Blockchain simulation and analysis on mining weaknesses

Author:
Aurélia Autem

Supervisor:
Dr. Eduardo Solana

Assistant supervisor:
Ricardo Abraham Wehbe

Department of Computer sciences
University of Geneva
Switzerland
July 26, 2019

Contents

Chapter 1

Introduction

1.1 Abstract

A blockchain is a list of blocks linked together using cryptography. Each block holds a list of transactions and the previous block's hash.
The first idea of the blockchain came from 1991 by Stuart Haber and W. Scott Stornetta but it was really used the first time by Satoshi Nakamoto in 2008 to serve its most famous application, the cryptocurrency Bitcoin.

The main advantage of blockchain is the decentralization, it means there is no need of a third party to validate transactions but it's still keeping essential properties as integrity, durability, reliability and longevity.

To ensure security, blockchain rests on cryptography methods like signatures with RSA encryption to protect transactions and hash functions for the mining process, which is the way to add a block in the blockchain.
This paper's goal is, first, to understand how mining works by simulating a blockchain. Then, we'll analyze the weaknesses and possible attacks on mining.

1.2 Blockchain structure

In this part, we'll try to understand in more details the blockchain's components and how they work together.
Bitcoin is a specific implementation of the blockchain but it's the starting point of our analysis.

As we mentioned before, the blockchain is based on a decentralized implementation. Actually, it's based on a peer-to-peer network.

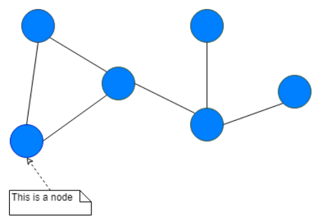


Figure 1.1: Peer-to-peer network

Each node has three responsibilities :

- Keeping a copy of confirmed transactions, which is the blockchain itself.
- Validating a transaction if it's following the rules.
- Sharing information with their neighbors such as unconfirmed but valid transactions and mined blocks.

1.2.1 Transaction

The main feature of the blockchain is to record transactions.
A transaction is composed by a list of inputs and a list of outputs (see Figure 1.2). Outputs can include the payer itself .

While doing a transaction, one wants to be sure that the payer is the true owner of the money. Then, in order to check this, we use digital signatures :

- The inputs of the transaction are encrypted with the public key of the payer.
- To prove he owns the coins, the payer "unlock" them by decrypting the signature with his private key.
- Then, he "locks" the outputs with the public key of the payees.

Technically, this is done with scripts which accompanied each input and output (see [1] for more details).

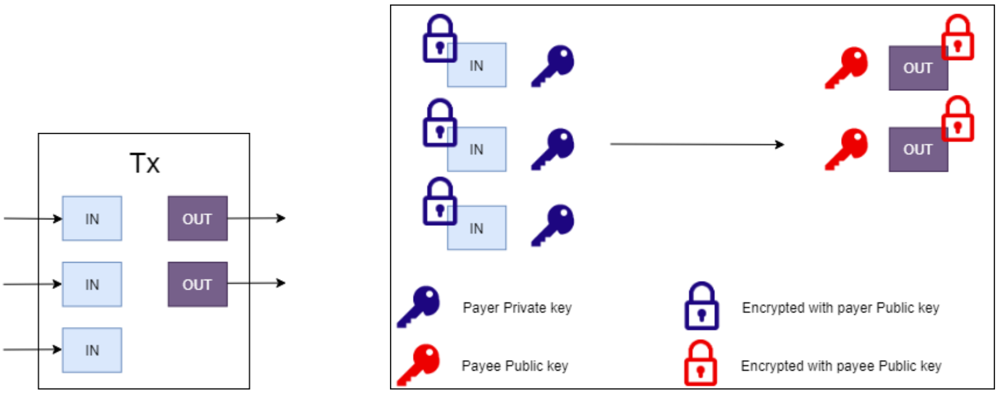


Figure 1.2: A transaction diagram and digital signatures

1.2.2 Block

Now that we have a transaction, we can transmit it to a node of the network. There, it needs to pass verification and if it's correct, it will be broadcast to other nodes.

Once the transaction is accepted by a node, it will stay in a special area called the Mempool (Memory pool), this is where all unconfirmed transactions wait to be added in a block. A node can prioritize the transactions in its Mempool, especially if he's running out of memory, he can choose the order of arrival or, more probably, the highest transaction fee.
When a node receives the information that a new block has been added to the blockchain, he removes all new confirmed transactions from his Mempool.

To form a new block candidate, a miner gathers transactions from the Mempool. Then, he will try to mine this block to add it to the blockchain. The miner will also add a block header which gives more information about the block (see 1.2.3).

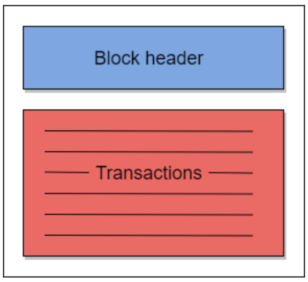


Figure 1.3: A transaction diagram and digital signatures

1.2.3 Block header

A block header is a summary of the block, it's like its metadata. A block header contains six fields :

version	The block's version	4 bytes
hashPrevBlock	The previous block's hash	32 bytes
hashMerkleRoot	The Merkle root representing all transactions in the block (see 1.2.4)	32 bytes
time	The Unix time at which the block header was hashed	4 bytes
target	This is a shortened version of the target (see 1.3.1)	4 bytes
nonce	A random number	4 bytes

We'll see later that block headers are used for mining (see 1.3).

1.2.4 Merkle root

As we've just seen, one of the block header fields is a merkle root. Conceptually, it represents a fingerprint of the transactions' list and concretely it's just a hash.
The blockchain uses Merkle trees for two reasons :

- To have a lightweight representation of the transactions, because it results with only a hash.
- To be able to check if a transaction exists in a block without knowing all transactions in this block.

Merkle trees use an hash function, for blockchains, they use the same function than mining, which is SHA256(SHA256(.)).

