Proof of useful work

Ben Stokmans
5V2, natuur en gezondheid

Camiel Schilham
6V2, natuur en techniek

24th April 2022

Research paper in the field of computer science

Under supervision of Garmt de Vries-Uiterweerd, Ph.D.

Abstract

Due to the rapid growth of blockchain technology in recent years, it has become apparent that traditional proof of work is not environmentally sustainable. To retain advantages such as decentralization and security, a modified hash function can be used to make the majority of the work done through mining useful. The pseudo-random output of a hash function can be utilized to deterministically select a section of a data set to perform useful operations on. The solution can then be appended to the initial output of the hash to be hashed again. This effectively integrates useful work into a hash function.

The integration of prime factorization of integers into a hash function is possible. Some issues arise, such as the nonuniform complexity of this modified hash function. This causes a so-called "lazy mining problem" meaning miners will selectively solve hashes of which the complexity is known to be low. This maximizes their hash rate to the detriment of the usefulness of work done. Further research is necessary to address this issue.

Contents

List of Figures

1 Preface

1.1 Introduction

A blockchain is a digital, decentralized data storage medium consisting of "blocks"—units of data linked together to form a chronological chain. Copies of a blockchain are stored in a decentralized fashion by nodes[1] using an internet connection. The most common blockchain protocol used is proof of work, in which new blocks are created through a process called "mining". This process can be summarized as a complex mathematical function that can be used to arbitrarily determine who gets to create the next block, weighted by the amount of "work" they do (i.e., if one does twice as much work, one is twice as likely to create the next block). This function will be referred to as the "work function" going forward. Being chosen to create the next block is desirable because of the commonly associated "block reward", a significant monetary reward to whomever appends the next block.

Usually, as a proof of work blockchain scales, so does the total amount of work done by miners. This can cause the electricity consumption induced by mining to increase to such an extent that it becomes a global climate issue. For example, it is estimated that one Bitcoin transaction uses 1779 kWh of electrical energy [2], approximately as much as is needed to drive an electric car from Amsterdam to Vladivostok—11 000 km. Besides generating a potential monetary reward for miners and adding a block to the chain, mining on a traditional proof of work blockchain is effectively useless; all the approximately 1779 kWh of electrical energy used per Bitcoin transaction goes to waste.

If the work function on these blockchains can be made useful, this downside would be fully or partially resolved. The enormous amounts of computing power behind a large blockchain could, for example, be used for scientific purposes. The viability of such a blockchain design will be the primary focus of this paper.

1.2 Structure

To answer the research question of this paper regarding the viability of a proof of useful work blockchain, a logical structure shall be followed. First, in chapter 2, the fundamentals of contemporary proof of work blockchain technology will be explained. Following that in chapter 3, an overview of potential methods to realize proof of useful work. Their respective drawbacks will be addressed and their feasibility assessed. Lastly, in chapter 4,

[1] Nodes are devices that store and allow interaction with a copy of a blockchain-based on a pre-programmed set of rules. This set of rules defines the protocol that the blockchain uses. In practice, not all nodes keep a copy of the entire blockchain, however, the utility remains mostly the same.

a practical implementation is presented in detail, and its viability is investigated.

2 Contemporary methods and technology

2.1 Proof of work

2.1.1 Hash functions

In order to improve upon the work function of a proof of work blockchain, the current work function must be understood first: a hash function. One could argue that this is the single most important function in a proof of work blockchain.

Quite simply put, a hash function is a function that takes data of an arbitrary length as input and outputs a fixed-length value, as depicted in figure 1. Hash functions also have several other characteristics, most importantly: uniform distribution of outputs and deterministic output for any given input.

Because the input of a hash function can be of arbitrary length, there are an infinite number of inputs that will produce a given output. Take for example SHA-256, a hash function that will output a 256-bit value. This means there are $2^{256} \approx 1.158 \cdot 10^{77}$ possible outputs (so, a finite number) and an infinite number of inputs. From this one can deduce that there are an infinite number of possible inputs that produce the same output and that, subsequently, it is exceedingly difficult to recover an output's respective input.

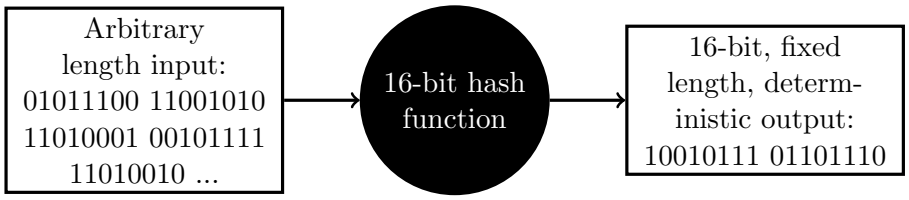

Figure 1: A basic illustration of the working of a 16-bit hash function.

2.1.2 Difficulty

The characteristics of hash functions can be utilized to prove that one has done a certain amount of work. In this instance, that is to say: prove one has done a certain number of computations.

This works as follows. First, a criterion for the output of a hash function has to be set, this is otherwise known as "difficulty". Because the value produced by a 256-bit hash function is merely a number ranging from 0 to $2^{256} - 1 \approx 1.158 \cdot 10^{77}$, doing so is quite simple given the distribution of outputs is roughly uniform. If, for example, approximately 1 out of every 100 hashes should be valid, a target value for the output T can be set

accordingly. Values below T are considered valid, and values equal to or above it invalid. The target value can be defined as:

$$T = \frac{H_{max} + 1}{D} \tag{1}$$

Where H_{max} is the maximum output value of the hash function used and D is the difficulty. This definition is valid so long as $H_{min} = 0$. The definition for the difficulty follows easily from this, namely:

$$D = \frac{H_{max} + 1}{T} \tag{2}$$

Given a difficulty of 100 this would result in:

$$T = \frac{2^{256} - 1 + 1}{100} \approx 1.158 \cdot 10^{75} \tag{3}$$

We can confirm that the chance of the produced output being valid for any given input is what we expect it to be (1 in 100 or 1%):

$$\frac{1.158 \cdot 10^{75}}{2^{256}} = 0.01 \tag{4}$$

Depicted in figure 2 is what the target value for the output of a hash function looks like with a difficulty of 10.

