Security of Blockchain Technologies

Master Thesis

Author(s):
Wüst, Karl

Publication date:
2016

Permanent link:
https://doi.org/10.3929/ethz-a-010703416

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Security of Blockchain Technologies

Master Thesis

Karl Wüst

2016-07-11

Supervisor: Prof. Dr. Srdjan Capkun
Advisor: Arthur Gervais

Department of Computer Science, ETH Zürich

Abstract

Proof of Work (PoW) powered blockchains currently account for more than 90% of the total market capitalization of existing digital currencies. Although the security of Bitcoin has been analysed extensively, most analyses use restricted models that do not take parameters such as network delays or block intervals into account and do not allow a comparison of different blockchain instances with different consensus parameters. In this thesis, we provide a quantitative framework based on Markov Decision Processes (MDP) to analyse the security of different PoW blockchain instances with various parameters against selfish mining and double spending attacks. We devise optimal adversarial strategies for double-spending and selfish mining that take into account real world parameters such as network delays, different block sizes and block generation intervals that we capture through the stale block rate and we consider the impact of eclipse attacks. Additionally, we perform a practical security analysis of Ethereum and Stellar where we describe vulnerabilities in Ethereum and discuss the security of the current state of the Stellar network.

Acknowledgements

I would like to thank my advisor Arthur Gervais for his guidance and for the valuable feedback and advice that he provided. I would also like to thank Professor Srdjan Capkun for the opportunity to write my thesis in his research group.

Contents

Chapter 1

Introduction

Since the creation of Bitcoin in 2009, interest in blockchain technology has increased continuously and has spiked innovation. While blockchains are mostly used for cryptocurrencies, other uses such as smart contracts have been developed. Bitcoin has been forked multiple times to change consensus parameters such as the block interval or the hash function used for the Proof of Work (PoW) algorithm with the goal of increasing the performance of the chain. Other blockchains such as Ethereum have included the possibility to execute smart contracts—applications that run decentralised in the network—and use modified PoW consensus mechanisms.

While there exist many different consensus protocols (e.g. PBFT [1], Proof of Stake [2], Proof of Elapsed Time [3]), most blockchains currently use the Proof of Work (PoW) consensus mechanism. For example, the three largest cryptocurrencies, Bitcoin, Ethereum, and Litecoin, which all use PoW, currently make up over 90% of the market capitalisation of all cryptocurrencies. While Proof of Work is computationally expensive, it has the advantage that it allows open participation to anyone and scales to a large number of participating nodes.

Although the security of Bitcoin has been analysed extensively [4, 5, 6, 7], most analyses use restricted models that do not take parameters such as network delays or block intervals into account and do not allow a comparison of different blockchain instances with different consensus parameters. In this thesis, we provide a quantitative framework based on Markov Decision Processes (MDP) to analyse the security of different PoW blockchain instances with various parameters against selfish mining and double spending attacks.

Parameters such as different block generation times, block sizes, and information gossip mechanisms impact the *stale block rate*, i.e. the rate of blocks that are created by the network but not incorporated into the main chain due

to a conflicting block. We use the stale block rate to summarise these parameters and quantify their impact on the security of a blockchain instance.

To this end, we use Markov Decision Process to find optimal adversarial strategies for selfish mining and double-spending attacks on confirmed transactions. Our analysis shows that a higher stale block rate results in more severe selfish mining attacks and increases the incentive for an adversary to execute double-spending attacks. In addition, we show that the value of a block reward of a blockchain has considerable impact on the resilience against double-spending. A higher block reward causes double-spending to be less rewarding for an adversary and thus increases the security of the system.

For instance, since Ethereum has much lower block rewards (in dollar value) and a higher stale block rate than Bitcoin (6.8% vs. 0.41%), a transaction in Ethereum needs at least 37 confirmations to match the security of 6 confirmations in Bitcoin (against an adversary with 30% of the mining power).

We additionally conduct a practical security analysis of two different blockchains, Ethereum (PoW based) and Stellar (a protocol for Federated Byzantine Agreement [8]). Our analysis led to the discovery of three vulnerabilities in Ethereum and show that current state of the stellar network is insecure and vulnerable to a split network that allows to easily double spend transactions.

The remainder of the thesis is organized as follows. In Chapter 2, we give an overview of the basic concepts behind PoW blockchains. In Chapter 3, we conduct our quantitative analysis of different PoW blockchains. In chapter 4, we describe our analysis of Ethereum and Stellar and describe the discovered vulnerabilities. We give an overview of related work in Chapter 5 and conclude the thesis in Chapter 6.

Chapter 2

Background on Proof of Work Blockchains

The goal of a blockchain is to eliminate the need of a central entity that ensures consistent data storage. To this end, a blockchain consists of blocks containing data entries that are chained together by referencing the previous block. In order to ensure that all entries are valid and consistent, the network is required to achieve *consensus* about the blocks in a decentralised fashion. The most prominently used consensus mechanism in blockchains is *Proof of Work*, but other mechanisms gain in popularity. The Proof of Work consensus mechanism is described below.

2.1 Consensus Layer

The Proof of Work (PoW) consensus mechanism was introduced by Bitcoin [9] and is currently the most widely used consensus mechanism in blockchains. PoW consists of creating blocks containing some data entries (e.g. financial transactions) and solving computationally hard (but easily verifiable) problems for every block. If a solution to the PoW puzzle is found, the block is broadcast to the network and is accepted by the other nodes if the solution is valid. PoW can be understood as voting with computing power. In general, the PoW consensus mechanism can be viewed as a practical solution to the Byzantine Generals Problem [10] without the requirement that participants are known beforehand, i.e. PoW allows free participation of any entity.

Most blockchain instances (e.g. Bitcoin, Ethereum) employ a hash based PoW algorithm, which consists of finding a nonce such that, when information about a block is hashed together with this nonce, the value of the hash is smaller than a target value. The block including this nonce is then forwarded on the network layer (cf. Section 2.2) to the other nodes, who can

then check the validity of the PoW easily.

In particular, given the set of transactions that have been announced since the last block's generation, and the hash of the last block, Bitcoin miners need to find a nonce such that:

$$\texttt{SHA256}\{\mathsf{Bl}_l \,||\, \mathrm{MR}(\mathrm{TR}_1, \ldots, \mathrm{TR}_n) \,||\, \mathrm{No}\} \leq \text{target}, \tag{2.1}$$

where Bl_l denotes the last generated block, $\mathrm{MR}(\overline{x})$ denotes the root of the Merkle tree with elements $\overline{x}$, $\mathrm{TR}_1 \,||\, \ldots \,||\, \mathrm{TR}_n$ is a set of transactions that have been chosen by the miners to be included in the block[1], No is the 32-bit nonce, and target is a 256-bit number. To generate the PoW, each miner performs brute-force search and it is apparent that the bigger target is, the easier it is to find a nonce that satisfies the PoW.

2.1.1 Security

In Proof of Work, we assume that the honest entitities in the network collectively supply more than 50% of the mining power. If an adversary controls more than 50%, he will be able to eventually sustain the longest chain and thus can trivially execute double spending attacks or censor the network. However, it has been shown that an adversary with less than 50% of the mining power can also execute attacks on the blockchain, and we briefly outline some known attacks below.

Accepting transactions without blockchain confirmations is insecure, since double spending such a transaction can be done at low cost [5]. The likelihood that a transaction is reversed in the future sinks as the number of confirmations grows.

While the honest mining protocol mandates that a newly found block is published immediately, an adversary does not need to adhere to that protocol and it has been shown in previous work that a *selfish miner* can increase his relative mining share by selectively withholding or publishing his mined blocks [7, 11].

If the network has tight synchronisation, selfish mining and double-spending attacks can be avoided. *Eclipse attacks* [12, 13], in which an adversary controls parts of the network communication and logically creates a partition of the network can increase synchronisation delays and thus weaken the security of the network. Eclipse attacks can take many different forms and range from simple denial of service (DoS) attacks to improved selfish mining and double-spending [13]. In an eclipse attack, the adversary selectively keeps some information hidden from one or more participants, e.g. an adversary

[1] These transactions are chosen from the transactions which have been announced (and not yet confirmed) since Bl_l's generation.

delays the delivery of a block to a node. An example of an eclipse attack on Ethereum is given in section 4.1.2.

PoW facilitates the construction of an *unpermissioned* blockchain in which the participants are not known a priori. This allows PoW blockchains to scale to a large number of nodes and enables decentralisation in the sense that the participants can join and leave at will. PoW-powered blockchains are, however, often criticised for wasting resources and for their limited performance. For the Bitcoin PoW, the cost for one confirmed transaction is estimated to be 6.2$ [14] and the throughput of transactions is about 3.3-7 transactions per second in Bitcoin [14], which is very low when compared to other payment networks such as Visa, which is capable of handling up to 56000 transactions per second [15]. We discuss alternative proposals for blockchain consensus in section 5.

2.1.2 Block interval

The block interval defines the average amount of time that passes between the creation of two blocks. With a shorter block interval, the latency with which a transaction is written to the blockchain gets shortened, i.e. a transaction is confirmed quicker. The block interval also directly correlates to the difficulty of the underlying PoW puzzle. The lower the difficulty, the faster blocks are created (all other things being equal).

In Bitcoin and most other PoW blockchains (e.g. Ethereum, Dogecoin, Litecoin), the difficulty is adjusted regularly in order to reach a target block interval. It is reasonable to assume that the block interval has an influence on the security of a blockchain instance and thus we include it in our model in Section 3.1. Note that a smaller block interval leads to a higher stale block rate since more conflicting blocks are found in the network. This increases the advantage of an adversary as it slows the growth of the valid chain relative to the block creation rate.

Besides their security implications, stale blocks also impact the performance of the system. Stale blocks cause additional overhead for validation and network propagation and in most blockchain instances, no block reward is awarded to miners of stale blocks.

2.2 Network Layer

The network layer is of interest for the security analysis of a blockchain instance since network delays have a direct impact on the stale block rate. The two main parameters on this layer are the block size and the information propagation mechanisms.

2.2.1 Block size

The throughput of a system is bounded by the maximum blocksize (given a fixed block interval), as the maximum number of included transactions is directly dependent on the block size. Larger blocks do however cause slower propagation speeds, which causes more stale blocks to be created. An unlimited blocksize could, for example, result in a DoS attack on the system by creating a block that takes a long time to validate.