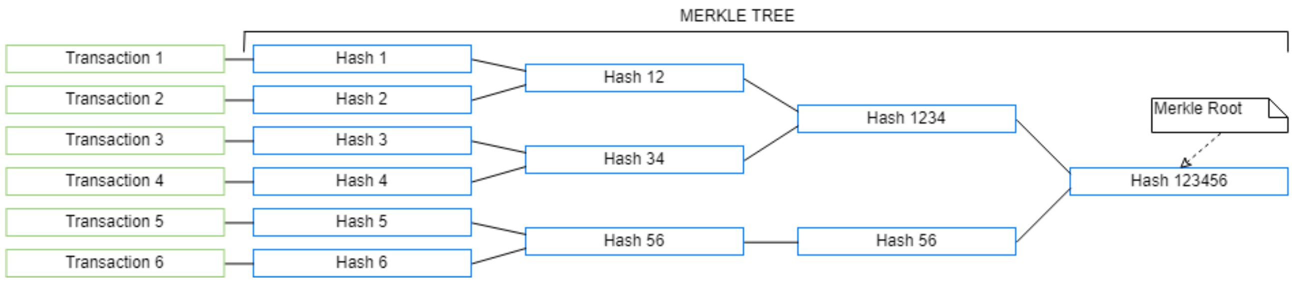


Figure 1.4: A merkle tree

The main advantage of this technology is the speed and the ease to verify if a transaction belongs to a block. We don't need to know all transactions of the block and we don't even need to reveal any data from the transactions but only their hashes.

For example, if we want to check if the transaction 3 belongs to the block in our previous figure. We have to know its hash (Hash 3), Hash 4, Hash 12 and Hash 56. With those hashes , we can reconstruct the Merkle root. If it's the same, this means that the transaction 3 belongs to the block and that no transaction has been modified so the whole block is correct.

1.2.5 The network

With all the concepts presented above, we can create a transaction, add it in a block, which will be mined thanks to its block header. Now, let's see how the nodes in the network secure the blockchain together.

The strength of the blockchain lies in its network, because each node keep a complete or partial copy of the blockchain. To keep the network updated, the nodes constantly share information between them. Typically, to broadcast a new transaction or a new mined block :

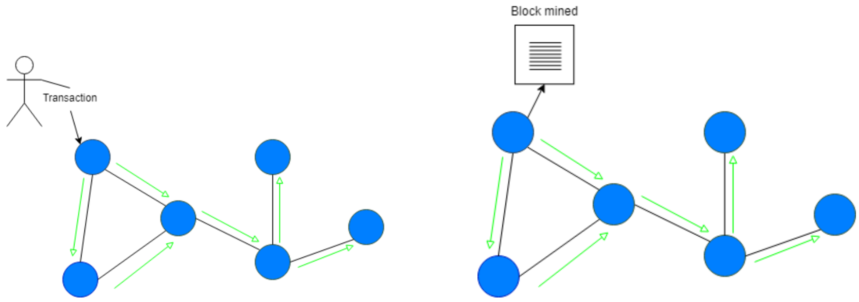


Figure 1.5: Communication inside the network

Then, each node updates its own version of the blockchain. Now, one can wonder what happens if a node receives two different mined blocks at the same time ?

The node will fork the blockchain and have two versions of it and he will keep accepting blocks for both chains. As long as they have the same length, the node will choose to work on one of the chains but if one chain becomes longer, the node will keep it and forget the other one.

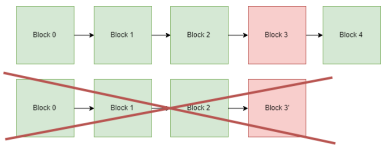


Figure 1.6: Fork chains

The blockchain is always the longest chain created because it's the one that has requested more work of mining. This means that to have control over the blockchain, an attacker should have the majority of the computational power (at least 51%).

1.2.6 Bitcoin Block example

Bitcoin blockchain is public, we can follow the evolution on web sites like www.blockchain.com ([2]). We can see the fields we described and some other details, for example :

Summary	
Height	573522 (Main chain)
Hash	000000000000000000021d691b53c5fd815f3c87b2681ba0d5410245769515ad1
Previous Block	000000000000000000016b35a6ede34250eb4ab437c19a3890cd233d4e01dc7f3
Next Blocks	0000000000000000001fd325c4ad75768248fc8b38c3239ad86d03db7702ed40
Time	2019-04-27 21:19:37
Difficulty	6,353,030,562,983.98
Bits	388779537
Number Of Transactions	2824
Output Total	1,060.84986755 BTC
Estimated Transaction Volume	260.30901865 BTC
Size	1148.763 KB
Version	0x20000000
Merkle Root	e33ebb8eee10f32cb071b4d0d6f11f0e69fa2a033532d80c7b6458c7fccb821e
Nonce	2454731301
Block Reward	12.5 BTC
Transaction Fees	0.10361587 BTC

Figure 1.7: Example of a Bitcoin Block

1.3 Mining in blockchain

In this section, we give a general view of the mining process and the math involved.

1.3.1 Proof of work

As we've seen before, blockchains lie on networks. Some nodes in these networks are miners, their role is to create a block of transaction and to mine it in order to add it to the network.

How do they mine it ?

They hash the block header with a specific function : $H_M(x) = SHA256(SHA256(x))$.
This is an easy and fast operation for miners but they need to fulfil a condition : the resulting hash has to be under a target value.

The target In the block header, this target is described by a field, which is 4 bytes long.
Let's take an example : if the target field is :

Target : 0x1207e540

12 is the exponent, it's an hexadecimal number and represents 18 in base 10. This means, the target will be 18 bytes long.
07e540 is the coefficient. The exponent gives us a long series of zeros and we replace the 3 first bytes by the coefficient.

This will give us : 07e540000000000000000000000000000000 .

The miners have to find an hash lower than the target value. We can see this condition from another angle, the hash will be 256 bits (64 bytes) long but the target is shorter. So there is a difference of bits, let's note d, this means that the hash will have to start with at least d zeros to fit the condition.

The target is not chosen randomly, it is set so a block will take about 10 minutes to be mined. This way, the number of blocks added in the network is controlled and it let enough time for the mined blocks to be broadcast in the network.
The difficulty is updated about every two weeks, every 2016 blocks exactly. It depends on the expected time (2016 x 10 minutes) and the actual time.

The nonce The last question remaining is how do miners affect the hash. They use another field of the block header, the nonce. This is the only field the miner can change, so the nonce is simply a random number and the miner's goal is to find the right random number which gives an hash lower than the target.

This describes an algorithm called the proof-of-work.

The rewards Then, we have a working protocol for mining but one can wonder why nodes will be willing to work for the blockchain.
It's quite simple actually, once a node has mined a block, he receives a reward (in Bitcoin if he works for Bitcoin blockchain).This is a way to motivate miners to work according to the rules for the blockchain.