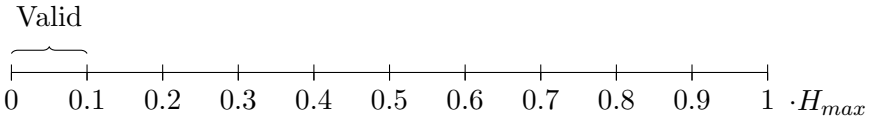

Figure 2: An illustration of blockchain proof of work difficulty

Typically, the difficulty is set automatically to keep the rate at which blocks are found consistent. The following formula can be used to calculate the difficulty, where S is the rate at which new blocks should be found in seconds per block, and R is the cumulative hash rate[2] in hashes per second:

$$D = RS \tag{5}$$

When this target value criterion is applied to a proof of work blockchain, the input of the hash function will be the contents of a block. Using the output of this hash function, we can determine whether a block is valid.

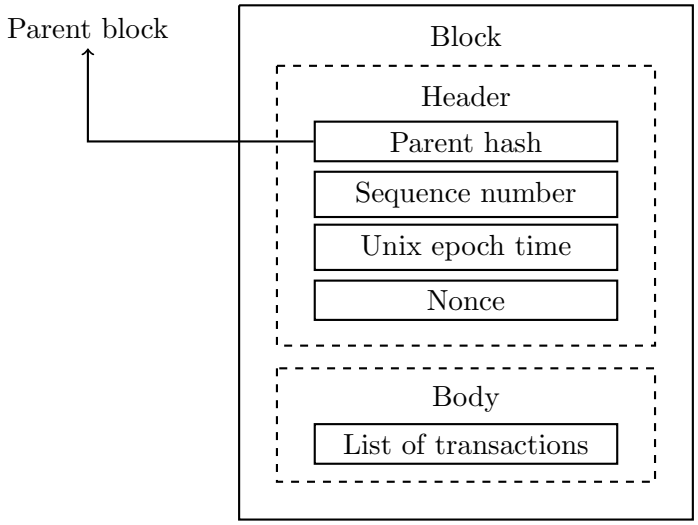

Figure 3: Structure of a simple block

2.1.3 Mining

Keeping in mind the characteristics of a hash function, this raises an issue: what to do if one wants to add some data to the blockchain, but the hash of the block is considered invalid, making the block invalid? Since a hash function is deterministic, we cannot get a different output for the same input. But this does not doom the contents of the block to never be added to the chain. To solve this, a nonce (number used once) and time field are added to a block. The nonce does not contain any relevant information, its sole purpose is to slightly alter the input of the hash function to attempt to create a block with a valid hash.

The time field is the Unix epoch time[3] in seconds. Strictly speaking, the time field is not necessary for proper operation of the blockchain, although it does add extra redundancy for finding new hashes in blocks with the same data. With the time field, the maximum cumulative hashing power for each block is 2^{64} hashes per second—that is, at least until the year 2038 problem[4] occurs, or essentially indefinitely if a 64-bit Unix epoch time is used. In case the time field is omitted, the total possible amount of hashes per block is 2^{64}. In theory, if the difficulty is absurdly high, this could result in a non-negligible chance that a block will not have any valid hashes. In practice,

[2]The sum of the hash rate of all nodes

[3]Seconds since January 1, 1970 12:00:00 AM GMT

[4]The year 2038 problem is a computational problem that will occur on the 19th of January 2038. On this day, the Unix epoch time will exceed the 32-bit unsigned integer limit. Time can then no longer be kept by machines who store Unix epoch time in a 32-bit unsigned integer.

this scenario is so improbable that it could be ignored. Still, having a time field adds redundancy and information, which is why it should be considered good practice.

The nonce and time fields of a block are continuously altered as described until the hash produced by a block is valid—in essence, a brute-forcing process. This process is called mining. Easily deduced from this is the fact that a miner with a higher hash rate will find more blocks, given the difficulty remains the same. What has been discussed so far explains the behaviour that mining determines arbitrarily who can append the next block to the chain, weighted by their hash rate as a fraction of the total hash rate of the blockchain.

The proof of work process described thus far using hash functions seems to be the perfect fit for its purpose. With the exception of a few improbable scenarios that will be discussed later, it is secure and a reliable backbone. There is no current alternative to a hash-like function that provides the same degree of security and decentralization for proof of work blockchains.

2.1.4 Wallets, transactions and consensus

A mechanism for creating blocks and adding them to the chain has now been established, as well as one for selecting who may add the block to the chain. Now, we must add purpose to what is being stored on the blockchain. The scope of this paper will be limited to blockchains storing transactions, consequently allowing parties to store currency as one would in a bank account. Contrary, however, to a banking system, the blockchain is decentralized and there is no authority to guarantee the validity of transactions. Instead, all nodes agree on a set of basic "rules", previously referred to as the blockchain protocol. These rules are enforced by miners and nodes. Some examples of these rules would be: one cannot transact more currency than the sum of previous transactions, one cannot create transactions if one is not the owner of the sending wallet, etcetera. So long as all the miners agree that these rules should be enforced, trust is ruled out completely—one does not need to trust any other party interacting with the blockchain.

The deterministic property of hash functions effectively allows anyone to verify that blocks on the blockchain are valid and have not been tampered with, provided that that party trusts that the hash function has not been compromised (i.e., that there is no computationally undemanding way of producing an input given an output besides brute-forcing).

To verify that a transaction was indeed made by the owner of the wallet, a signing mechanism is used. In fact, a wallet is no more than an asymmetric key pair, where the address is simply a string-encoded public key. When a transaction is made, the wallet's owner includes a signature. This signature is the hash of the transaction, encrypted with the wallet's private key. Anyone can then verify this signature by decrypting it with the public

key and verifying that the result is indeed the hash of the transaction.

2.2 Higher abstraction proof of work concepts

2.2.1 Protocol disagreements

When a disagreement about the blockchain protocol occurs (i.e., the set of rules that facilitate consensus) a fork can be created. This can be further divided into so-called "soft" and "hard" forks. Hard forks occur in the event that a branch of the network wants a protocol change, whilst the rest does not. This effectively splits the network into two, one with the modification applied and one without—the original network.