2.2.2 Block propagation mechanism

In a blockchain network, all nodes are expected to receive all blocks. The block propagation mechanism that dictates how information spreads through the network thus also impacts both the performance and the security of the network. Below, we describe different network layer implementations.

Advertisement-based information dissemination: Bitcoin and Bitcoin forks use an advertisement based request management system. If a node A receives a block, A will advertise the block to its peers with an *inv* message. A node B will then request the block only if it has not received the block previously to which A replies by sending the block.

Send headers: *Send headers* is a modification of the basic advertisement-based protocol that skips the use of *inv* messages. Instead a peer sends a *sendheaders message* to directly receive block headers when they are available. This mechanisms effectively reduces the latency of block message propagation.

Unsolicited block push: The unsolicited block push allows the miner of a block to broadcast the block without advertising it first. Since a miner can be sure that no other node could have known the block beforehand, this is a sensible modification that reduces the latency and the bandwidth overhead of the block propagation.

Relay networks: Relay networks [16] improve the synchronisation of miners sharing a common transaction pool with a global ID for each transaction. Since the transaction IDs are smaller than the transactions, the transactions are replaced by their IDs when a miner broadcasts a block. This reduces the propagation delay due to the effectively smaller block size.

Hybrid Push/Advertisement Systems: A direct block push can be combined with an advertisement based system. For example, in Ethereum a node A pushes a block directly to $\sqrt{n}$ peers, where n is the total number of neighbors of A and advertises the block hash to all other directly connected peers.

	Bitcoin	**Litecoin**	**Dogecoin**	**Ethereum**
Block interval	10 min	2.5 min	1 min	10-20 seconds
t_{MBP}	8.7 s [14]	1.02 s	0.85 s	0.5 - 0.75 s [18]
Public nodes	6000	800	600	4000 [19]
r_s	0.41%	0.273%	0.619%	6.8%
s_B	534.8KB	6.11KB	8KB	1.5KB

Table 2.1: Comparison of different Bitcoin forks, Ethereum and the impact of parameter choices on the network propagation times. Stale block rate (r_s) and average block size (s_B) were measured over the last 10000 blocks. t_{MBP} is the median block propagation time.

2.3 Impact on the stale block rate

In Table 2.1, we summarise the block intervals, average block sizes, number of public nodes, propagation delays, and stale block rates of different blockchains from [17]. The data shows that the block interval and block sizes strongly influence the stale block rate. For instance, due to a larger number of nodes and a much higher transaction load, the stale block rate of Bitcoin is higher than that of Litecoin despite the former having a much longer block interval. The block sizes of Dogecoin and Litecoin are however similar, which shows the impact of the block interval on the stale block rate. Ethereum, with the shortest block interval, has the highest stale block rate. Note that Table 2.1 shows the uncle rate, the actual stale block rate may be slightly higher.

Chapter 3

Quantifying the Security of Proof of Work Powered Blockchains

In this chapter, we model selfish mining and double-spending as single-player decision problems to devise optimal adversarial strategies. Based on these optimal strategies we are able to quantify the security of different Proof of Work blockchains while taking into account a number of different real world parameters such as network delays, different block sizes and block generation intervals. We capture these parameters in our model through the stale block rate. Parts of [17] are based on this chapter.

3.1 Blockchain Model

In this section, we introduce our blockchain model and use it to quantify the optimal adversarial strategies for selfish mining and double-spending. These strategies aim to increase the advantage of the adversary, either financially or in terms of the relative mining power. Based on our model, we compare the security provisions of PoW-based blockchains when instantiated with different parameters.

To this end, we extend the selfish mining model of Sapirshtein *et al.*'s [11] by *(i)* introducing the stale block rate to account for network delays and different block sizes, *(ii)* considering mining costs of the adversary for double spending, and *(iii)* capturing full eclipse attacks against a single victim.

In our model, we define the following parameters:

- The adversarial mining power α captures the computing power of the adversary.
- ω captures the mining power of the eclipsed victim.
- $1 - \alpha - \omega$ corresponds to the mining power of the honest network.

- The stale block rate r_s allows us to model the network delay resulting from the network propagation and the block sizes.
- The propagation parameter γ captures the connectivity of the adversary within the network, i.e. it captures the fraction of the network that receives the adversary's blocks first when the adversary and an honest miner release blocks simultaneously in the network.

To analyse the optimal double-spending strategy, we also define the double-spending amount v_d, and the number of required transaction confirmations k (i.e. the number of blocks that the transactions needs to be accepted by a merchant). The adversarial mining costs $c_m \in [0, \alpha]$ correspond to the expected mining costs of the adversary (i.e., total mining costs for hardware, electricity and man-power) and are expressed in terms of block rewards. For example, if $c_m = \alpha$, the mining costs of the adversary are equivalent to his share of the mining power times the block reward, i.e., the mining costs are covered exactly by the earned block revenue in honest mining. To account for eclipse attacks, we assume an eclipsed victim with mining power ω and the adversarial chain of length l_a contains b_e blocks which were mined by the victim.

We model the blockchain from the view of the adversary using a single-player decision problem $M := \langle S, A, P, R \rangle$ where S corresponds to the state space, A to the action space, P to the transition matrix, and R to the reward matrix. In our model, we assume that while the adversary may deviate from the standard protocol, all other participants are honest and follow the standard protocol.

A state $s \in S$ is defined as a tuple of the form $(l_a, l_h, b_e, fork)$, where l_a and l_h represent the length of the adversarial and honest chain respectively, b_e is the number of blocks mined by the eclipsed victim, and $fork$ can take the three values *irrelevant*, *relevant* and *active*:

relevant The label *relevant* signifies that (i) the last block has been found by the honest network, and (ii) if $l_a \geq l_h$ the *match* action is applicable. A state of the form $(l_a, l_h - 1, b_e, \cdot)$ for instance results in $(l_a, l_h, b_e, relevant)$.

irrelevant When the last block was found by the adversary, the previous block has likely already reached the majority of the nodes in the network. The adversary is therefore not able to perform a *match* action. A state of the form $(l_a - 1, l_h, b_e, \cdot)$ for instance results in $(l_a, l_h, b_e, irrelevant)$.

active The state contains the label *active*, if the adversary performed a *match* action, i.e. the network is currently split and in process of determining the longest chain.

The action space consists of the following actions available to the adversary:

Adopt The adversary accepts the chain of the honest network, which effectively restarts the attack.

Override The adversary publishes one block more than the honest chain contains and consequently overrides conflicting blocks.

Match The adversary publishes as many blocks as the honest chain contains, and triggers an adoption race between the two chains.

Wait The adversary continues mining on his hidden chain until a block is found.

Exit The *exit* action corresponds to a successful double-spend with k confirmations and is only feasible if $l_a > l_h$ and $l_a > k$. This action is only relevant for double-spending, and does not apply to selfish mining.

All of the available actions, except the *exit* action, correspond to the creation of one block in the network (either by the adversary or an honest actor).

With the *adopt* action, the adversary accepts the honest chain and restarts his attack. The action is appropriate when the adversarial chain is too far behind the honest chain to catch up with a high enough likelihood.

If the secret chain of the adversary is longer than the main chain (i.e. $l_a > l_h$), the adversary can publish $l_h + 1$ of his blocks with the *override* action. Since this corresponds to a fork where the adversary's released blocks are part of the longest chain, the network will accept the chain of the adversary and the honest chain is replaced.

Instead of performing an *override*, the adversary might alternatively choose the *match* action to trigger a race between his and the honest network's chain. For the *match* action, the adversary is required to have a chain that is at least as long as the main chain (i.e. $l_a \geq l_h$). Additionally, the last block must have been mined by the honest network, since otherwise the last honest block has already been propagated to all nodes. The success of the *match* action is highly dependent on the communication capabilities of the adversary in the network, i.e. the fraction γ of peers the adversary can reach with his block, before they are reached by the newly mined block of the honest network.

With the *wait* action, the adversary simply continues mining on his private chain without publishing a block.

The *exit* action corresponds to a successfull double spend with k confirmations and is applicable if $l_a > l_h$ and $l_a > k$. After the *exit* action, the adversary will mine honestly.

Given the adversarial mining power α, the initial state $(0, 0, 0, \mathit{irrelevant})$ transitions to $(1, 0, 0, \mathit{irrelevant})$ with probability α, corresponding to the case where the adversary finds the next block. If the honest network finds a non-stale block, the resulting state is $(0, 1, 0, \mathit{relevant})$. If the honest network's

block results in a stale block, however, the state remains $(0, 0, 0, \mathit{irrelevant})$ since a stale block does not count towards the longest chain. The last case accounts for the eclipsed victim which finds a block with probability ω, resulting in state $(1, 0, 1, \mathit{irrelevant})$.

By capturing the stale block rate r_s, our model allows us to indirectly capture a number of fundamentally important real world parameters of the blockchain, such as the block size, block propagation time, number of network nodes, number of node connections, and the block propagation protocol. In our model, we assume that the miners of the honest network are affected by the stale block rate, while the adversary and the colluding eclipsed victims do not mine stale blocks. This is due to the fact that the adversary can use any mined blocks for an attack and effectively only has a small chance of mining a stale block after the adopt action when mining honestly. The adversary therefore exhibits a significantly lower real stale block rate than the honest network in practice. The blocks found by the eclipsed victim also count towards the private chain of the adversary which is explained in the following. As we show in Sections 3.2 and 3.3, our model allows us to find the optimal strategies for selfish mining and double-spending.

The state transitions are summarised in Table 3.1. The reward for each transition consists of a tuple, where the first element is the reward for the adversary and the second element is the reward for the honest network.

3.1.1 Selfish Mining vs. Double-spending

We consider double-spending and selfish mining independently here, since selfish mining and double-spending have different adversarial goals. Selfish mining is not always a rational strategy: the objective of selfish mining is to increase the relative share of the adversarial blocks committed to the main chain, while in double-spending the adversary aims to maximize his absolute revenue.