Miners have also another way of winning money, the payers can add a fee to their transaction. This transaction fee will be taken by the miner which confirm the transaction in a mined block.
As we've seen before, miners have a Mempool and they will prioritize the transactions according to the fees they can obtain. This is a way for payers to speed up transactions in the network which is often highly congested.

1.3.2 Hash functions

In this paper, we'll analyze the potential consequences if the hashing function for mining is broken. So let's recall some properties about hashing functions.

The function used by mining is $H_M(x) = SHA256(SHA256(x))$.
Essentially, we can describe hash functions as deterministic functions, easy and fast to compute, whose output completely changes if the input changes and knowing the output gives no information at all on the input.

More precisely, hash functions, like SHA256, hold three properties :

1. Pre-image resistance : knowing y_1, it's difficult to find x_1 such that $h(x_1) = y_1$.
2. Second pre-image resistance : knowing x_1, it's difficult to find x_2 such that $h(x_1) = h(x_2)$.
3. Collision resistance : it's difficult to find x_1 and x_2 such that $h(x_1) = h(x_2)$.

Chapter 2

Breaking mining if SHA256 is broken

We know that mining is based on SHA256 so, in this part, we'll see different techniques (inspired from [1]) to win against traditional mining if SHA256 is broken.

2.1 Pre-image with fixed Merkle root

First, we analyze the probability for an attacker with a pre-image oracle to break mining, i.e. to get an high probability to solve the PoW before the other nodes in the network.
If we suppose we can have a pre-image oracle for SHA256, then we will be able to build an oracle for $H_M(x) = SHA256(SHA256(x))$ by applying the first oracle twice.

2.1.1 Input set of H_M

As explained before, a miner applies the hashing operation $H_M(x)$ to the block header. A classic miner only controls the value of the nonce but an attacker will try to control more.

The version, the previous block's hash and the merkle root are fixed fields.

For the time field, the system allow a range of 7200 seconds around the current time. So, over the 32 bits dedicated for this field, an attacker will be able to control about 13 bits.
For the target, the protocol will check if the target value is at most the one defined by the consensus of the network, this means the attacker can control about 28 bits.
For the nonce, like any miner, the attacker can control the 32 bits allowed to it.

The block header's size is $4 + 32 + 32 + 4 + 4 + 4 = 80$ bytes $= 640$ bits.
An attacker can control $b = 13 + 28 + 32 = 73$ bits on the input value, which means he has 2^b possibilities to call the hash function H_M.

2.1.2 Output set of H_M

The result of H_M is an hash from SHA256, so it has a length of $n = 256$ bits and there are 2^n possibilities of outputs.

Then, we've seen that to fulfill the condition given by the target, the hash needs to start with a specific number of zeros, let's note d zeros.
In reality, matching with the coefficient may require more effort (see A for more details).

This means that there are 2^{n-d} hashes lower than the target.

2.1.3 Probability of breaking mining

By calling H_M, one has a probability of $\frac{2^{n-d}}{2^n}$ to get a valid hash.
That way, we can get the number of correct pre-images i.e in the input set, how many inputs will end as a correct hash.

$$proba\_of\_correct\_hash \times \#possible\_inputs = \frac{2^{n-d}}{2^n} \times 2^b = 2^{b-d} \tag{2.1}$$

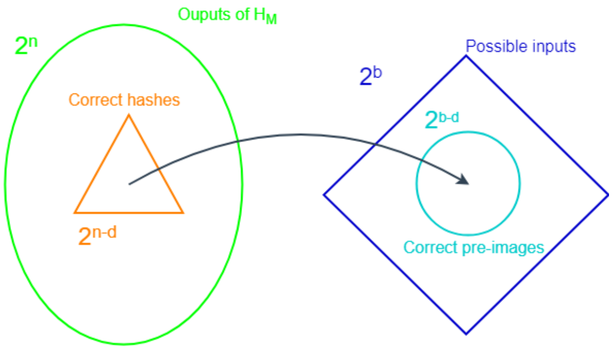


Figure 2.1: Probability of getting a correct pre-image

In his attack attempt, the attacker queries the pre-image oracle for specific target hashes. So he will choose a correct hash (triangle set) and hope that the output is one of the correct pre-images. This is the probability that he gets an accepted pre-image.

$$\frac{\#pre-images}{\#accepted\_outputs} = \frac{2^{b-d}}{2^{n-d}} = 2^{b-n} \tag{2.2}$$

Then, to get the real probability of success, we need to add a constraint on the number of queries allowed to the attacker. Because, otherwise, the attack won't be faster than the traditionnal mining. So let's consider the attacker can query the oracle 2^a times.

$$P_{success} = 2^a \times 2^{b-n} = 2^{a+b-n} \tag{2.3}$$

Finally, we can estimate this probability of success. Let's take $a = 80$ so the attacker have 2^{80} tries (above this value, he may need more than 10 minutes to complete his attack). So, we have $P_{success} = 2^{80+73-256} = 2^{-103} \approx 10^{-32}$, which completely negligible.
Then, we can conclude that having a simple pre-image oracle doesn't help to break mining.

2.2 Pre-image with variable Merkle root

2.2.1 Random merkle root

In the previous section, we've assumed that the merkle root is a fixed field, because we've supposed that the list of transactions was also fixed.

But an adversary could try work backwards, to choose a random merkle root and to try to reconstruct the merkle tree leading to semantically valid transactions. By contrast with the previous method, the transactions are created by the attacker himself. However, even without figuring out any probabilities, we can conclude that this method is infeasible. That's because, the reconstructed transactions will have to contain valid outputs and valid signatures.

2.2.2 Coin base transactions

However, there is an exception for the previous remark based on two observations. First, the system doesn't have a constraint about the number of transactions in a block and second, each block has a coinbase transaction, which creates money as reward for the miner.

An attacker could create a block with only a coinbase transaction. These transactions have a fixed prefix and suffix and they have a length-variable field up to 100 bytes, scriptSig.
Then, the attacker can create a valid block by finding a valid merkle root : $h(a||x||b) = T$, where x is the merkle root, T is the target and a and b are the prefix and suffix of the block header.

Finally, he'll have to solve : $h(b||y||d) = x$, where y is the scriptSig and b and d are the prefix and suffix of the coinbase transaction. So the merkle root match with a valid transaction.
($||.||$ means concatenation)

Both prediction will have a pre-image with high probability but if not, the attacker can try with another target or with different length for the scriptSig. Nevertheless, his probability of success is very high.