Soft forks, contrary to this section's title, do not really stem from protocol disagreements. Quite the opposite, in fact. A soft fork is simply a protocol update which is gradually rolled out across the entire blockchain network. This means an agreement must have been reached to update the protocol. An example of such a fork is EIP-1559 on the Ethereum network[5].

2.2.2 Mining pools

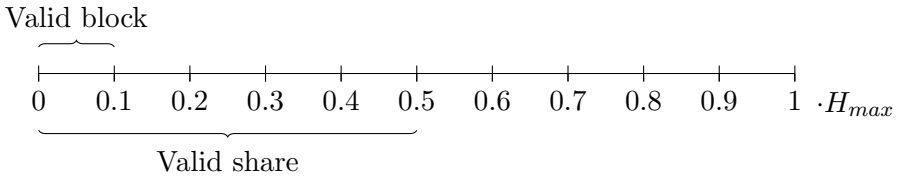

Figure 4: Valid shares (PPoWs) and valid blocks (FPoWs). Note that these difficulties are not proportional to what one would see in a practical implementation of a proof of work blockchain. Both the ranges for valid blocks and valid shares will be much smaller, and the difference between the extent of the ranges will be much larger.

Unless the blockchain network in question is minimal, the chance that an individual miner will find a block is negligibly slim. Therefore, miners often work together in so-called "pools". Large pools may control a significant share of a network's cumulative hashing power, in the order of tenths. A clever mechanism is used to allow many individual miners to cooperate and contribute to a pool.

All miners that want to contribute to a pool must attempt to make

[5]EIP-1559 also known as Ethereum Improvement Proposal 1559 is a proposal to make Ethereum transactions more efficient by using base fees and tips dependent on network usage.

blocks where the miner address field[6] is set to that of the pool's wallet. This means all block rewards and transaction fees are added to the pool's wallet. In return, a miner will get a consistent reward approximately proportional to their hash rate.

One cannot pay a miner based on how many blocks he finds. Some miners who contribute only a small amount of hashing power may never find a block. Doing this defeats the purpose of a pool completely. Instead, "partial proofs of work" or PPoWs are used, also commonly referred to as "shares". This differs from blocks which are deemed valid by the network, that will be referred to as "full proofs of work" or FPoWs. PPoWs are blocks that a miner finds, but with a much lower difficulty criterion. Thus, many of these PPoWs will not be accepted by the network; all FPoWs are PPoWs, but not nearly all PPoWs are FPoWs. Essentially, the pool tells a miner to report all blocks that he finds with difficulty D_{PPoW} or higher, where $D_{min,PPoW} \ll D_{min,FPoW}$.

A miner will be paid a fraction of the total rewards accumulated by the pool based on the proportion of shares he submits. As these shares are much easier to find than blocks, even miners with very little hashing power will find them regularly. They make reliable proof that the miner is contributing to the pool.

2.3 Other protocols

Combining everything in the previous paragraphs provides some fundamental insight into the proof of work protocol, that is: a blockchain protocol where the appender of the next block is chosen pseudo-randomly through a hash function. However, as has been made evident quite recently with blockchains like Cardano, this is not the only protocol. Some of these will be discussed in this subsection, along with their respective advantages and drawbacks.

2.3.1 Proof of stake

Proof of stake uses a set of rules similar to proof of work but instead of work, it uses stake. This difference is what makes it much more environmentally sustainable. A proof of stake network consists of nodes, these are often staking pools (proof of stake variant of mining pools). These pools provide users with a stable and reliable connection to the network. The protocol processes blocks by dividing the chain into epochs[7] and epochs into time slots. Every new slot a slot leader is elected from all nodes on the network,

[6] A field in the block header. The block reward and transaction fees are added to the specified address.

[7] An epoch is a period of time. This epoch of time is used to determine when specific events will take place inside a blockchain network, such as when rewards will be distributed or when a new group of validators will be assigned to check transactions.

this leader is then responsible for adding a new block to the chain. A single user will most likely join a staking pool with their stake in a cryptocurrency that resides on the blockchain. Participants in these pools receive rewards in the form of said cryptocurrency. To sum up, instead of many brute-forced mathematical equations, this protocol uses the fact that users have a stake in the cryptocurrency to form a consensus mechanism. This, though very cost and energy-effective, has some downsides. One example is that it is more vulnerable to majority attacks, which are discussed in 2.5.2.

2.3.2 Proof of space-time

In the case of proof of space-time, there is also a different type of work called seeding. During the seeding process, a miner (referred to as farmer in the context of proof of space-time) allocates unused disk space into plots. These plots store a collection of pseudo-random hashes. When a new block needs to be added to the chain, the network broadcasts a so-called challenge to all farmers, the farmers then scan their plots for the closest matching hash to the challenge. The farmer with the closest hash is then elected to add the next block to the chain. The probability of winning a block is the percentage of the total disk space that a farmer has seeded compared to the entire network. The network implements a verifiable delay function, which acts as a fast and efficient way to verify farmers' data and make sure all computation is done sequentially. The latter of which ensures that there is no advantage to be gained from parallel machines. Because of this, the network power consumption is relatively low.

2.4 Examples of practical applications of blockchain technology

2.4.1 Cryptocurrency

Cryptocurrency is one of the most well-known applications of blockchain technology. Bitcoin, Ethereum, Solana, and Cardano are just a few examples. Blockchain in this case is very useful because it allows for complete transparency and decentralization of the currency system. It allows many parties to form a consensus, regardless of whether they trust each other.

2.4.2 Non-fungible tokens

NFTs or non-fungible tokens are tokens that prove ownership of something, this can be a digital piece of data, an art piece, or a real estate deed. NFTs are digitally unique, there are no two NFTs that are the same, and the legitimacy of said NFT can be verified by the blockchain it is from. NFTs allow creators of content, art, or other digital content to verify ownership of their content and hereby sell online content without it being easily plagiarized.

Normally images of for example digital art can be copied and there will be no official art piece, whereas physical art pieces like Rembrandt's Night's Watch are unique and are not duplicated easily. But using NFTs, one can verify the ownership of even digital art pieces.

2.4.3 Smart contracts

Every user with a wallet on the blockchain[8] can create a smart contract, which is a self-executing contract with predetermined conditions. By providing code to be run on the blockchain and often a set fee paid in cryptocurrency, any user can create a smart contract. With a set of predetermined conditions, the network will know when to run said code, this code can do anything from releasing funds to transferring the ownership of an NFT. After the transaction is completed, the network will update the smart contract, so it cannot be changed, and only selected parties will be able to see the results.