Namely, as long as the difficulty of a PoW blockchain does not change (e.g. Bitcoin's difficulty changes only once every two weeks), selfish mining yields fewer block rewards than honest mining. In honest mining, the adversary is rewarded for every mined block, while he will lose any previously mined blocks when adopting the main chain in selfish mining. Since the adversary has less mining power than the honest network, he has a high probability of falling behind the main chain, causing him to adopt the main chain when he has no significant chance of catching up—which in turn leads to lost block rewards. For instance, following our optimal selfish mining strategy (cf. Section 3.2), an adversary with 30% of the mining power earns on average 209 block rewards (if he wins every block race) in a duration where 1000 blocks are mined by the whole network (as opposed to 300 for honest mining). Similarly, Eyal and Sirer's [7] strategy yields 205.80 blocks

State × Action	Resulting State	Probability	Reward (in Block reward)
$(l_a, l_h, b_e, \cdot), adopt$	$(1,0,0,i)$ $(1,0,1,i)$ $(0,1,0,r)$ $(0,0,0,i)$	α ω $(1-\alpha-\omega)\cdot(1-r_s)$ $(1-\alpha-\omega)\cdot r_s$	$(-c_m, l_h)$ $(-c_m, l_h)$ $(-c_m, l_h)$ $(-c_m, l_h)$
$(l_a, l_h, b_e, \cdot), override$	$\left(l_a - l_h, 0, b_e - \lceil (l_h+1)\frac{b_e}{l_a} \rceil, i\right)$ $\left(l_a - l_h, 0, b_e - \lceil (l_h+1)\frac{b_e}{l_a} \rceil + 1, i\right)$ $\left(l_a - l_h - 1, 1, b_e - \lceil (l_h+1)\frac{b_e}{l_a} \rceil, r\right)$ $\left(l_a - l_h - 1, 0, b_e - \lceil (l_h+1)\frac{b_e}{l_a} \rceil, i\right)$	α ω $(1-\alpha-\omega)\cdot(1-r_s)$ $(1-\alpha-\omega)\cdot r_s$	$\left(\lfloor (l_h+1)\frac{l_a-b_e}{l_a} \rfloor - c_m, 0\right)$ $\left(\lfloor (l_h+1)\frac{l_a-b_e}{l_a} \rfloor - c_m, 0\right)$ $\left(\lfloor (l_h+1)\frac{l_a-b_e}{l_a} \rfloor - c_m, 0\right)$ $\left(\lfloor (l_h+1)\frac{l_a-b_e}{l_a} \rfloor - c_m, 0\right)$
$(l_a, l_h, b_e, i), wait$ $(l_a, l_h, b_e, r), wait$	(l_a+1, l_h, b_e, i) (l_a+1, l_h, b_e+1, i) (l_a, l_h+1, b_e, r) (l_a, l_h, b_e, i)	α ω $(1-\alpha-\omega)\cdot(1-r_s)$ $(1-\alpha-\omega)\cdot r_s$	$(-c_m, 0)$ $(-c_m, 0)$ $(-c_m, 0)$ $(-c_m, 0)$
$(l_a, l_h, b_e, a), wait$ $(l_a, l_h, b_e, r), match$	(l_a+1, l_h, b_e, a) (l_a+1, l_h, b_e+1, a) $(l_a - l_h, 1, b_e - \lceil (l_h)\frac{b_e}{l_a} \rceil, r)$ (l_a, l_h+1, b_e, r) (l_a, l_h, b_e, a)	α ω $\gamma\cdot(1-\alpha-\omega)\cdot(1-r_s)$ $(1-\gamma)\cdot(1-\alpha-\omega)\cdot(1-r_s)$ $(1-\alpha-\omega)\cdot r_s$	$(-c_m, 0)$ $(-c_m, 0)$ $\left(\lfloor (l_h)\frac{l_a-b_e}{l_a} \rfloor - c_m, 0\right)$ $(-c_m, 0)$ $(-c_m, 0)$
$(l_a, l_h, b_e, \cdot), exit$	$exit$	1	$(l_a - b_e + v_d, 0)$

Table 3.1: State transition and reward matrices for optimal selfish mining and double-spending strategies in PoW blockchains. α is the mining power of the attacker, ω is the mining power of the eclipsed node, b_e is the number of blocks in the attacker chain that were mined by the eclipsed node, γ is the fraction of nodes that an attacker can reach faster than the honest network, r_s is the stale block rate and v_d is the value of the double-spend. The actions *override* and *match* are feasible only when $l_a > l_h$ or $l_a \geq l_h$, respectively. We discount the mining costs $c_m \in [0, \alpha]$ in the state transition reward only for double-spending. The fork label (last element of the state) is denoted by i, r and a for *irrelevant*, *relevant* and *active* respectively. The first index of the reward tuple corresponds to the adversary, while the second stands for the honest network for selfish mining.

rewards. A rational adversary would thus choose honest mining when mining within a time frame with constant difficulty.

3.1.2 Eclipse attacks

In an eclipse attack, a fraction ω of the overall mining power is eclipsed [12, 20] from receiving information from the honest network, i.e. the adversary controls a significant part of the communication between the victim and the rest of the network. The adversary has the following possibilities to use a fully eclipsed victim for selfish mining or a double-spending attack:

No eclipse attack This case is captured in our model if $\omega = 0$.

Isolate the victim This is captured implicitly in our model. Namely, this corresponds to a decrease of the total mining power and thus an in-

crease of the attacker mining power to $\alpha' = \frac{\alpha}{1-\omega}$.

Exploit the eclipsed victim Here, the adversary exploits the victim's mining power ω and uses it to advance his private chain. This is the most likely choice of a rational adversary when performing double-spending attacks. In this case, we assume that the victim is fully eclipsed from the network and does not receive/send blocks unless permitted by the adversary [12, 20].

Other variants to exploit the eclipsed victim are possible and the adversary may only be able to conduct partial eclipse attacks (cf. Section 4.1.2 and [13]). Due to the large number of different possibilities, we consider only full eclipse attacks, i.e. the adversary fully controls the flow of information between the network and the victim.

3.2 Optimal Strategies for Selfish Mining

As mentioned above, the objective for the adversary in selfish mining is not to optimise the absolute reward, but to increase his share of blocks that are included in the chain accepted by the network. We capture this by optimising the relative revenue r_{rel} defined by Equation 3.1, where r_{a_i} and r_{h_i} are the rewards in step i for the adversary and the honest network, respectively.

$$r_{rel} = \mathbb{E}\left[\varliminf_{n\to\infty} \frac{\sum_{i=1}^{n} r_{a_i}}{\sum_{i=1}^{n}(r_{a_i} + r_{h_i})}\right] \tag{3.1}$$

Since the objective function, i.e. the relative reward, in selfish mining is non-linear and thus the single-player decision problem cannot be modelled directly as an MDP, we adapt the technique of Sapirshtein *et al.*'s [11] which we describe below. From the single-player decision problem defined by Table 3.1 and Equation 3.1, we construct a family of Markov Decision Processes (MDPs) by applying a transformation function on the reward of the decision problem. Namely, we assume that the value of the objective function (i.e. the optimal relative reward) is ρ and define for any $\rho \in [0, 1]$ the transformation function $w_\rho : \mathbb{N}^2 \to \mathbb{R}$ with the adversarial reward r_a and the reward of the honest network r_h in Equation 3.2.

$$w_\rho(r_a, r_h) = (1 - \rho) \cdot r_a - \rho \cdot r_h \tag{3.2}$$

This results in an infinite state MDP $M_\rho = \langle S, A, P, w_\rho(R) \rangle$ for each ρ that has the same action and state space as the original decision problem and the same transition matrix but where the reward matrix is transformed using w_{rho}. The expected value of such an MDP under policy π is then defined by v_ρ^π in Equation 3.3, where $r_i(\pi)$ is the reward tuple in step i under policy π.

$$v_\rho^\pi = \mathbb{E}\left[\underline{\lim}_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} w_\rho(r_i(\pi))\right] \tag{3.3}$$

The expected value under the optimal policy is given by

$$v_\rho^* = \max_{\pi \in A} \left\{ v_\rho^\pi \right\} \tag{3.4}$$

We base our method to optimise r_{rel} on the following propositions from [11]:

1. If $v_\rho^* = 0$ for some $\rho \in [0, 1]$, then an optimal policy π^* in the transformed MDP M_ρ also maximises r_{rel} and $r_{rel} = \rho$.

2. v_ρ^* is monotonically decreasing in ρ.

Since standard MDP solvers are not able to solve infite state MDPs, we restrict the state space of our family of MDPs by only allowing either chain to be of length at most c, resulting in a finite state MDP M_ρ^c. If either chain reaches length c, the adversary is only allowed to perform the *override* or *adopt* action. This gives a lower bound for the optimal value of the infinite state MDP.

Intuitively, we can reason about the correctness of the first proposition as follows for the bounded single-player decision problem. In a recurring finite state MDP, the initial state will be visited again in expectation after some finite number of steps S. During that time, the adversary gains an expected reward of $R_a = \mathbb{E}\left[\sum_{s=1}^{S} r_{a_i}\right]$ and the honest network gains a reward of $R_h = \mathbb{E}\left[\sum_{s=1}^{S} r_{h_i}\right]$ in the original (bounded) decision problem. It follows that the expected reward per step in the Markov Chain is $\overline{r_a} = \mathbb{E}\left[\frac{1}{S}\sum_{s=1}^{S} r_{a_i}\right]$ and $\overline{r_h} = \mathbb{E}\left[\frac{1}{S}\sum_{s=1}^{S} r_{h_i}\right]$ for the adversary and the honest network, respectively. We can thus simplify the expected relative revenue r_{rel} to:

$$
\begin{aligned}
r_{rel} &= \mathbb{E}\left[\varliminf_{n\to\infty} \frac{\sum_{i=1}^{n} r_{a_i}}{\sum_{i=1}^{n}(r_{a_i}+r_{h_i})}\right] && (3.5)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \frac{n\cdot\overline{r_a}}{n\cdot(\overline{r_a}+\overline{r_h})}\right] && (3.6)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \frac{\overline{r_a}}{(\overline{r_a}+\overline{r_h})}\right] && (3.7)\\
&= \mathbb{E}\left[\frac{\overline{r_a}}{\overline{r_a}+\overline{r_h}}\right] && (3.8)\\
&= \frac{\overline{r_a}}{\overline{r_a}+\overline{r_h}} && (3.9)\\
& && (3.10)
\end{aligned}
$$