2.3 Bounded pre-image oracle

2.3.1 Construction of the oracle

In reality, a hash function breakage can imply more powerful tools than a simple pre-image oracle. To cover this possibly, we can construct a general oracle (see [1]).

This oracle takes as input : $(a, b, y_l, y_i, i, s) = (prefix, suffix, target\_low, target\_high, position, length)$. It will returns x_i such that : $y_l <= h(a||x_i||b) <= y_h$ or none (if there is no pre-image).

In other words, the prefix, suffix and length of the pre-image are fixed, this allows us to have control over the format of the pre-image and, in our context, to fix some fields of the block header. The value will be between a target range, this will help us satisfy the condition given by the target.
The position of the pre-image is also specified, this means the same x_i is returned on each call and a call on j will return $x_j \neq x_i$.

Technically, the suffix is added to keep a symmetry but it's not needed for our attack. Moreover, in reality this is usually the hardest part of the oracle to setup.

2.3.2 Attacks against mining

An attacker with access to our bounded oracle can simply call for $(headerBeginning, none, 0^{256}, target, 0, 32)$.

For recall, the target is 256 bits long, the block header is 32 bits and the headerBeginning is the beginning of the block header until the nonce. This will return a pre-image of 32 bits with the first fields of the block header and a correct nonce such that the hash is under the target value.

This kind of attack will completely break mining, which allows the attacker to create deep forks and then, reversing transactions or double-spend.

2.4 Second pre-image and collision

We've seen in the previous section vulnerabilities linked to pre-images. Let's have a look to second pre-images and collisions.

2.4.1 Attacks on blocks

Second pre-image Let's recall the theory about second pre-images : for a specific hash given by a specific input, a second pre-image oracle allow us to find another input giving the same hash.
In our context, one could want to replace a block by a new block with the same hash to maintain the blockchain valid, this way one could completely modify the blockchain.
However, this is concretely infeasible because more than just the hash, the new block has to be valid in terms of transactions. This means the transactions have correct inputs and correct signatures, the probability that the new block respects these constraints is almost zero.

Collision For collisions, the idea would be to create several blocks with the same hash and to insert them in the network. This would allow an attacker to be able to fork the chain and possibly double-spend or steal coins. However, following the same remark we did for second pre-images, the probability to create valid blocks is negligible.

To conclude, collisions and second pre-images are irrelevant to break mining.

2.4.2 Merkle roots

As recall the hash function H_M is also in merkle trees, then one could alter already mined blocks by changing transactions but with the same merkle root.

Blocks inside the blockchain This idea would be to change the transactions in a block by getting a merkle root with the same hash. As we expose before, the probability of getting valid transactions is very low. So the nodes will reject the modified block.

New added blocks However, the situation is different if the adversary focuses on the last block. The attacker can create a new block with different transactions but the same merkle root. Even if, this new created block is invalid, the attacker can send it in the network and this may cause the nodes to reject both blocks or even to accept the invalid block.
This attack was done in July 2015 ([3]), nodes were accepting invalid blocks and then, validating wrong transactions. Nowadays, a new version of the protocol has been released and these problems are solved.

The adversary can also double-spend coins by creating a new block with conflicting transactions according to the valid block using a collision or second pre-image oracle. Then, he can transmit both blocks in the network, this will fork the blockchain and may fool the vendor.

2.5 Conclusion

2.5.1 Consequences summary

We can sum up the different consequences of breakages on SHA256.

Breakage	Consequences
Pre-image	Complete breakage of the blockchain
Bounded pre-image	Complete breakage of the blockchain
Second pre-image	Double spending, steal coins
Collision	Double spending, steal coins

2.5.2 Contingency plans

All the attacks presented above are based on potential SHA256 vulnerabilities. We cannot guarantee that SHA256 will stay safe forever. However, we can notice it was created in 2001 and no weaknesses have been discovered yet so we can conclude SHA256 is quite robust (see a stackexchange conversation about SHA256 security, [4]).

In case SHA256 is broken, Bitcoin has a contingency plan (see [5]).
As we've seen this situation would be dramatic, an attacker could compromise the whole system, this includes the alert system.
The plan in this situation is to ask the users to shut down their clients and to hardcode the public keys of all addresses that have unspent outputs. Then, these keys will be used when a new version of the blockchain is released.

The code for all of this should be prepared but, in reality, this is not the case because, even if the consequences would be severe, the risk is very low.

Chapter 3

Attacks using mining weaknesses

In the previous chapter, we've seen that mining is based on SHA256 and we've analyzed the consequences if SHA256 is broken. In this chapter, we'll try to find some weaknesses to mining and use them to trick the process.

3.1 Double-spending

One of the very famous technic to trick a seller is called double-spending. As explained in [6], a double-spending attack is used by a buyer to foul the seller, following the next steps :

1. The buyer A broadcast a transaction AB in the network where he pays the seller B.
2. The buyer A creates another block with a transaction $\overline{AB}$ which invalidates the transaction AB.
3. The buyer A secretly mine a branch on this new block.
4. The buyer A waits that the seller B sends his product.
5. Once the buyer's branch is long enough, he broadcasts it which will unconfirm the transaction AB.

With this method, the attacker uses his bitcoin twice, that's why it's called double-spending.

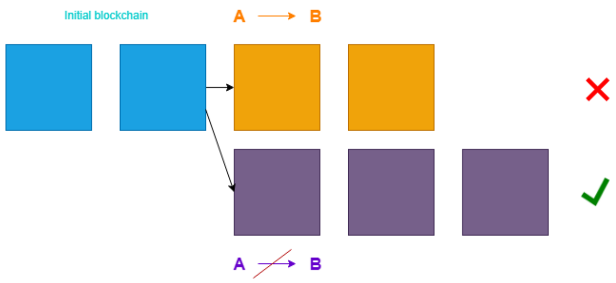


Figure 3.1: Double-spending - The longest chain will the purple one which invalidates the bitcoins sent to B while the seller has already sent his product

3.2 51% attack

3.2.1 What is it?

The main difficulty to achieve this attack is to be able to build a longer chain to invalidate the transaction which gives bitcoins to the seller. One possible solution is the 51% attack.
The 51% attack has always been an important concern about blockchains' security. The main idea is that an attacker will control the majority of the computational power, i.e. at least 51%.