2.4.4 Decentralized applications

Decentralized applications or "dapps" are applications that are deployed onto a decentralized network, such as a blockchain. Dapps are comprised of a front-end user interface and a smart contract. The advantage of such an application is that they are decentralized, so there is virtually no downtime, and it has great resistance to censorship. Some downsides include poor performance and maintenance issues because every task that needs to be performed by the network needs a miner to process said task and propagate it through the network. The time to process such a task is related to the blockchains' difficulty, as to add data to the blockchain the application has to wait for a new block to be added to the chain.

2.4.5 Blockchain as a Service

The blockchain-as-a-service model or BaaS for short is based on software-as-a-service (SaaS). SaaS works by providing a software product to a client, often on a subscription basis. BaaS works similarly by having a provider, in this case often a company that provides cloud computing solutions including BaaS which provides the necessary blockchain infrastructure for the client to build their blockchain software application with. This client company can then focus its efforts on creating its application while the provider takes care of the blockchain infrastructure and maintenance.

[8]The blockchain in question has to support smart contracts

2.5 The primary limitations of current blockchain technology

2.5.1 Environmental concerns

Due to proof of work's relatively rudimentary nature compared to other blockchain protocols, it consumes a large amount of electrical energy which, in turn, has a considerable impact on the environment. Consider, for example, the Ethereum blockchain. As of the September 20, 2021 the hash rate is $640.71\,\mathrm{TH\,s^{-1}}$ (terahashes per second) [8] and the network difficulty stands at 8.82P [7]. The Ethereum network uses a custom hash function called Ethash which is designed to make it harder to create an ASIC or application-specific integrated circuit for. This is done to lower the barrier of entry into Ethereum mining, which can be done on any graphics card[9] rather than requiring specific hardware designed for said hash function. One of the most popular graphics cards used for mining for Ethereum is the GeForce GTX 1070 [13] which has a hash rate of $30.9\,\mathrm{MH\,s^{-1}}$ at a power draw of 103.6 W [11] so it would take

$$\frac{8.82 \cdot 10^{15}}{3.09 \cdot 10^{7}} = 2.85 \cdot 10^{7}\,\mathrm{s} = 7.93 \cdot 10^{4}\,\mathrm{h} \tag{6}$$

for a GeForce GTX 1070 to find a new block, which is approximately 9 years. Obviously, the entire Ethereum network does not consist of a single graphics card. The time to find a block taking into account the cumulative hashing power of the Ethereum network is:

$$\frac{8.82 \cdot 10^{15}}{6.4071 \cdot 10^{14}} = 13.8\,\mathrm{s} \tag{7}$$

With this a large amount of hashing power comes an enormous amount of electricity consumption. The annual electricity usage of Ethereum sits at, 72.58 TW h which is equivalent to the annual energy consumption of Columbia[6]. This, of course, is a significant amount of electricity, but with almost 13 million active addresses and 1.2 million daily transactions [5] it is to be expected that proof of work network consumes a substantial amount of electricity.

With this come concerns that proof of work is not sustainable for future expansion and this has made room for other protocols to arise like proof of stake and proof of space-time.

The consensus mechanism used by proof of stake does not require resource-intensive mathematical equations. The power usage of for example Cardano, one of the most prevalent proof of stake blockchains, is estimated to be around 6 GW h [15]. To put that in perspective, the average household in the United States uses 10 715 kW h [9].

[9] Ethereum mining can only be done by graphics cards with 4 or more gigabytes of VRAM

So where the Cardano network could power approximately 560 US homes, the Ethereum network could power the whole of Columbia. As for proof of space-time, it still uses a lot of hardware in the form of hard drives. Notwithstanding, these consume significantly less power than graphics cards. The annual power consumption of the Chia blockchain[10] sits at, 0.306 TW h which is approximately

$$\frac{0.306}{72.58} = 0.00420 = 0.420\% \tag{8}$$

of the annual Ethereum network power consumption, which is considerably lower.

2.5.2 51%-attacks

Though little vulnerabilities of the proof of work protocol are known, it has been speculated that some scenarios could be detrimental for the integrity of one. Most significantly: a 51%-attack. One could state that this is the most consequential known attack to a proof of work network. Results of this attack include but are not limited to double-spending and mutating the blockchain's history is various other ways. The attack regards a scenario in which one party, most likely to be a mining pool, controls more than half of a network's hashing power.

The attack works as follows: suppose you control 60% of the hashing power of a proof of work blockchain. Instead of contributing that hashing power to the network, you could instead start work on a modified version of the main chain. This version is local and not published to the network for now. In this chain, you could, for instance, completely erase a transaction from history. Once you are happy with your modifications and generously refunded yourself that large transaction for, say, a brand-new Tesla you bought with cryptocurrency (i.e., you removed the transaction from the history of the blockchain), you can then publish your version of the chain.

The network will accept this version of the blockchain over the original one as an indirect consequence of having a larger amount of hashing power. As discussed previously, the difficulty can be defined as $D = RS$. S must remain constant for most blockchains, so the difficulty is adjusted automatically when the hash rate, R, changes. Your chain unmistakably has a higher difficulty than the main chain, because your hash rate is higher. When two versions of the same blockchain exist, the one with the highest difficulty will generally be accepted as the "real" one, meaning yours.

2.5.3 Mining pool manipulation attacks

Mining pool manipulation is the situation in which an attack pool infiltrates a target pool by sending infiltrators that then perform a block withholding

[10] As of January 10, 2022, the Chia Network is the largest proof of space-time blockchain

attack. A block withholding attack means that a malicious miner submits only PPoWs and discards all FPoWs. This, when executed on a large scale over a prolonged period of time, will cripple the target pool and will cause a decrease in the pay-outs of honest miners. These attacks are not common, as the attackers do not gain anything. There are few imaginable scenarios in which such an attack would make sense other than a mining pool rivalry.

2.6 Towards proof of useful work

To create a blockchain protocol where the work function achieves something useful, a suitable problem must be adapted to replace a hash function. It must retain some of the fundamental characteristics of a hash function, otherwise the security would suffer as a consequence. The next chapter will discuss possibilities for suitable problems.