We also have:

$$
\begin{aligned}
v_\rho^* &= v_\rho^{\pi^*} && (3.11)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} w_\rho(r_i(\pi^*))\right] && (3.12)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n}((1-\rho)\cdot r_{a_i}-\rho\cdot r_{h_i})\right] && (3.13)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \frac{1}{n}\left((1-\rho)\cdot\sum_{i=1}^{n} r_{a_i}-\rho\cdot\sum_{i=1}^{n} r_{h_i}\right)\right] && (3.14)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \frac{1}{n}\left(n\cdot(1-\rho)\cdot\overline{r_a}-n\cdot\rho\cdot\overline{r_h}\right)\right] && (3.15)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \left((1-\rho)\cdot\overline{r_a}-\rho\cdot\overline{r_h}\right)\right] && (3.16)\\
&= \mathbb{E}\left[(1-\rho)\cdot\overline{r_a}-\rho\cdot\overline{r_h}\right] && (3.17)\\
&= (1-\rho)\cdot\overline{r_a}-\rho\cdot\overline{r_h} && (3.18)
\end{aligned}
$$

And thus, for the case where $\rho = r_{rel} = \frac{\overline{r_a}}{\overline{r_a}+\overline{r_h}}$:

$$v_\rho^* = (1-\rho)\cdot\overline{r_a} - \rho\cdot\overline{r_h} \tag{3.19}$$

$$= (1-\frac{\overline{r_a}}{\overline{r_a}+\overline{r_h}})\cdot\overline{r_a} - \frac{\overline{r_a}}{\overline{r_a}+\overline{r_h}}\cdot\overline{r_h} \tag{3.20}$$

$$= \frac{\overline{r_a}\cdot(\overline{r_a}+\overline{r_h}) - \overline{r_a}^2}{\overline{r_a}+\overline{r_h}} - \frac{\overline{r_a}\cdot\overline{r_h}}{\overline{r_a}+\overline{r_h}} \tag{3.21}$$

$$= \frac{\overline{r_a}^2 + \overline{r_a}\cdot\overline{r_h} - \overline{r_a}^2 - \overline{r_a}\cdot\overline{r_h}}{\overline{r_a}+\overline{r_h}} \tag{3.22}$$

$$= 0 \tag{3.23}$$

The reasoning for the second proposition is straight forward. For any given policy π, it holds for $\rho > \rho'$ that $w_\rho(r_a, r_h) \leq w_{\rho'}(r_a, r_h)$ for every transition with rewards r_a and r_h for the adversary and the honest network respectively. It follows directly that $v_\rho^\pi \leq v_{\rho'}^\pi$ for every policy π and thus $v_\rho^* \leq v_{\rho'}^*$, i.e. v_ρ^* is monotonically decreasing in ρ.

We use binary search on our restricted family of MDPs for $\rho \in [0,1]$ in order to find the ρ for which the expected value in the instantiated MDP is zero and which thus maximizes the reward in the original single-player decision problem [11]. Since v_ρ^* is monotonically decreasing, this can be done efficiently as follows:

```
function OPTIMAL STRATEGY(c, ε)
    low ← 0
    high ← 1
    repeat
        ρ ← (low + high)/2
        (π, v*_ρ) ← MDP_SOLVER(M^c_ρ)
        if v*_ρ > 0 then
            low ← ρ
        else
            high ← ρ
        end if
    until high − low < ε
    return π, ρ
end function
```

In contrast to previous models, our model captures various parameters such as block propagation times, block size, block generation interval, and known network vulnerabilities such as eclipse attacks. Since the objective in selfish

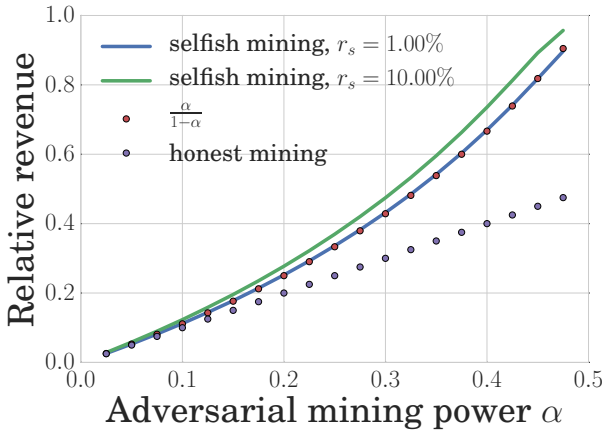


Figure 3.1: Selfish mining for r_s of 1%, 10%.

mining is to increase the relative mining share and not to optimise the absolute monetary reward, we do not consider mining costs in the model for selfish mining.

In order to solve the constructed MDPs, we apply a standard MDP solver for finite state space MDPs [21], and use a cutoff value of 30 blocks, i.e. we allow the length of the adversarial or the honest chain to be at most 30 blocks.

3.2.1 Impact of the stale block rate

We first analyse the impact of the stale block rate on selfish mining. We compare selfish mining with stale block rates of 1% and 10% in Figure 3.1 and observe that a higher stale block rate increases the relative revenue of the adversary significantly compared to a lower stale block rate (up to a maximum difference of 0.074). For comparison, we also plot $\frac{\alpha}{1-\alpha}$, which is an upper bound for selfish mining if the stale block rate is not considered, corresponding to the case where every block of the adversary is used to override one block generated by the honest network (as reported by Sapirshtein *et al.* [11]). Note that this bound is exceeded when taking into account the parameters captured via the stale block rate.

In Figure 3.2, we plot the relative revenue of an adversary with 10% or 30% as a function of the stale block rate, where we observe a non-linear relationship. This shows that the resilience against selfish mining attacks is highly influenced by the stale block rate of a blockchain instance.

3.2.2 Impact of Eclipse Attacks

In Figure 3.3, we investigate the impact of eclipse attacks on selfish mining. Due to the variety of shapes that an eclipse attack can take, we only con-

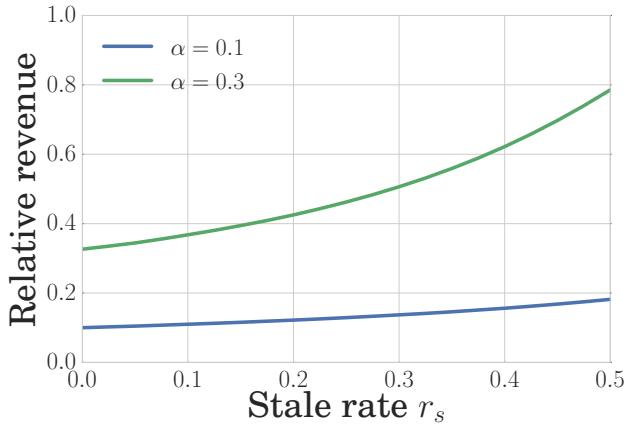


Figure 3.2: Selfish mining for $\alpha = 0.1$ and 0.3.

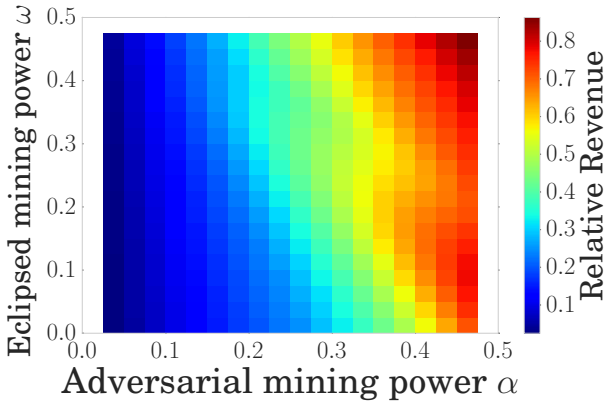


Figure 3.3: Selfish mining with eclipse attacks.

sider the simple case where the adversary fully controls the communication between the eclipsed victim and the rest of the network. Additionally, we assume that the adversary exploits the victims mining power and uses the blocks of his victim to advance his private chain. Note that the case where no eclipse attack occurs or the blocks of the victim are simply ignored are also captured in our model as explained in Section 3.1.2. However, we do not consider combinations of exploitative strategies and only determine the optimal adversarial policy given these restraints.

Overall, we observe that the higher the eclipsed mining power ω, the higher the relative reward of the adversary. However, for some values of ω and α (e.g., $\omega = 0.3$, $\alpha = 0.38$), it is more rewarding for the adversary not to include all blocks of the victim in his private chain. This is due to the fact

that the blocks mined by the victim count towards the reward of the honest network, thus reducing the relative revenue of the adversary.

3.3 Optimal Strategies for Double-Spending

As described in Section 3.1, we consider a different adversarial model for double-spending than for selfish mining, since selfish mining is not necessary financially rewarding when compared to honest mining. For double-spending we assume an economically rational adversary interested in maximising his financial gains.

In our model, we implicitly require that the adversary publishes a transaction T_l in the network while including a conflicting transaction T_d in his private chain each time the adversary starts a double-spending attempt (e.g. after an *adopt* action). Since the adversary receives a good or service in exchange for the transaction T_l (e.g. he receives an equivalent amount of some other currency), we assume that the operational costs of an unsuccessful attempt are small.

In contrast to selfish mining, where the objective function of the single-player decision problem is non-linear, we can model the decision problem for double-spending directly as Markov Decision Process (MDP). The MDP for double-spending also features an *exit* state in addition to the states in the selfish mining model (cf. Table 3.1). The *exit* state corresponds to a successful double spend and can only be reached if the adversarial chain is at least one block ahead of the honest chain ($l_a > l_h$) after k confirmations ($l_a > k$). After reaching the *exit* state, the adversary will conduct honest mining. Before the adversary reaches the *exit* state, he adopts an optimal strategy to maximise his reward, given the action and state spaces described in Section 3.1. Since we assume a rational adversary, the optimal strategy may not include any double-spending attempts depending on the value of the attempted double spend, i.e. the *exit* state is unreachable in the optimal strategy. In a transition to the *exit* state, the adversary earns a block reward of $l_a - b_e + v_d$, after an override action he earns $\lfloor (l_h + 1)\frac{l_a - b_e}{l_a} \rfloor - c_m$ block rewards and $\lfloor (l_h)\frac{l_a - b_e}{l_a} \rfloor - c_m$ block rewards if the adversary's chain wins the race after a *match* action. Note that the b_e blocks mined by the eclipsed victim need to be discounted as they are not part of the adversary's income. Since every state transition corresponds to one mined block (except *exit*), we discount the mining costs c_m in every state transition.

