer Programming Approach to Mixed-Integer Nonlinear Programming*. PhD in Mathematics, Humboldt-University Berlin, 2012.

[16] Wattenhofer, R. *The Science of the Blockchain*. Inverted Forest Publishing, 2016.