3 Proof of useful work

The work problem in proof of work is an example of an intractable problem, which means there is no algorithmic way to solve it efficiently. There are very few problems that are similar enough to it to be used in exactly the same way. There are contenders such as protein folding and plasma insulin or glucose dynamics that are very similar in nature. If adapted properly, these problems could be integrated into a hash function, effectively making it useful.

3.1 Authority problem

An important factor is that one must be able to generate arbitrary data instead of receiving it from an external source. If the latter were the case, trust is no longer factored-out and everyone on the network must trust the authority that provides the data to be processed as useful work. In the event that the authority is malicious, it could provide data that it itself has already processed, making their own mining process much less resource intensive and increasing their chance of finding blocks disproportionately relative to their hashing power.

3.2 Hash function integration

One might be able to integrate a useful work problem into a hash function. This means the characteristics of a hash function are kept, while some useful work problem is solved as well. This might look as follows. First, run the hash function over your arbitrary data as usual and use the pseudo-random output to decide which part of the useful work problem to solve. Then, append the solution to the hash and hash it again. This creates a hash which is deterministic, given the solution to the useful work is deterministic and obligates the solution of said useful work.

3.2.1 Lazy mining problem

This brings us to a related problem: all executions of the work function must be approximately as difficult as any other execution. If this is not the case, miners will always want as many executions per second for as little work as possible, hence the name “lazy mining problem”. Miners could actively avoid computing hashes if they deem the complexity too high. They would then move on to a hash of which the complexity is known to be low, thereby maximizing their hash rate. This kind of selective solving could be fatal for a useful work implementation, as it might be those specific parts of the data set that are most important to solve.

3.3 Protein folding

In the field of biology, there are a great number of processes exist that can only be predicted by simulations, which are repetitive and resource-intensive. One example is protein folding. Protein folding is the problem of knowing into what three-dimensional form a one-dimensional sequence of amino acids (polypeptide) will fold. Achieving the correct three-dimensional structure of a protein is essential to knowing its function. Misfolded proteins are a result of a protein failing to fold into its native structure. These misfolded proteins are the cause of many allergies and other diseases. This is why the simulation of protein folding is so important for disease research. For example cancer, but also the very relevant SARS-CoV-2 virus. To understand how to counter the viral workings of the proteins that make up the COVID-19 virus, one must first know the functions of said viral proteins and for this, we need protein folding simulations. This folding has been done by distributed computing projects before, like Folding@Home [4]. Because of this, it might be an acceptable replacement for a work function.

It was found, however, that this problem has been made mostly redundant with the publication of extensive artificial intelligence predictive models like DeepMind's AlphaFold project [16] that can predict the three-dimensional structures of proteins using deep learning, posing a threat to the integrity of a blockchain using protein folding as work.

3.4 Astronomy

Comparable to biology, astronomy also has some examples of tasks that require large amounts of processing power. The SETI (Search for Extra-terrestrial Intelligence) project by the University of California, Berkeley [1] is an example of this. Radio SETI employs radio telescopes to listen for narrow-bandwidth radio transmissions from space. Such signals do not occur naturally, thus detecting them would be proof of extraterrestrial technology. Noise (disturbance from astronomical sources and telescope electronics) and signals from man-made objects such as radar and satellites make up the majority of radio telescope signals. All data from these telescopes is digitally analysed by SETI projects which is where the need for a large amount of processing power comes in, the amount of data produced by these telescopes is too large for one computer or even supercomputer to handle.

Unfortunately, this useful work implementation would suffer from the authority problem[11]. The data set must be provided by a trusted authority, compromising decentralization and reintroducing trust into the blockchain. It is therefore deemed unfit as useful work in a blockchain context.

[11] see 3.1

3.5 Integer factorization

Integer factorization is the process of breaking up a number into its prime components, for example, $15 = 3 \cdot 5$ where the factors are 3 and 5. The general number field sieve is the most efficient way to factorize integers larger than 10^{100}, citing a heuristic complexity of $[\log_2 n] + 1$ [12] where n is the integer to be factorized.

This problem does not seem to suffer from the authority problem. The data can be generated by anyone. This makes it a promising contender for the integration of useful work into a hash function. As a superior contender was not found, integer factorization will be the solution of choice.

4 Blockchain implementation

4.1 Design goals

The goal of this implementation is to ascertain whether or not it is possible to create a working and secure proof of useful work blockchain. The security, decentralization and viability are paramount aspects. It will not be necessary to create a production-ready version of the blockchain, as its sole purpose it to investigate the possibility of compromiseless proof of useful work[12]. This saves a lot of time and simplifies the implementation greatly, saving time and making analysis more forgiving. The code for our blockchain can be found at github.com/cfschilham/kophos.

4.2 Tools

Version 1.17 of the Go programming language will be used [17] for its is reliability, speed and relative simplicity despite it being a lower-level language. Prior experience using this programming language was also a major factor in the decision process.

4.3 Functionality

4.3.1 Structure

The wallets in our blockchain consist of a Rivest-Shamir-Adleman or RSA asymmetric key pair [10]. The balance of a wallet is calculated by the sum of all transactions to and from this wallet present on the blockchain, as well as the block rewards earned. Each block mined by a miner has a set reward of 10 currency. Our blockchain uses a block structure based on the basic structure shown in figure 3.

The transactions have a structure quite similar to a block also containing a hash of the previous transaction and a sequence number. The header of the transaction also contains the wallet IDs of the sender and recipient, as well as the amount of currency to transfer. The body of the transaction contains the signature of the sender, once the transaction is signed with the private key of the sender. Once a transaction is created, it will be placed in a queue. When the transaction is signed, the miner will process it. If the sender wallet belonging to a transaction has insufficient currency, it is removed from the queue, otherwise, it will be added to the blockchain contained within the next block.

[12]No aspects of the workings of a regular proof of work blockchain should be compromised in the implementation of proof of useful work.