Depending on the parameters P (cf. Equation 3.24) and the double-spending value v'_d, the adversary will either use the optimal double-spending strategy π or he will perform honest mining, depending on the expected reward R. We are now interested in the smallest value v_d, such that the reward gained

by the adversary by double-spending v_d is larger than the reward gained through honest mining (cf. Equation 3.25).

$$P = (\alpha, \gamma, r_s, k, \omega, c_m) \tag{3.24}$$

$$v_d = \min\{v'_d | \exists \pi \in A : R(\pi, P, v'_d) > R(\text{honest mining}, P)\} \tag{3.25}$$

We argue that v_d emerges as a robust metric to quantify security under double-spending attacks. Namely, merchants can safely accept a transaction of a value smaller than v_d since a rational adversary will not attempt to double spend such a transaction. However, if dishonest behaviour is financially more rewarding given the value of the transaction, merchants should choose to require more confirmations or accept the associated double-spending risk. Moreover, v_d can be used as a metric to compare the resilience against double-spending attacks of different blockchain instantiations. Namely, if v_d of a blockchain instance A is bigger than for blockchain B for given α, γ and ω, then blockchain A can be considered more resistant against double-spending attacks.

In the following, we analyse and evaluate the optimal strategies for double-spending given different parameters. We rely on the *pymdptoolbox* library[1] to solve for the optimal strategy in our MDP and apply the *PolicyIteration* algorithm [21] with a discount value of 0.999.

In order to determine the minimum value v_d for which a double-spending attack is economically rational for an adversary, we instantiate the double-spending MDP with a high double-spending value ($> 10^9$ block rewards) such that the *exit* state is reachable given the optimal strategy.

If the *exit* state is reachable, there exists a double-spending strategy that yields a higher expected reward than honest mining. Otherwise, honest mining is more financially rewarding and will thus be the preferred strategy of a rational adversary. To find the lowest double-spending value v_d, we apply binary search on v_d (within an error margin of 0.1 block rewards), given α, k, r_s, γ and c_m.

This methodology also allows us to assess whether the number of transaction confirmations k are sufficient to ensure security against double-spending in the presence of a rational adversary, with respect to the considered transaction value. A transaction cannot be considered secure given k confirmations if the adversary has a higher expected financial gain in double-spending than honest mining and a merchant should wait for additional confirmations or accept the corresponding risk.

[1] `https://github.com/sawcordwell/pymdptoolbox`

	l_h								
l_a	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	w**	*a*	***	***	***	***	***	***	***
1	w**	ww*	ww*	*a*	***	***	***	***	***
2	w**	ww*	ww*	ww*	ww*	*a*	***	***	***
3	w**	ww*	ww*	ww*	ww*	ww*	*a*	***	***
4	w**	ww*	ww*	ww*	ww*	ww*	ww*	*a*	***
5	w**	ww*	ww*	ww*	ww*	ww*	ww*	ww*	*a*
6	w**	ww*	ww*	ww*	ww*	ww*	ww*	ww*	ww*
7	e**	e**	e**	e**	e**	e**	e**	w**	ww*
8	***	***	***	***	***	***	***	e**	w**

Table 3.2: Optimal double-spending strategy for $\alpha = 0.3, \gamma = 0, r_s = 0.41\%, c_m = \alpha, \omega = 0$ and $v_d = 19.5$. The rows correspond to the length l_a of the adversary's chain and the columns correspond to the length l_h of the honest network's chain. The three values in each table entry correspond to the fork labels *irrelevant*, *relevant* and *active*, where * marks an unreachable state and *w*, *a* and *e* denote the *wait*, *adopt* and *exit* actions, respectively.

In Table 3.2, we show as an example the optimal strategy for the case where $\alpha = 0.3$ (adversarial mining power), $\gamma = 0$ (propagation parameter), $c_m = \alpha$ (maximum mining costs) and $\omega = 0$ (no eclipse attack). For these parameters, we observe only *wait*, *adopt* and *exit* actions. As mentioned above, the MDP is resctricted to a length of 20 blocks for the adversarial and the honest chain, since we can only solve finite MDPs. In the following, we discuss the impact of the different parameters in greater details and the implications on the security of transaction confirmations.

The resulting Markov Chain that corresponds to an optimal double-spending strategy is absorbing in the *exit* state. We can thus use the fundamental matrix of the Markov Chain to calculate the expected number of steps before being absorbed by the *exit* state for every state [22]. Since every state transition corresponds to the creation of a block, the number of steps is equal to the expected number of blocks required to execute a successful double spend.

The expected number of blocks until a double spend is successful is plotted in Figure 3.4 with respect to the adversarial mining power and the number of transaction confirmations k. We observe that an adversary with a mining power of more than 25% of the total mining power is expected to need fewer than 1000 blocks for a successful double-spending attack (when fewer than

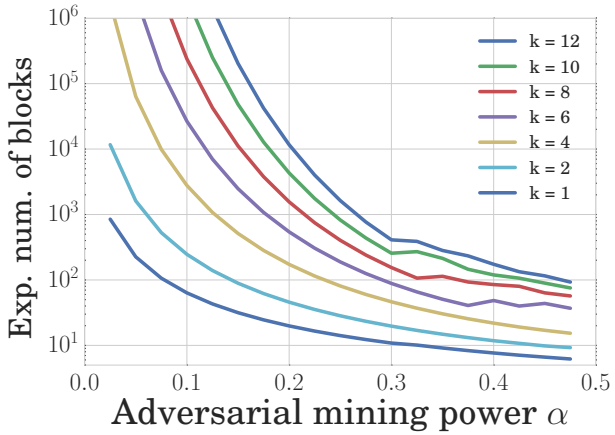


Figure 3.4: Expected number of blocks for successful double-spending given $r_s = 0.41\%, \gamma = 0, c_m = \alpha$ and $\omega = 0$.

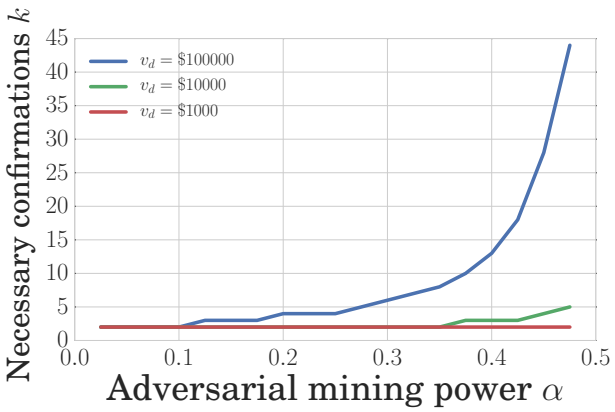


Figure 3.5: The number of required confirmations in Bitcoin for different transaction values

$k = 10$ confirmations are required), which corresponds to an attack duration of one week in Bitcoin.

In Figure 3.5, we show the number of required confirmations for different transaction values, such that double-spending is no longer profitable for an adversary. Our results show that the proposed 6 confirmations offer enough security against strong adversaries for transaction values up to USD 10000, but that more confirmations are required for much higher transaction values.

3.3.1 Impact of the propagation parameter

Recall that the propagation parameter γ specifies the connection capabilities of the adversary. In Figure 3.6, we depict the minimum double-spending transaction value for which the expected financial gain is higher with adversarial behaviour than with honest mining (cf. Equation 3.25) for $\gamma \in \{0, 0.5, 1\}$.

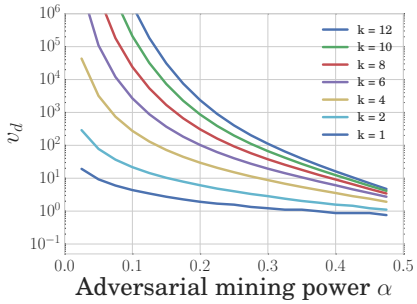
The adversary clearly has a higher advantage and the required transaction value of a double spend transaction is lower when γ is higher. For example, if $\gamma = 1$, double-spending with $k = 6$ confirmations is profitable for an adversary with 30% of the total mining power if the transaction has a value of at least 0.5 block rewards (one block reward in Bitcoin is currently 25 BTC, where 1 BTC = 436.7 USD). In contrast, if $\gamma = 0.5$, the value of the double-spend transactian has to be at least 12.9 block rewards.

3.3.2 Impact of the mining costs

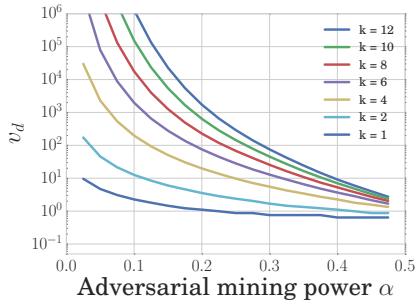
In Figure 3.7, we plot the required double-spending value for mining costs $c_m = 0$ and $c_m = \alpha$, and their difference Δ_{v_d}. The impact of the mining costs on the minimum required double-spending transaction value is negligible as our results show.

3.3.3 Impact of the stale block rate

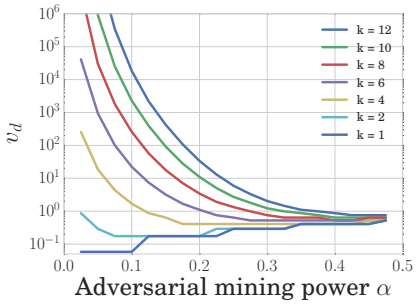
In Figure 3.8, we plot the impact of the stale block rate on the required double spending value for adversaries with $\alpha = 0.1$ and $\alpha = 0.3$. We observe that the relationship between the stale block rate and the required double spending value is non-linear and the stale rate has a very high impact on v_d. The higher the stale block rate, the worse is the resilience against double spending and selfish mining (cf. Figure 3.2). This shows that reducing the stale block rate is one of the main factors in improving the security of PoW blockchains.



(a) $\gamma = 0$



(b) $\gamma = 0.5$



(c) $\gamma = 1$

Figure 3.6: Impact of the propagation parameter γ. We observe that the required double-spending value for double-spending to be more profitable than honest mining sinks as γ increases. Other parameters are $r_s = 0.41\%$ (Bitcoin's stale block rate), $c_m = \alpha$ (maximum mining costs), $\omega = 0$ (no eclipse attack).