The ability of someone controlling a majority of network hash rate to revise transaction history and prevent new transactions from confirming. (Bitcoin glossary [7])

In other words, an adversary with at least 51% of the network will be able :

- To prevent some users to do transactions, by systematically unconfirming theirs transactions. Because the attacker control the majority of the network, even if some nodes confirm the user's transaction, it will never be part of the longest chain.
- To reverse his own transactions, this is double-spending.

3.2.2 Is it possible to implement?

For Bitcoin To set a 51% attack up, an attacker needs to gather enough mining computers to get more than 50% of the computational power of the actual network.
Moreover, the attacker will have to supply enough electricity power to run those computers.

Now, the real question is how much will it cost. We can try to estimate the cost for Bitcoin (see [8]).

First, buying specialized computers for mining will cost about $2.4 million and $250 million in infrastructure to install these computers and the equipment needed (like ventilation).

Then, to power this structure, one will need around 30 Terawatts of electricity per year. For example, Morocco consumed 29 Terawatts in 2017 and Switzerland consumed 63 Terawatts the same year. All this electricity will cost around $2 million by day.

To sum up, a 51% attack against Bitcoin will cost $1.4 billion. This makes the attack almost impossible due to this huge cost, even for a government such as China (which is one of the biggest economy in the world), it will be very complicated to set up this attack.

What about other blockchains ? We've studied the feasibility with Bitcoin which is one of the most famous blockchain nowadays and its network is now very large. But one can wonder if the threat is more important with smaller blockchains.

Indeed, the cost for a 51% attack decreases for smaller coins. However, for every serious blockchain, there are still thousands or millions of nodes and anyways, it would more profitable to mine honestly and win coins through rewards.
On Crypto51, we can observe the cost to make a 51% attack on different cryptocurrencies.

Name	Symbol	Market Cap	Algorithm	Hash Rate	1h Attack Cost	NiceHash-able
Bitcoin	BTC	$208.66 B	SHA-256	71,147 PH/s	$1,104,830	0%
Ethereum	ETH	$31.62 B	Ethash	157 TH/s	$160,091	6%
Litecoin	LTC	$7.81 B	Scrypt	468 TH/s	$77,561	3%
BitcoinCashABC	BCH	$7.45 B	SHA-256	2,543 PH/s	$39,483	3%
BitcoinSV	BSV	$3.59 B	SHA-256	923 PH/s	$14,338	7%
Monero	XMR	$1.50 B	CryptoNightR	342 MH/s	$6,884	5%
Dash	DASH	$1.41 B	X11	5 PH/s	$6,598	6%
EthereumClassic	ETC	$867.77 M	Ethash	7 TH/s	$7,622	119%
Zcash	ZEC	$712.02 M	Equihash	5 GH/s	$19,835	6%
BitcoinGold	BTG	$466.69 M	Zhash	4 MH/s	$1,964	26%

Figure 3.2: Crypto51

Then, the probability remains low especially for the biggest blockchains. However, it can still happen (see [9]).
For example, EthereumClassic was attacked on January 2019 and it lost almost $1.1 million in one day.
Preventing this attack can be very complicated but there are still some ideas :

- Merging mining. Smaller cryptocurrencies can use mining power of larger ones so they become less vulnerable.
- Penalizing delayed blocks. The attacker mines blocks secretly and broadcasts them lately, a penalty will reduce the benefit of the attack.

- A detection algorithm for the 49% left. Once the attack is detected, the 49% left can try to acquire more power to stop the attack.

3.3 Selfish mining

In the previous section, we've considered an attack to trick vendors by taking advantages on the system. Now, we will study a method to trick the system itself.
As we mentioned before, a node wins a reward when he successfully mines a block and add it in the blockchain, the node earns the rewards as long as his block is in the blockchain.

We will present the method of selfish mining, where miners try to increase their rewards allowed for mining (see [10]).

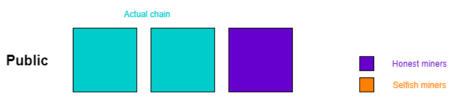
3.3.1 What is it?

First, let's suppose that all the computational power is divided in two groups : first group follow the selfish mining strategy and second group follow the classic mining strategy.

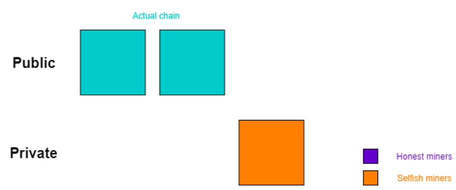
The main idea of selfish mining is to mine secretly blocks in advance so the miners create a longer chain than the others. Then, at the right time, they reveal their chain which will become the new longest chain, so they win the rewards and waste the work of the others miners.

Let's see in details how this strategy works, there are four important cases :

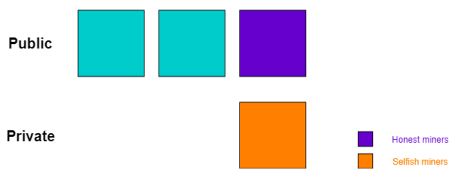
- Case 1 : The honest miners find a new block on the public chain, the selfish miners will just adopt the new chain.
- Case 2 : The selfish miners find a new block on the public chain, they will keep this block secret and then, there are two possibilities (Case 3 or 4).
- Case 3 : The selfish miners have secretly one block in advance but the honest miners find a new block first and broadcast it, in that case the selfish miners broadcast their secret block immediately. Since two blocks are broadcast almost at the same time, miners will have to choose on which block they mine after. All the selfish miners will mine after their block, and the honest miners will mine on the first they receive. Let's note γ the proportion of honest miners who mine after the selfish miners' block.
- Case 4 : The selfish miners have secretly one block in advance and they find a new one, so they keep this new block secret. Then, they will try to keep two blocks or more in advance the longest time possible. If they only have one block left, they publish their secret chain.



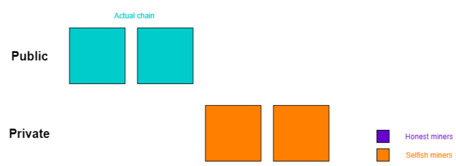
(a) Case 1



(b) Case 2



(a) Case 3



(b) Case 4

3.3.2 Is it feasible?

As we've just seen, this strategy is based on the fact that the selfish miners succeed in mining blocks ahead of the honest miners. To do so, it's obvious that their combined computational power will affect their chance of success, let's note α the computational power of the selfish miners.
As describe in [10], we can represent the selfish mining algorithm with a finite state machine.