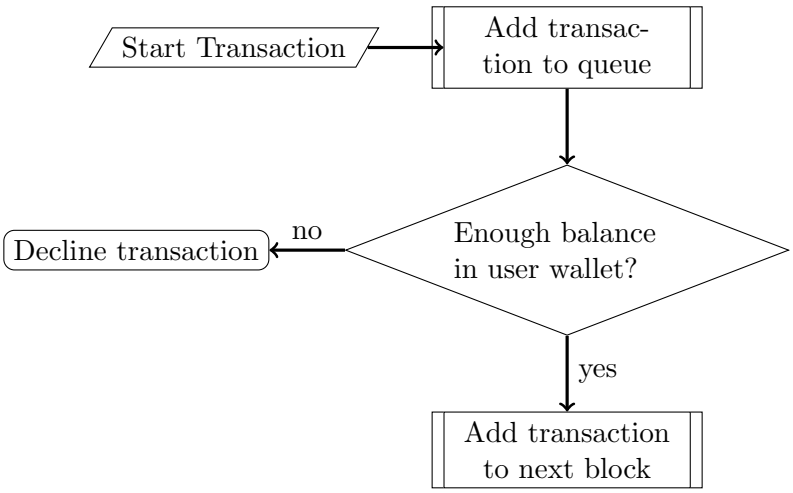

Figure 5: An illustration of the transaction process

4.3.2 Hash function

In order to implement useful work without compromising any decentralization or security, a new hash-like work function must be created of which execution is useful.

Our hashing function bears some resemblance to the SHA-2 hash family, but is slightly more simplified to accommodate our useful work implementation. First, the input data is broken into 512-bit chunks. If the data length is not a multiple of 512, the chunk will be extended to a length conforming to $512 \mid length$ and the added length is padded with bits with set to zero. The final 64 bits of the final chunk are set to the length of the data. All the chunks are exactly 512 bits long. These chunks consist of 16 32-bit integers. This is needed to be able to compress the data into the final 256 bits. The initial values (IV) for the hash are initialized with a set string, and then every chunk goes through a predefined process consisting of bitwise operations and rotations. After this process, the data is added to the initial values. When all the chunks have been processed, all the values are appended to form the final hash 256-bit. This process is represented schematically in figure 6 and its pseudocode can be found in appendix A.1

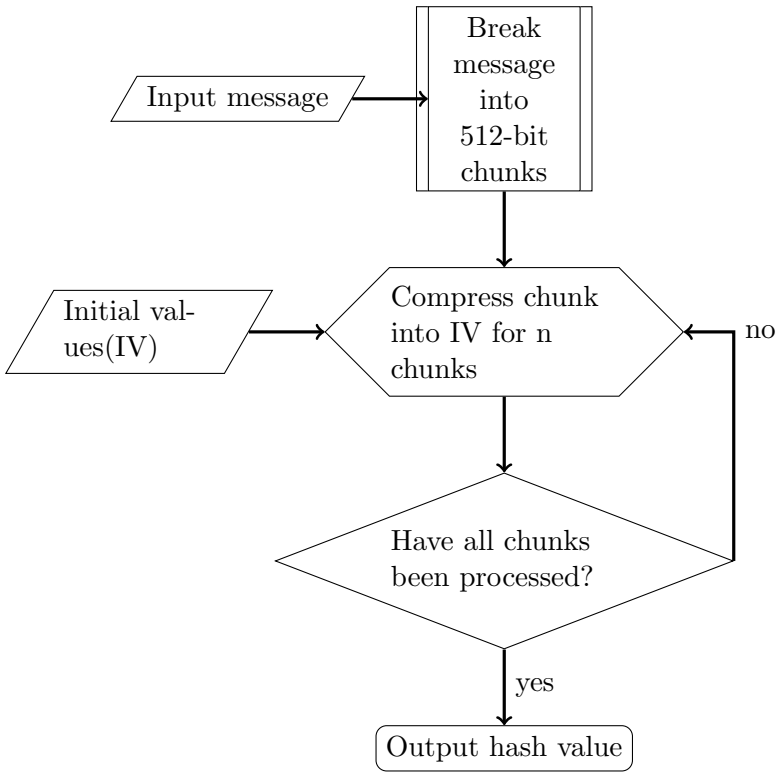

Figure 6: A simplified illustration of our hash function.

4.3.3 Factorization

There is a project similar to Folding@Home [4] called NFS@Home [3] by the California State University, Fullerton, that implements distributed computing for integer factorization using a general number field sieve. Because of time constraints, a less complicated quadratic sieving [14] algorithm has been implemented for the factorization process, the pseudocode for this can be found in appendix A.2. First, our hashing algorithm is used to get a 256-bit hash from the block. For this implementation, the first 64 bits are used, resulting in a 64-bit unsigned integer. Any number below 10^9 cannot be classified as useful, as it has already been factorized before. The chance that a number falls in this range is:

$$\frac{4.3 \cdot 10^9}{9.2 \cdot 10^{18}} = 2.3 \cdot 10^{-10} \tag{9}$$

A 64-bit unsigned integer can hold the numbers $\left[0, 9.2 \cdot 10^{18}\right]$. This integer is then factorized into primes, meaning that every factor has to be a prime number. This is because exactly half of all integers would be divisible by two, making one half easier than the other half if it were required to give only two factors. Instead, a miner has to compute all factors (e.g., $94\,868 = 2 \cdot 2 \cdot 37 \cdot 641$). When all the factors have been calculated they are added to the block, then the hash of the block with the added factors will dictate whether the block is valid or not. The block's hash is modified to reflect the result of this factorization process[13].

The general running time for the factorizing process of an integer n can be calculated as follows:

$$T = e^{(1+o(1))\sqrt{\ln n \ln \ln n}} \tag{10}$$

This raises another issue because if miners can estimate the complexity, a lazy mining problem[14] occurs. To combat this, an additional difficulty will be set, transforming the current difficulty parameter into a difficulty function that checks whether the running time is greater than t (a running time constant set on the blockchain). With all these new steps in place, the validation of a traditional proof of work blockchain is still in place, but the difficulty will be lowered keeping in mind the added complexity of the factorization process. The whole process is illustrated in figure 7.