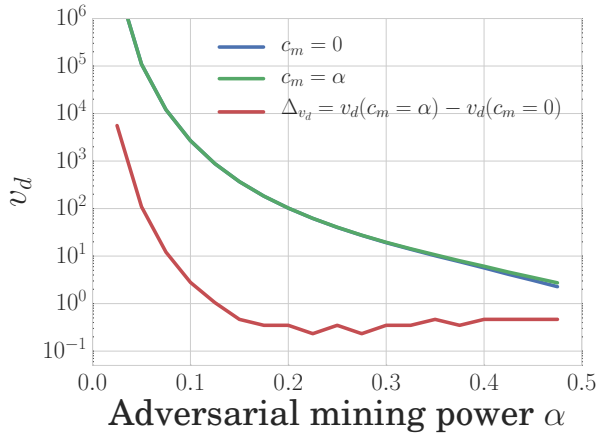
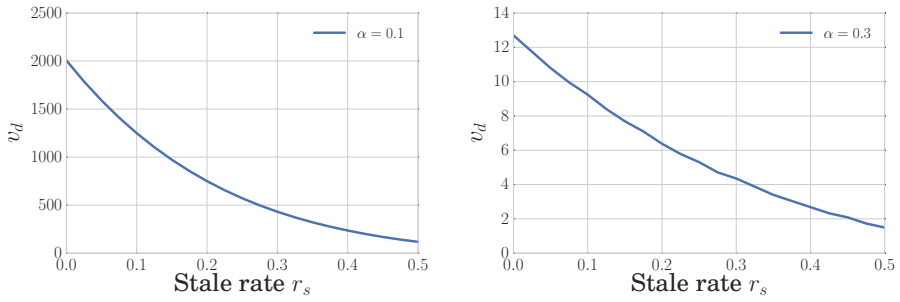


Figure 3.7: Impact of the mining cost c_m on the security of double spending ($r_s = 0.41\%$, $\gamma = 0$, $\omega = 0$) is relatively low.



(a) $\alpha = 0.1$ (b) $\alpha = 0.3$

Figure 3.8: Impact of stale block rate r_s on the security of double-spending given $\gamma = 0.5$, $\omega = 0$ for $\alpha = 0.1$ and $\alpha = 0.3$.

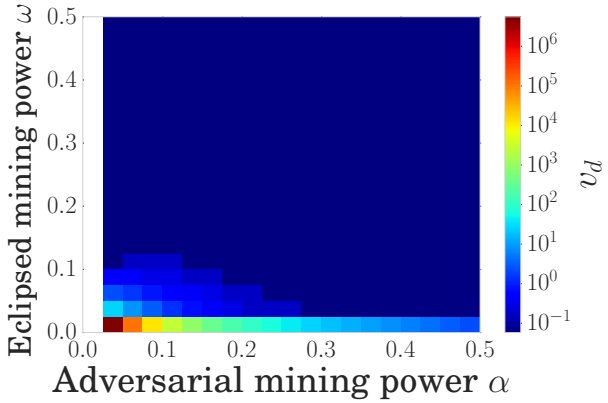


Figure 3.9: Full eclipse attack for $r_s = 0.41\%, \gamma = 0, c_m = 0$.

3.3.4 Impact of eclipse attacks

The ability to perform an eclipse attack clearly empowers the adversary as we show in Figure 3.9. Here, the adversary performs a full eclipse attack on the victim with mining power ω and exploits the victims mining power to advance his private chain. Since the blocks of the victim can be hidden without the risk of lost block rewards, the required double spending value is very low even for low values of α and ω.

3.3.5 Bitcoin vs. Ethereum

In this section, we compare the resilience against double spending of the two most prominent blockchains, Bitcoin and Ethereum. While they both use PoW, Ethereum is not a fork of Bitcoin and its consensus mechanism

is slightly different. While it is widely claimed[2] and even stated in the Ethereum Yellow Paper [23][3] that Ethereum uses a variant of GHOST [6], this is not actually the case. GHOST is an alternative rule to decide on the accepted fork that does not suffer from lowered security with a higher stale block rate (in contrast to the longest chain rule). Instead, Ethereum uses the longest chain rule with the addition that some rewards are also paid to miners of stale blocks (so called *uncle blocks*). These uncle blocks do, however, not count towards the difficulty of a chain.

Additionally, Ethereum's consensus mechanism differs from Bitcoin in that Ethereum's longest chain algorithm was recently modified to incorporate uniform tie breaking [24], meant as a selfish mining countermeasure but which has been shown to actually increase an adversary's chances of catching up to the honest chain [11]. In order to account for these differences (i.e. uncle rewards and uniform tie breaking), we extend our double-spending MDP in Table 3.3. We use this modified model to find optimal strategies for double-spending in Ethereum in order to compare the security of Ethereum to that of Bitcoin.

In the MDP for Ethereum (cf. Table 3.3), r_s is the stale block rate, v_d is the value of the double-spending and r_u is the uncle reward (i.e. $\frac{7}{8}$ of a block reward). Every state includes a flag (where nr = not released, rel = released, inc = included) indicating whether an attacker block has been or will be included as an uncle in the honest chain. The actions *override* and *match* are feasible only when $l_a > l_h$ or $l_a \geq l_h$, respectively. The *exit* action corresponds to a successful double spend with k confirmations and is only feasible if $l_a > l_h$ and $l_a > k$. The *release* action corresponds to the release of the first block of the attackers fork with the intention to be included as uncle in the honest chain. Therefore, it is only feasible if $1 < l_h \leq 6$ and $l_a \geq 1$, since it is otherwise equivalent to a *match* or *override* action or the honest chain is too long to include it as uncle. With the *release* action no block is mined. With the *release* action, a state transitions from not released to released, which transitions to included with the next block mined on the honest chain. In Ethereum, γ is fixed at 0.5 and a match is possible even without a prepared block. We discount the mining costs $c_m \in [0, \alpha]$ in the state transition reward.

In Figure 3.10, we compare the the double-spending resilience of Ethereum ($r_s = 6.8\%$, cf. MDP in Table 3.3) and Bitcoin ($r_s = 0.41\%$, cf. MDP in Table 3.1), given $\gamma = 0, c_m = 0$ and $\omega = 0$. We rely on the US dollar values of the block rewards in order to provide a fair comparison. Since Bitcoin's

[2] `https://github.com/ethereum/wiki/wiki/White-Paper#modified-ghost-implementation`

[3] The Yellow Paper is the official Ethereum specification. Even though it contains the claim that a variant of GHOST is used, the specification itself contradicts this.

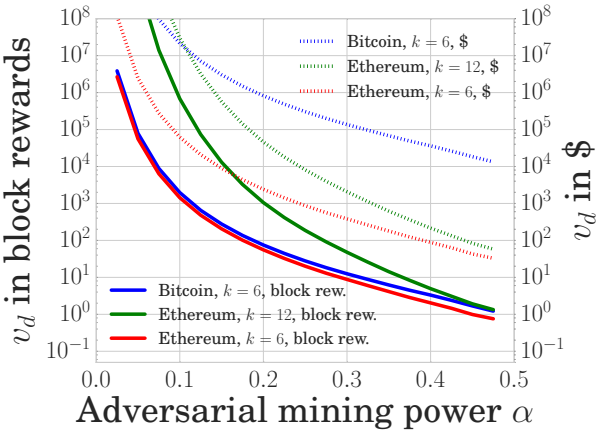


Figure 3.10: Double-spending resistance of Ethereum ($k \in \{6, 12\}$) vs. Bitcoin ($k = 6$). USD exchange rate of 2016-04-20.

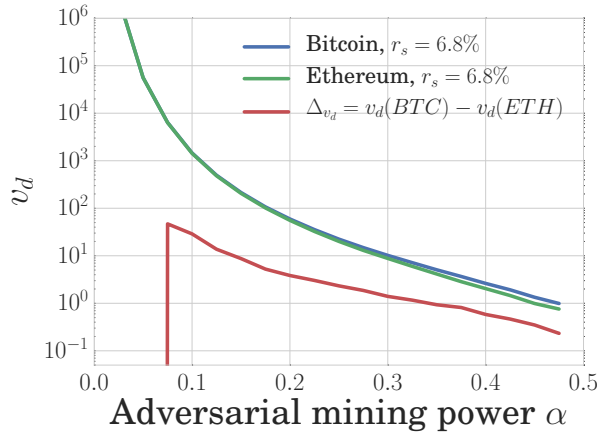


Figure 3.11: Direct comparison between the Protocols used by Ethereum and Bitcoin with $k = 6$ at a stale rate of $r_s = 6.8\%$.

block reward is more than 200 times higher than Ethereum's, Bitcoin is much more robust against double-spending attacks.

We observe that 6 Bitcoin[4] block confirmations are more resilient to double-spending than 6 Ethereum[5] block confirmations and more resilient than 12 Ethereum confirmations for adversaries with more than 11% of the total mining power. Note, however, that since the block interval of Ethereum is much shorter than that of Bitcoin, 12 Ethereum confirmations are likely to be generated in less than 4 minutes compared to an average of one hour

[4]Block generation time of 10 minutes.
[5]Block generation time between 10 and 20 seconds.

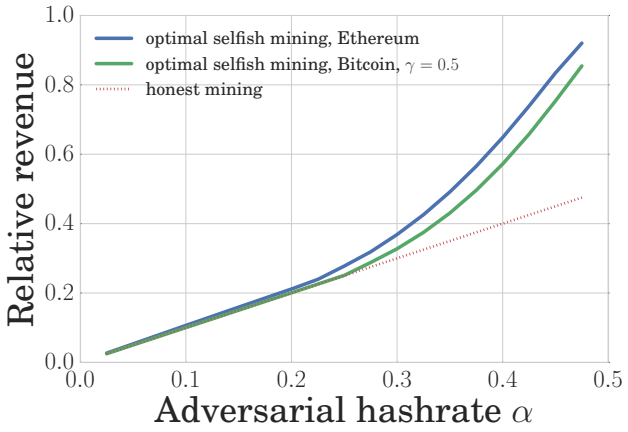


Figure 3.12: Optimal selfish mining in Bitcoin and Ethereum

for 6 Bitcoin confirmations. This also shows that the monetary value of the block reward directly impacts the double-spending security: the higher the block reward of a blockchain (e.g., in USD) the more resilient it is against double-spending.

In addition to comparing the instantiations of Bitcoin and Ethereum, we compare in Figure 3.11 the two blockchain protocols by setting Bitcoin's stale block rate equal to Ethereum's stale block rate. This allows us to objectively evaluate their security implications. We observe that the security of Ethereum's protocol is slightly weaker than Bitcoin's, and conclude that the uniform tie breaking and the uncle reward weaken the security of Ethereum.

In Figure 3.12, we additionally compare selfish mining in Bitcoin with selfish mining in Ethereum. Note that, since the adversary aims to increase his share of blocks that are included in the main chain, we do not consider uncle rewards. We observe that Ethereum is not as resilient against selfish mining attacks as Bitcoin.