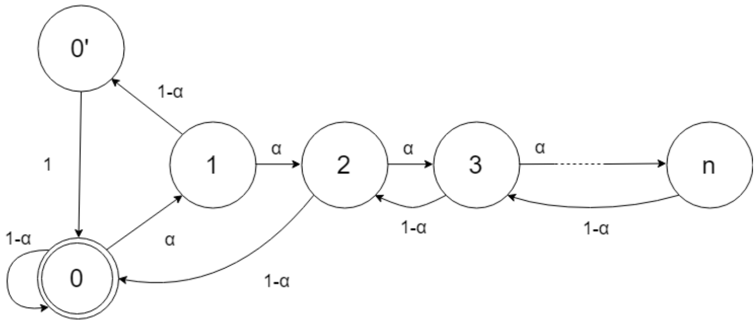


Figure 3.3: Selfish mining finite state machine

In this machine, the initial state is the state 0 and it corresponds to case 1. The state 1 corresponds to case 2, the state 0' to case 3 and the states 2 and more correspond to case 4.

We can compute the probabilities of this state machine, we note p_i the probability of being in state i. From these probabilities, we can compute the rewards the selfish and honest miners will win.

We can find more explanations about the formulas in this appendix B.

$$r\_honest = p_0.(1-\alpha).1 + p_{0'}.\gamma.(1-\alpha).1 + p_{0'}.(1-\gamma)(1-\alpha).2$$

$$r\_selfish = p_{0'}.\gamma.(1-\alpha).1 + p_{0'}.\alpha.2 + p_2.(1-\alpha).2 + P[i>2].(1-\alpha).1$$

Table 3.1: Revenue won by the honest and selfish miners according to α and γ

We can implement this finite state machine and try some values of α and γ.

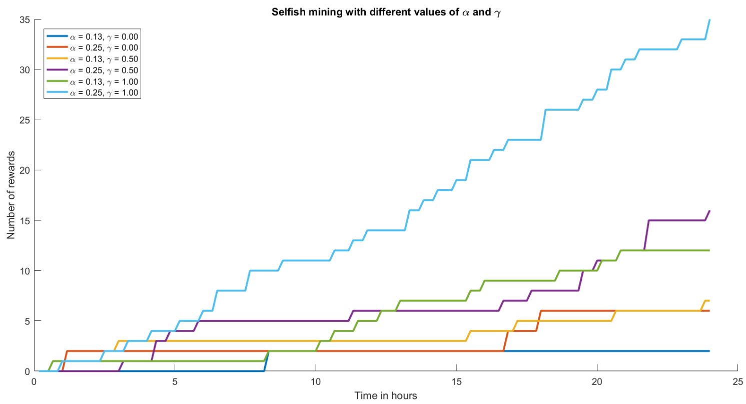
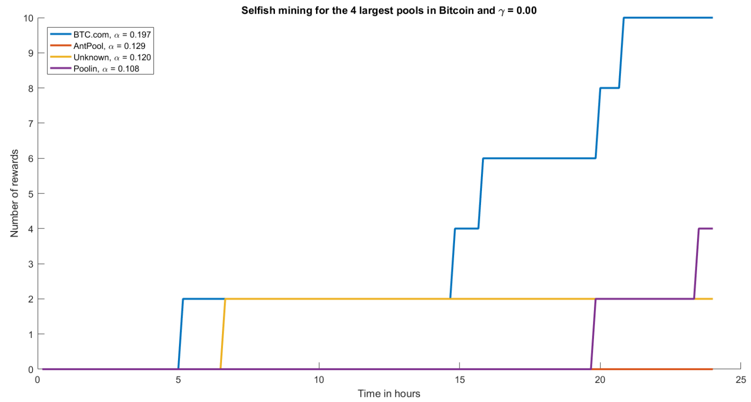
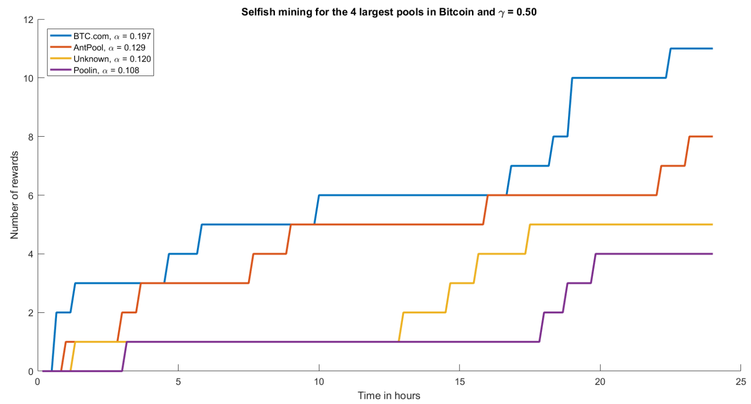
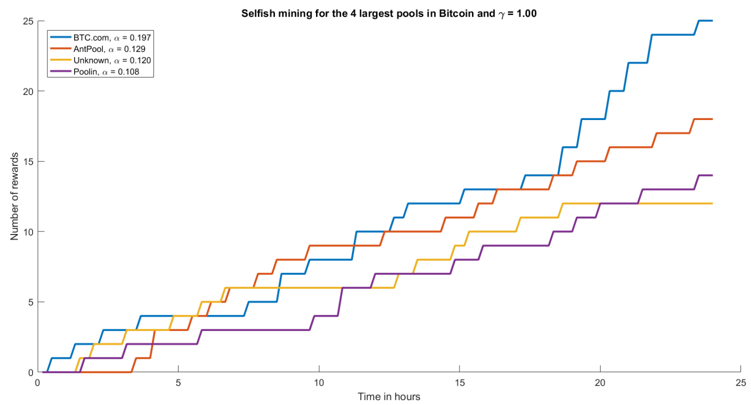


Figure 3.4: Rewards according to different α and γ

We see that γ and α are linked because the more γ increases the more the pool size α needs to increase to win the same reward. Let's try several values of γ with the hash rates of the four largest pools in Bitcoin network (see [11]).