4.4 Hash function evaluation

As discussed in section 2.1.1 hash functions have some fundamental properties that define their functionality. The hash function will be analysed by

[13] see 3.2
[14] see 3.2.1

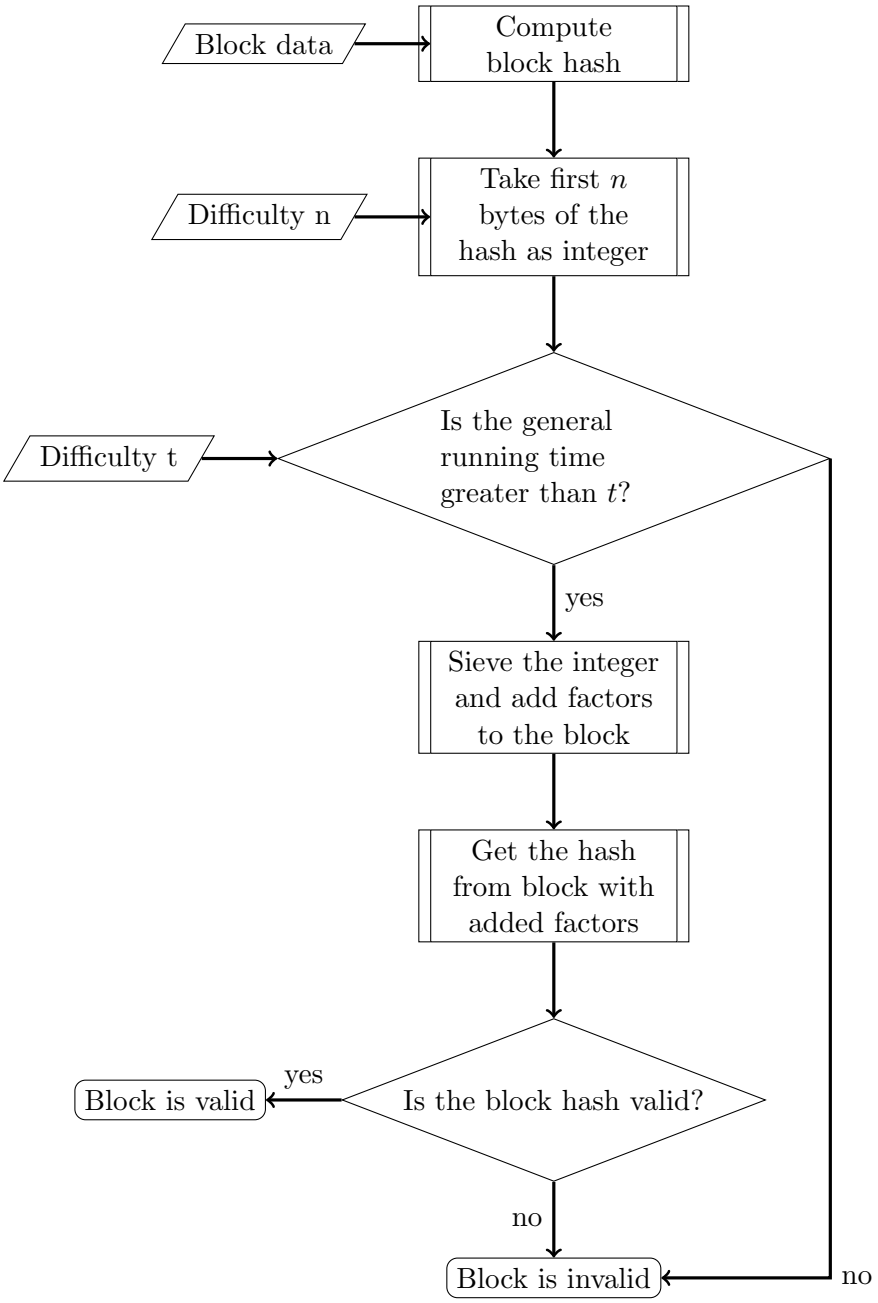

Figure 7: An illustration of the block validation process

calculating the Spearman rank correlation coefficient. Given the correlation coefficient it should be clear whether or not there is any correlation between the input and output of the hashing function. The Spearman correlation coefficient is calculated as follows:

$$r_s = 1 - \frac{6 \sum_{i-1}^{n} d_i^2}{n(n^2 - 1)} \tag{11}$$

where d_i is the rank number of two paired variables. To increase the reliability of the analysis a sample size of 5 million pairs of inputs and outputs has been generated using the hashing function.

```
=== RUN   TestSumSpearmanRhoCorrelationCoefficient
    dullhash_test.go:74: spearmanr correlation: 0.0005696983010621087,
    associated p-value: 0.2027045585947238, batch size: 5000000
--- PASS: TestSumSpearmanRhoCorrelationCoefficient (20.04s)
```

$r_s \in [-1, 1]$ where 1 is a full positive correlation, -1 a full negative correlation, and 0 no correlation. In this case $r_s \approx 0.00057$. As seen in the results, the correlation between inputs and outputs is minimal.

5 Conclusion

Using hash function-integration[15] useful work in the form of prime factorization of integers can be achieved. The proof of work blockchain will retain all traditional characteristics with regards to security, decentralization and trust. Out of the considered methods[16] integer factorization was determined to be the most suitable useful work problem.

That is not to say, however, that the current solution is flawless. The complexity of factorizing an arbitrary integer is not constant, which leads to a lazy mining problem[17]. If this problem is left unaddressed, this can lead to a miner selecting only the least complex numbers to maximize the amount of hashes they can calculate in a given period. The result of this has not been tested, but it could range from a slight loss in the amount of useful work done to reducing the useful work to a negligibly low proportion.

A possible solution is to adjust the difficulty of a block in order to account for the complexity of the hash function. Instead of setting the hash function output criterion to simply accept all hashes below a certain value, the number that is to be factorized could be restricted to a predefined range. A rough estimation of the complexity can be made using equation 10, for example. Still, this would likely only solve the lazy mining problem partially, as one cannot know in advance exactly the complexity of the hash function for each input without calculating it first.

In order to make useful work implemented in this[18] way completely feasible, more research is required. This will presumably involve either a complex difficulty function or choosing a different type of useful work which has a more uniform complexity distribution. Suitable candidates for this include but are not limited to optimization problems and the problems which are currently being solved by @Home projects, although some of these may be less suitable due to the requirement for authority.

5.1 Hash function implementation

The minute positive correlation can be attributed to coincidence. If the sample size was infinitely large, it would be 0. It therefore satisfies the criterion of a uniform distribution of outputs.