State × Action	Resulting State	Probability	Reward	Condition
$(l_a, l_h, \cdot, nr), adopt$	$(1, 0, relevant, nr)$ $(0, 1, relevant, nr)$ $(0, 0, relevant, nr)$	α $(1-\alpha)\cdot(1-r_s)$ $(1-\alpha)\cdot r_s$	$-c_m$ $-c_m$ $-c_m$	- - -
$(l_a, l_h, \cdot, inc), adopt$	$(1, 0, relevant, nr)$ $(0, 1, relevant, nr)$ $(0, 0, relevant, nr)$	α $(1-\alpha)\cdot(1-r_s)$ $(1-\alpha)\cdot r_s$	$r_u - c_m$ $r_u - c_m$ $r_u - c_m$	- - -
$(l_a, l_h, \cdot, rel), adopt$	$(1, 0, relevant, rel)$ $(0, 1, relevant, inc)$ $(0, 0, relevant, rel)$	α $(1-\alpha)\cdot(1-r_s)$ $(1-\alpha)\cdot r_s$	$-c_m$ $-c_m$ $-c_m$	- - -
$(l_a, l_h, \cdot, \cdot), override$	$(l_a - l_h, 0, relevant, nr)$ $(l_a - l_h - 1, 1, relevant, nr)$ $(l_a - l_h - 1, 0, relevant, nr)$	α $(1-\alpha)\cdot(1-r_s)$ $(1-\alpha)\cdot r_s$	$l_h + 1 - c_m$ $l_h + 1 - c_m$ $l_h + 1 - c_m$	$l_a > l_h$ $l_a > l_h$ $l_a > l_h$
$(l_a, l_h, relevant, nr), wait$	$(l_a + 1, l_h, relevant, nr)$ $(l_a, l_h + 1, relevant, nr)$ $(l_a, l_h, relevant, nr)$	α $(1-\alpha)\cdot(1-r_s)$ $(1-\alpha)\cdot r_s$	$-c_m$ $-c_m$ $-c_m$	- - -
$(l_a, l_h, relevant, inc), wait$	$(l_a + 1, l_h, relevant, inc)$ $(l_a, l_h + 1, relevant, inc)$ $(l_a, l_h, relevant, inc)$	α $(1-\alpha)\cdot(1-r_s)$ $(1-\alpha)\cdot r_s$	$-c_m$ $-c_m$ $-c_m$	- - -
$(l_a, l_h, relevant, rel), wait$	$(l_a + 1, l_h, relevant, rel)$ $(l_a, l_h + 1, relevant, inc)$ $(l_a, l_h, relevant, rel)$	α $(1-\alpha)\cdot(1-r_s)$ $(1-\alpha)\cdot r_s$	$-c_m$ $-c_m$ $-c_m$	- - -
$(l_a, l_h, active, nr), wait$ $(l_a, l_h, relevant, nr), match$	$(l_a + 1, l_h, active, nr)$ $(l_a + 1, l_h, active, rel)$ $(l_a - l_h, 1, relevant, nr)$ $(l_a, l_h + 1, relevant, nr)$ $(l_a, l_h + 1, relevant, inc)$ $(l_a, l_h, active, nr)$ $(l_a, l_h, active, rel)$	α α $\gamma\cdot(1-\alpha)\cdot(1-r_s)$ $(1-\gamma)\cdot(1-\alpha)\cdot(1-r_s)$ $(1-\gamma)\cdot(1-\alpha)\cdot(1-r_s)$ $(1-\alpha)\cdot r_s$ $(1-\alpha)\cdot r_s$	$-c_m$ $-c_m$ $l_h - c_m$ $-c_m$ $-c_m$ $-c_m$ $-c_m$	$l_h > 6$ $l_h \leq 6$ - $l_h > 6$ $l_h \leq 6$ $l_h > 6$ $l_h \leq 6$
$(l_a, l_h, active, inc), wait$ $(l_a, l_h, relevant, inc), match$	$(l_a + 1, l_h, active, inc)$ $(l_a - l_h, 1, relevant, nr)$ $(l_a, l_h + 1, relevant, inc)$ $(l_a, l_h, active, inc)$	α $\gamma\cdot(1-\alpha)\cdot(1-r_s)$ $(1-\gamma)\cdot(1-\alpha)\cdot(1-r_s)$ $(1-\alpha)\cdot r_s$	$-c_m$ $l_h - c_m$ $-c_m$ $-c_m$	- - - -
$(l_a, l_h, active, rel), wait$ $(l_a, l_h, relevant, rel), match$	$(l_a + 1, l_h, active, rel)$ $(l_a - l_h, 1, relevant, nr)$ $(l_a, l_h + 1, relevant, inc)$ $(l_a, l_h, active, rel)$	α $\gamma\cdot(1-\alpha)\cdot(1-r_s)$ $(1-\gamma)\cdot(1-\alpha)\cdot(1-r_s)$ $(1-\alpha)\cdot r_s$	$-c_m$ $l_h - c_m$ $-c_m$ $-c_m$	- - - -
$(l_a, l_h, \cdot, nr), release$	$(l_a, l_h, \cdot, rel)$	1	0	$l_h \leq 6 \wedge l_h > 1 \wedge l_a \geq 1$
$(l_a, l_h, \cdot, \cdot), exit$	$exit$	1	$l_a + v_d$	$l_a > l_h \wedge l_a > k$

Table 3.3: State transition and reward matrices for an MDP for optimal double-spending strategies in Ethereum. r_s is the stale block rate, v_d is the value of the double-spending and r_u is the uncle reward (i.e. $\frac{7}{8}$). Every state includes a flag (where nr = not released, rel = released, inc = included) indicating whether an attacker block has been or will be included as an uncle in the honest chain.

Chapter 4

Practical Security Analysis of different Blockchains

In this chapter, we consider two blockchains, Ethereum and Stellar, from a more practical perspective. Namely, we consider the implementation and current state of the blockchain's network and review their impact on the security. We present vulnerabilities in Ethereum that can cause consensus issues and enable a limited attacker to perform a denial of service attack against the whole network with very low resource requirements. We also give some background on the Stellar Consensus Protocol [8] and show that it does not offer sufficient security given the current state of the Stellar network.

4.1 Ethereum

Ethereum is a Proof of Work (PoW) based blockchain that, in addition to providing a digital currency, allows the execution of so called smart contracts, small programs that are executed deterministically and that store state on the blockchain. Contrary to many other PoW blockchains, Ethereum is not a fork of Bitcoin and instead is designed and implemented from scratch. In this section, we analyse the security of Ethereum's peer-to-peer network and describe three vulnerabilities.

4.1.1 Background

In the following we give a short overview over a part of the Ethereum peer-to-peer protocol that is necessary to understand the vulnerabilities described in Sections 4.1.2 and 4.1.3

Block Propagation

Blocks are propagated in three different ways in the Ethereum network. Firstly, a node P that receives a new block directly pushes the block to $\sqrt{n}$ of its connected peers, where n is the total number of connected peers. Secondly, P will send a NewBlockHashes message to all of its peers, advertising a new block. When a node N receives an advertisement, it will request the block explicitely after 0.5 seconds from a random peer from which it received a corresponding advertisement (unless N received the block from another peer in the meantime) and then forgets about all other advertisements for that block. This means that if N requests the block from P and P fails to answer, it will not request the block from any other peers. If N misses a block, the third method for block propagation is used, which is the block synchronisation explained below.

Block Synchronisation

A node will only synchronise with one other node at a time. A node N starts a block synchronisation in the following cases:

- N starts a connection to a new peer with higher advertised total difficulty (e.g. after joining or rejoining the network).
- A node advertising a higher total difficulty than N connects to N.
- N receives a block with higher total difficulty than the head of its current blockchain and is missing some of the block's ancestors.

The block synchronisation (Fig. 4.1) works as follows:

1. The node with lower total difficulty (node N) sends a `GetBlockHeaders` request to the node with higher total difficulty (node P), requesting the header of the latest block of P.
2. P responds with a `BlockHeaders` message containing the block header of the block specified in the received `GetBlockHeaders` message.
3. N requests the `MaxHeaderFetch` ($= 256$) blocks starting at `MaxHeaderFetch` blocks below the height of its own blockchain.
4. P sends up to `MaxHeaderFetch` of the requested blocks (but may send fewer).
5. If none of the blocks received from P are in N's blockchain, N starts a binary search over its own blockchain to find a common ancestor, requesting one block from P per step in the search.
6. As soon as N finds a common ancestor, N requests block headers and bodies from P starting from the common ancestor. N asks for `MaxHeaderFetch` blocks per request, but P may send fewer.