Selfish mining for the 4 largest pools in Bitcoin and γ = 0.00
BTC.com, α = 0.197
AntPool, α = 0.129
Unknown, α = 0.120
Poolin, α = 0.108
Number of rewards
Time in hours


Selfish mining for the 4 largest pools in Bitcoin and γ = 0.50
BTC.com, α = 0.197
AntPool, α = 0.129
Unknown, α = 0.120
Poolin, α = 0.108
Number of rewards
Time in hours


Selfish mining for the 4 largest pools in Bitcoin and γ = 1.00
BTC.com, α = 0.197
AntPool, α = 0.129
Unknown, α = 0.120
Poolin, α = 0.108
Number of rewards
Time in hours

We can see that the pools will be able to win more if they can propagate their blocks fast enough, i.e. if γ is high enough.
Now, we need to know if it's worth the effort against the honest mining, to do so we can analyze the link between α and γ in the formulas for the revenues (Table 3.1) as it was done in [10].

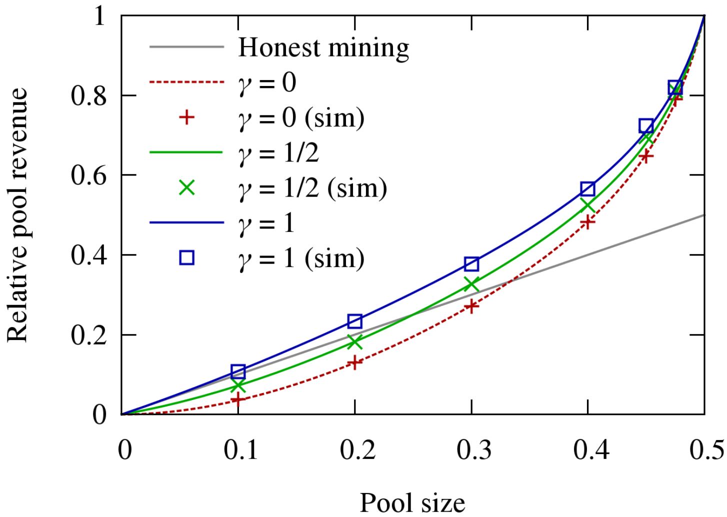


Figure 3.5: Revenue of the selfish miners for different values of α and γ

This figure shows the importance of γ. With an high γ, the pool will propagate its blocks to all the network very quickly so the selfish mining strategy will be efficient.
However, with a low γ, the honest miners will propagate their blocks quickly. If we take the extreme case of $\gamma = 0$, we'll need a pool with around 35% of the mining power. For Bitcoin, the largest pool has 19.7% of hash rate (see [11]) so it seems secured for low γ.

In Bitcoin, the protocol doesn't insure that γ, which represent the propagation rate of the pools' miners, will stay low. This is because when an honest miner has several chains of same length, he will choose to mine the first block he receives. Then, a pool can perform a Sybil attack, they use virtual miners who won't mine but when they detect that the honest miners have mined a new block, they propagate the pool's block so it will be broadcast faster and increase γ.

A solution to this problem would be to change the protocol and make the miners to choose randomly between chains of same length. This would lead to $\gamma = 1/2$ and it would require a pool with 25% of hash rate, which is hard to achieve.

Chapter 4

Quantum computing and blockchains' security ?

Since the 1990s, quantum computing has become an important field of research in computer science. In 1998, we've seen the first quantum computer with 3 qubits and since then a lot of improvements and researches have been done to build a stable quantum computer.

Let's anticipate the possible apogee of the quantum era and study how it could affect blockchains' security. We can watch some Youtube videos (see [12] and [13]) as an introduction to quantum computers.

4.1 What is Quantum computing ?

Quantum computing differs from classic computing by the way it store and manipulate information.

Let's have a quick reminder of how a classic computer works.

They use bits to represent data, a bit can be either 0 or 1. Those two states are represented though an electrical signal.

Figure 4.1: Coding of a bit

We process these signals with transistors to make logical gates, this is the basics of processors and computers architecture. To represent complex data, we create registers made of several bits. For example, a n-bits register can represent 2^n values / states.

Now, what about quantum computing ? (see [14]) As the name suggests it, this new technology is based on quantum physics, the big difference with classic computers is that there isn't bits anymore but qubits or quantum bits.

These qubits are not binary, either 0 or 1, but they exist in a superposition of 0 and 1. This is the superposition principle, we can understand it as probabilities so we commonly describe a qubit with the following formula:

$$\langle x| = \alpha . \langle 0| + \beta . \langle 1| \tag{4.1}$$

where α is the probability for 0 and β is the probability for 1.

It's important to highlight that a quantum computer isn't an evolution of classic computers like clusters can be. A whole new technology is involved heading to a new way of representing and manipulating data.

We won't go too far in the details about quantum physics and qubits but we can investigate two interesting questions.

How do we physically build a qubit ? There are different possibilities to build a qubit, we need to use a two-level system, a system where a quantum superposition can exist. For example, we could use photons and light polarization, where we measure how much the light is vertically or horizontally polarized.

One famous method is to use the spins of the electrons (see [15]) where we measure the spin angular momentum to describe the quantum state of the electron. This spin is affected by the magnetic field around the particle.

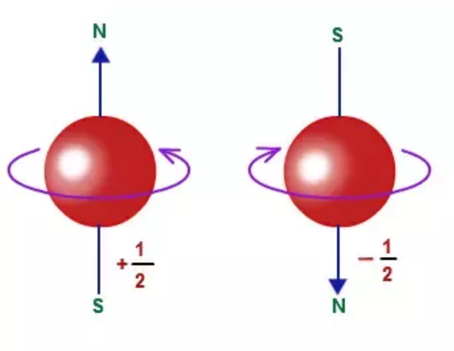


Figure 4.2: Electron spin from [16]

These methods are more complicated than electric signals for classic bits, it's harder to keep the quantum bits stable. Researches are working on stability of qubits in order to build bigger computers.

How can we recover the state form a qubit ? Quantum computers follow a rule called "Wave function collapse", which says that a superposed state can't be known entirely, we can only measure it and so reduce it to one value.
This means that even if we have a 2^n superposed state, we will reduce it to one result. Then a quantum computer can be used only for specific problems with specifically designed algorithms like Grover algorithm.

To conclude, a quantum computer will always be less polyvalent than its classical equivalent but it can be so much powerful for some specific problems. For example, it can be used for prime factorization, Peter Shor a scientist from Bell's lab found a quantum that could solve this problem.

4.2 Is quantum computing a treat to blockchains' security ?

Appendix A

Complements about the target

What does it imply to consider that the hash starts with d zeros ?

To simplify the analysis, we can assume that the condition of having an hash lower than the target is equivalent to having starting by d zeros, where d is 256 - exponent_length.