5.2 Discussion

The methods used to test the viability of the hash function use a statistical approach over a more fundamental mathematical one. The statistics suggest that the hash function works and has the desired characteristics, but more

[15] see 3.2
[16] see 3.3, 3.4, 3.5
[17] see 3.2.1
[18] using hash function-integration, see 3.2

research should be performed to confirm this is in fact the case. This should also be done to the hash function with prime factorization integrated.

5.3 Outlook

Given enough time, the concept of proof of useful work could change the blockchain landscape and have a considerable impact. Proof of stake has widely been considered as the only viable replacement for proof of work. In terms of electricity consumption, this may be true, but it also has a wide array of critics who are sceptic as to its security and scalability. Proof of useful work if worked out further for other implementation could be something refreshing and new possessing the secure qualities of a proof of work blockchain while perhaps having a significant impact on scientific research or any of the other mentioned applications.

6 Acknowledgements

Special thanks to our mentor Garmt de Vries-Uiterweerd, Ph.D. (Christelijk Lyceum Zeist) for his continued support and feedback on our paper's progress.

References

[1] *Berkeley SETI*. URL: http://seti.berkeley.edu/ (visited on 08/01/2022).

[2] Best. *Energy consumption of a Bitcoin (BTC, BTH) and VISA transaction as of October 2021*. 21st Oct. 2021. URL: https://www.statista.com/statistics/881541/bitcoin-energy-consumption-transaction-comparison-visa/ (visited on 07/12/2021).

[3] California State University Fullerton. *NFS@Home*. Sept. 2009. URL: https://escatter11.fullerton.edu/nfs/.

[4] *COVID-19*. July 2021. URL: https://foldingathome.org/diseases/infectious-diseases/covid-19/.

[5] *Ethereum Archives*. URL: https://www.theblockcrypto.com/data/on-chain-metrics/ethereum.

[6] *Ethereum Energy Consumption Index*. Sept. 2021. URL: https://digiconomist.net/ethereum-energy-consumption.

[7] *Ethereum ETH Network Difficulty Chart - 2Miners*. URL: https://2miners.com/eth-network-difficulty.

[8] *Ethereum ETH Network Hashrate Chart - 2Miners*. URL: https://2miners.com/eth-network-hashrate.

[9] *Frequently Asked Questions (FAQs) - U.S. Energy Information Administration (EIA)*. URL: https://www.eia.gov/tools/faqs/faq.php?id=97&t=3.

[10] IETF Trust. *PKCS 1: RSA Cryptography Specifications Version 2.2*. 2016. URL: https://datatracker.ietf.org/doc/html/rfc8017 (visited on 13/01/2022).

[11] Nathan Kirsch. *GeForce GTX 1070 Ethereum Mining - Small Tweaks For Great Hashrate and Low Power*. June 2017. URL: https://www.legitreviews.com/geforce-gtx-1070-ethereum-mining-small-tweaks-great-hashrate-low-power_195451.

[12] Arjen K Lenstra et al. 'The number field sieve'. In: *The development of the number field sieve*. Springer, 1993, pp. 11–42.

[13] Laura M. *The Best Ethereum Mining Hardware*. Jan. 2021. URL: https://www.bitdegree.org/crypto/tutorials/ethereum-mining-hardware#best-gpu-for-mining.

[14] Carl Pomerance. 'Smooth numbers and the quadratic sieve'. In: *Proc. of an MSRI workshop, J. Buhler and P. Stevenhagen, eds.(to appear)*. (2008), pp. 1–10.

[15] Jonathan Ponciano. *Cardano Surges During $300 Billion Crypto Crash As Musk Eyes Sustainable Bitcoin Alternatives*. May 2021. URL: https://www.forbes.com/sites/jonathanponciano/2021/05/13/cardano-surges-during-300-billion-crypto-crash-as-musk-eyes-sustainable-bitcoin-alternatives/?sh=4a9c9e36259e.

[16] The AlphaFold team. *AlphaFold: a solution to a 50-year-old grand challenge in biology*. Nov. 2020. URL: https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology.

[17] *The Go Programming Language*. URL: https://go.dev/ (visited on 07/01/2022).

A Pseudocode

A.1 Hash function

```
Initialize ASCII values "dullhash is a poor hash function"
as 8 32-bit unsigned integers:

h0 := 0x64756C6C
h1 := 0x68617368
h2 := 0x20697320
h3 := 0x6120706F
h4 := 0x6F722068
h5 := 0x61736820
h6 := 0x66756E63
h7 := 0x74696F6E

Break data into 512-bit chunks
 for each chunk
    Initialize current hash variables
    a = h0
    b = h1
    c = h2
    d = h3
    e = h4
    f = h5
    g = h6
    h = h7

    Main loop for compression function
    for 8 times
        for i from 0 to 16
            x := (a xor d) xor (chunk[i] rightrotate 11)
            y := f xor h xor (x and e)
            z := (b leftshift 10) or (y rightshift 22) xor c

            a = b + x
            b = h
            c = f
            d = e + z
            e = d + y
            f = a
            g = c
            h = g
```

```
Add chunk to hash value
h0 = h0 + a
h1 = h1 + b
h2 = h2 + c
h3 = h3 + d
h4 = h4 + e
h5 = h5 + f
h6 = h6 + g
h7 = h7 + h

Append for final hash (big-endian)
hash := (h0 leftshift 224) or (h1 leftshift 192) or
    (h2 leftshift 160) or (h3 leftshift 128) or
    (h4 leftshift 96) or (h5 leftshift 64) or
    (h6 leftshift 32) or h7
```

A.2 Quadratic sieve

Choose smoothness bound B in our case 10000

s :=$\pi(B)$

for i **from** 1 **to** $s + 1$ **do**
a_i := $\lfloor \sqrt{n} \rfloor$
b_i := $a_i^2 - n$
while b_i **is not** B Smooth **do**
v_i := $[e_{i,1} \bmod 2, ...e_{i,s} \bmod 2]$
$T \subseteq \{1, ..., s+1\} \sum_{i \in T} : v_i = 0 \in Z_2$
x := $\prod_{i \in T} a_i \bmod n$
y := $\prod_{j=1}^{s} p_j^{\sum_{i \in t} e_{ij}}$

if x **is not** (-y **mod** n) **and** x **is not** y **then**
break
if x **is** y **then**
return gcd(x - y, n), gcd(x + y, n)