The "real" condition is harder to fulfill so how much work does it require if we only fulfill the second condition ? Let's suppose we find a hash starting with d zeros :

0...0$a_1 a_2 a_3 a_4 a_5 a_6$0...0

and we have the following target :

0...0$c_1 c_2 c_3 c_4 c_5 c_6$0...0

There are 3 possibilities :

- $a_1 < c_1$: then the condition is fulfilled.
- $a_1 = c_1$: we look at the next the bit and we are in the same situation for c_2, same for c_3, ..., c_5.
- $a_1 > c_1$: we need a more bit of effort to satisfy the condition.

For a_6, if $a_6 = c_6$, we also need one more bit of effort to satisfy the condition.
Finally, whatever the situation, we need only one more bit of effort.

Appendix B

Explanations on the revenues

In the chapter about selfish mining, we've calculated the revenues of the honest and selfish miners.

$$r\_honest = p_0.(1-\alpha).1 + p_{0'}.\gamma.(1-\alpha).1 + p_{0'}.(1-\gamma)(1-\alpha).2$$

$$r\_selfish = p_{0'}.\gamma.(1-\alpha).1 + p_{0'}.\alpha.2 + p_2.(1-\alpha).2 + P[i>2].(1-\alpha).1$$

Table B.1: Revenue won by the honest and selfish miners according to α and γ

Here is more explanations about these formulas :

$p_0.(1-\alpha).1$	The honest miners find a block on the actual chain, they win 1 reward.
$p_{0'}.\gamma.(1-\alpha).1$	The selfish and honest miners broadcast a block at the same time, then the honest miners find the next block after the selfish miners' one, the honest and selfish miners win both 1 reward.
$p_{0'}.(1-\gamma)(1-\alpha).2$	The selfish and honest miners broadcast a block at the same time, then the honest miners find the next block after their own one, so they win 2 rewards.

$p_{0'}.\gamma.(1-\alpha).1$	The selfish and honest miners broadcast a block at the same time, then the honest miners find the next block after the selfish miners' one, the honest and selfish miners win both 1 reward.
$p_{0'}.\alpha.2$	The selfish and honest miners have both found a block, but the selfish miners find their next block first, they publish both blocks and they win 2 rewards.
$p_2.(1-\alpha).2$	The selfish miners had a lead of two blocks but the honest miners find one, so the selfish miners publish their two blocks and they win 2 rewards.
$P[i>2].(1-\alpha).1$	Each time the selfish miners increase their lead above 2 blocks, they win 1 reward.

Table B.2: Explanations for the revenue of honest miners and selfish miners

Bibliography

[1] I. Giechaskiel, C. Cremers, and K. B. Rasmussen, “On Bitcoin Security in the Presence of Broken Crypto Primitives.”

[2] “Official blockchain website for bitcoin, ethereum and bitcoin cash.” [Online]. Available: https://www.blockchain.com/explorer

[3] “Bitcoin.org - report isssue : Some Miners Generating Invalid Blocks.” [Online]. Available: https://bitcoin.org/en/alert/2015-07-04-spv-mining

[4] “Security stackexchange - Security of SHA256 and Bitcoins.” [Online]. Available: https://security.stackexchange.com/questions/6458/security-of-sha256-and-bitcoins

[5] “Contingency plans - Bitcoin Wiki.” [Online]. Available: https://en.bitcoin.it/wiki/Contingency_plans

[6] M. Rosenfeld, “Analysis of Hashrate-Based Double Spending,” *arXiv:1402.2009 [cs]*, Feb. 2014, arXiv: 1402.2009. [Online]. Available: http://arxiv.org/abs/1402.2009

[7] “51% Attack, Majority Hash Rate Attack - Bitcoin Glossary.” [Online]. Available: https://bitcoin.org/en/glossary/51-percent-attack

[8] “Analysis: Bitcoin Costs $1.4 Billion to 51% Attack, Consumes as Much Electricity as Morocco,” Nov. 2018. [Online]. Available: https://cryptoslate.com/analysis-bitcoin-costs-1-4-billion-to-51-attack-consumes-as-much-electricity-as-morocco/

[9] S. Bandyopadhyay, “Can blockchains survive 51% attacks? - Data Driven Investor,” Jan. 2019. [Online]. Available: https://medium.com/datadriveninvestor/51-attacks-are-happening-more-frequently-2400ffe42c4f

[10] I. Eyal and E. G. Sirer, “Majority is not Enough: Bitcoin Mining is Vulnerable.”

[11] “Hashrate Distribution.” [Online]. Available: https://www.blockchain.com/pools

[12] “Les Ordinateurs Quantiques — Science étonnante #40 - YouTube (In French).” [Online]. Available: https://www.youtube.com/watch?v=bayTbt_8aNc

[13] “Quantum computing explained in 10 minutes | Shohini Ghose - YouTube.” [Online]. Available: https://www.youtube.com/watch?v=QuR969uMICM&t=414s

[14] “Qubit - wikipedia.” [Online]. Available: https://en.wikipedia.org/w/index.php?title=Qubit&oldid=906072254

[15] “Spin quantum number.” [Online]. Available: https://en.wikipedia.org/w/index.php?title=Spin_quantum_number&oldid=905710478

[16] “What is the spin quantum number? - Quora.” [Online]. Available: https://www.quora.com/What-is-the-spin-quantum-number

[17] R. Virk, “How I Cornered the Bitcoin Mining Market Using a Quantum Computer (Theoretically!),” Sep. 2017. [Online]. Available: https://hackernoon.com/how-i-cornered-the-bitcoin-mining-market-using-a-quantum-computer-9e5dceba9f92

[18] G. O. Karame, E. Androulaki, and S. Capkun, “Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin.”

[19] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, “On the Security and Performance of Proof of Work Blockchains,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS’16*. Vienna, Austria: ACM Press, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2976749.2978341

[20] D. Aggarwal, G. Brennen, T. Lee, M. Santha, and M. Tomamichel, “Quantum Attacks on Bitcoin, and How to Protect Against Them,” *Ledger*, vol. 3, Oct. 2018. [Online]. Available: https://ledgerjournal.org/ojs/index.php/ledger/article/view/127

[21] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System.”

[22] “Wikipedia - blockchain,” Apr. 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Blockchain&oldid=892570721

[23] “Learn me a bitcoin - home page.” [Online]. Available: http://learnmeabitcoin.com/

[24] “Wikipedia - cryptographic hash function,” Apr. 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cryptographic_hash_function&oldid=892749881