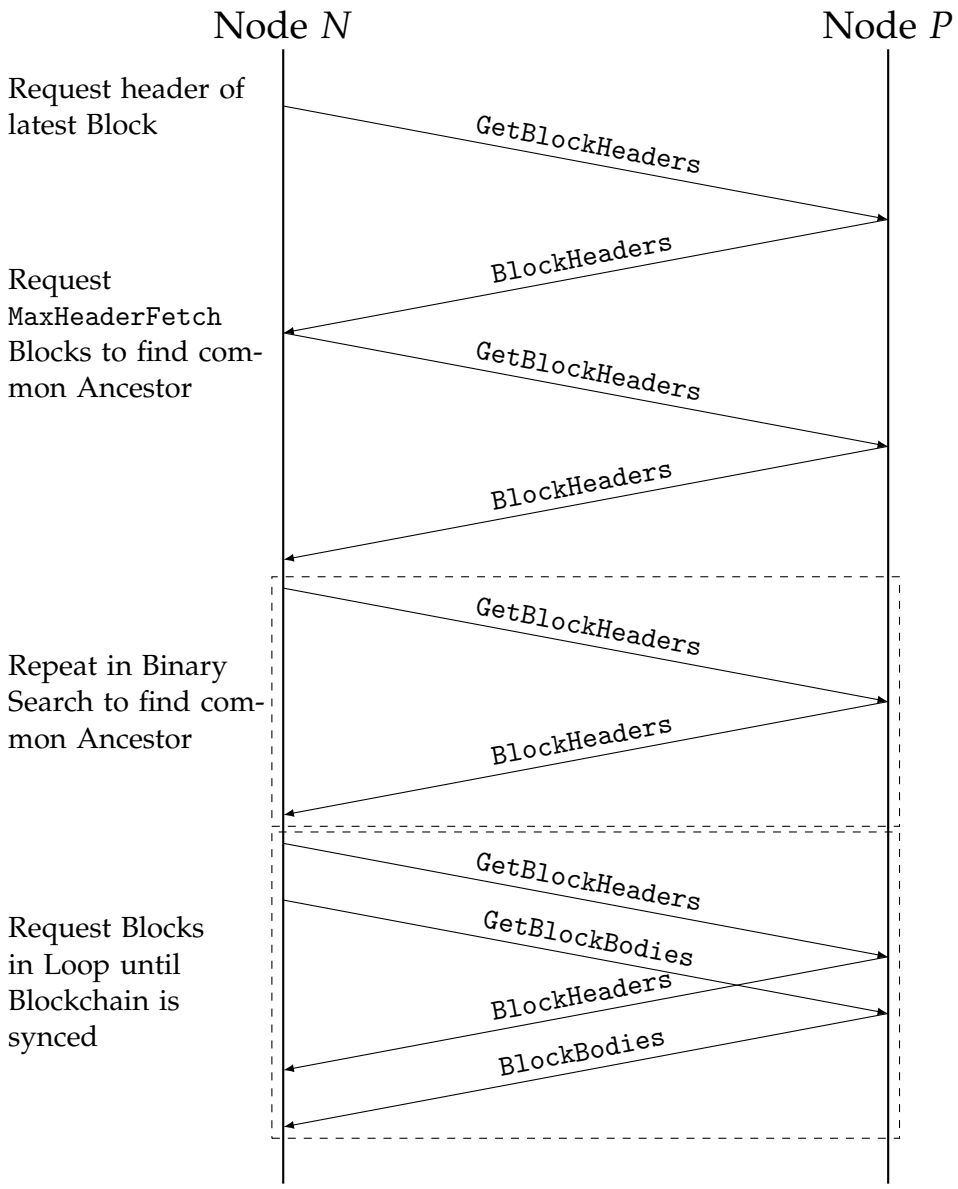


Figure 4.1: Block Synchronisation between two Nodes N and P

4.1.2 Eclipse Attack using a Block Synchronisation Flaw

The following is a consensus critical flaw in the Ethereum peer-to-peer protocol. Using this vulnerability, an attacker P can keep a victim N from receiving a block at height $n+1$ almost indefinitely. While N may receive later blocks from other peers, it will be stalled at height n since it misses a link in the blockchain. An attacker P can work as follows:

1. P creates a long blockchain starting from the genesis block. This can

be done by decreasing the difficulty for each block, thus shortening the time needed to generate a valid proof of work. Creating the alternative blockchain takes a relatively large amount of precomputation. However, once the alternative blockchain exists, it does not need to be computed again and can be used for multiple attacks.

2. If *P*'s chain is shorter than the valid chain, *P* forges a block with a high block number and valid proof of work, i.e. it creates a block without parent that uses an arbitrary value as parent hash.
3. If the attacker blockchain is shorter than the valid blockchain, *P* also needs to forge a block with block number `MaxHeaderFetch` below the height of *N*s blockchain and blocks at heights that will be queried in the binary search but are not contained in *P*s blockchain.
4. *P* connects to Node *N*, advertising a high total difficulty.
5. *P* sends the block header with the highest block number from his chain (or the block created in step 2) to *N* in response to the first `GetBlockHeaders` request from *N*
6. In response to the second `GetBlockHeaders` request from *N*, *P* sends the block from its seperate blockchain corresponding to the block number specified in *N*'s message (or the block forged in step 3).
7. In response to the requests from *N*s binary search, *P* responds again with block headers from its own blockchain until the genesis block is reached.
8. *N* will now request blocks starting from the genesis block. *P* responds with blocks from its own blockchain, while sending only one block in response for each request. For each request, *P* introduces a delay of up to 3 seconds.
9. As long as the attack is in progress, *N* will not be able to synchronise with any other peers.

Attack Scenario

A node will sometimes miss a block (i.e. receives neither the block directly nor a NewBlockHashes message, or the asked peer does not respond to the request to get the block corresponding to NewBlockHashes) and needs to synchronise with the network. *N* will start a synchronisation if it receives a NewBlock message containing a Block with total difficulty higher than the current total difficulty of the node plus the difficulty of the block. In this case, the node will try to synchronise with the peer that sent the NewBlock message.

However, a node will never attempt to synchronise with more than one node at the same time. This means that if an adversary connects to a node N and starts the synchronisation attack, the node will not be able to synchronise with the valid chain once it misses a block, thus keeping it from receiving the missing block as long as the synchronisation attack persists.

Let us assume that N has all blocks up to block number n from the valid chain, but does not receive block $n+1$ while the synchronisation attack is in progress. N will still receive newly mined blocks from the valid chain (e.g. blocks $n+2$, $n+3$), but it will not accept them as part of the blockchain yet, since it cannot connect them to its blockchain due to the missing block.

Normally, N would start a synchronisation with P' to receive all intermediate blocks when receiving block $n+i$ ($i \geq 2$) from peer P'. However, since N is already synchronising with P, N will not synchronise with P'.

Since the difficulty to mine a block can be reduced with every block by setting the timestamp of each block to the parent timestamp plus 13 (in the Frontier phase[1]) until it reaches the minimum difficulty, a chain surpassing the length of the valid chain can be mined with a single consumer grade GPU within a few weeks.

With such a long chain, the attack can be sustained for weeks. When mining at or close to the minimum difficulty, a block can be mined in fractions of a second. Since we can delay a block for up to 3 seconds, an attacker can even continue to mine blocks during the attack and thus only has to stop the attack once the timestamp is too far ahead of the current time.

Due to the block propagation mechanism described in Section 4.1.1, an attacker can even influence the rate of missed blocks at a victim node in order to make the attack successful more quickly. The attacker can connect many nodes to his victim that are modified to never push a block directly but always send NewBlockHashes immediately and never answer to a request to get an advertised block. Since the victim only asks one peer for an advertised block, it will not receive the block if it requests the block from one of the attacker nodes.

Impact

The vulnerability can cause Denial of Service and can be used to double spend confirmed transactions. The attack, when executed against multiple victims also causes multiple forks in the blockchain and severly impacts consensus. In our assessment, the vulnerability is critical with a CVSS score of 9.1[2].

[1] Frontier is the name for the beta phase of Ethereum.

[2] CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:H/E:F/RC:C/CR:L/IR:H/AR:H

Denial of Service

The attack described above is trivially a denial of service attack. While the attack is in progress, the attacked node N will not append any blocks to its blockchain and thus not update the accepted state.

This also has implications on mining. If N is a mining node, it will continue to mine on what N accepts as valid blockchain, i.e. N will mine on top of block n and not receive any rewards for mined blocks (except an uncle reward for the first block, if the attack stops early enough). Thus, the attack is effectively also a denial of service attack on mining.

Since the resource requirements of the attack are very low, an attacker could easily mount a denial of service attack on the whole network (see section 4.1.2).

Double Spending

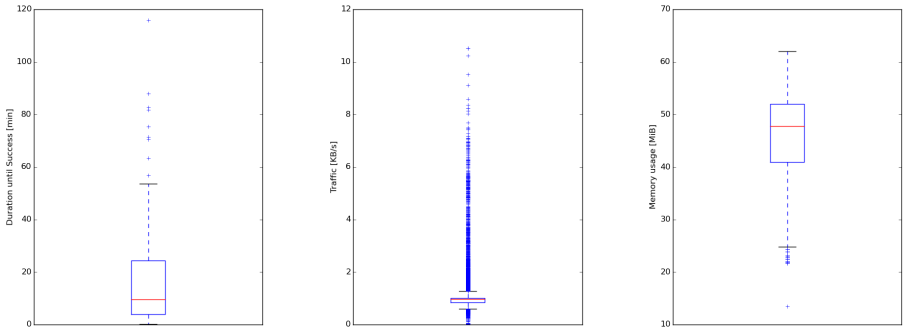
While the synchronisation attack is in progress and the victim N is stalled at block n, the attacker can connect a second node to N on which he mines a block on top of block n on the valid chain. N will accept this block as new head of his chain and will accept any transaction included in this block. However, since the network already has a chain with higher total difficulty, the rest of the network will not accept the transaction. This allows an attacker to double spend a transaction.

Implementation

The attack was implemented by modifying a `geth` client to mine a low difficulty chain as described above that surpasses the length of the valid chain and by introducing a delay of 2 seconds for responses to peers. The client was also modified to advertise a total difficulty higher than the difficulty of the valid chain. The adversarial chain with 1.2 million blocks was mined within 3 weeks on a single Radeon R9 280x GPU. After mining the chain and introducing delays in the client, the attack can then be executed as described in section 4.1.2.

Evaluation

The attack was tested on a victim node connected to the Ethereum network with the default settings. In our experiments, the attacker node is isolated from the network and only connects to the victim node. We ran the attack 100 times and measured the duration until the attack was successful (i.e. the time until the victim node is stalled at some block), and the network, memory and CPU usage of the attacker node (Fig. 4.2).



(a) Duration until Attack is successful (b) Network Traffic (Upload) (c) Memory Usage

Figure 4.2: Resource usage for the attacker node

The duration from start of the attack until success has a median value of 9.48 minutes, a mean of 18.68 minutes and a maximum of 115.87 minutes. The attack requires only 0.96 KB/s upload on average with a peak of 10.52 KB/s (download traffic is negligible). The attacker node requires 45 MB of memory on average (max. 62 MB) and has an average CPU usage of 0.07% on an Intel Core i7-4770 where 100% is one core.

Due to the small resource requirements, a Denial of Service attack on the whole network that would cause multiple forks is feasible with constraint resources. The Ethereum network currently consists of approximately 6500 nodes. Assuming the resource requirements scale linearly, attacking the whole network would require an upload bandwidth of approximately 6.5 MB/s, and approximately 307 GB of memory with a CPU usage of 455% (Intel Core i7-4770). Since attacking the whole network does not require running 6500 full nodes, the memory requirements are likely significantly lower. Attacking the whole network would thus be possible with a few desktop PCs, a fibre connection and a router that is able to sustain 6500 TCP connections. Note that it would be enough to attack mining nodes for a denial of service attack, i.e. the estimated requirements are an upper bound and the practical requirements are likely a lot lower.

The duration until the attack is successful can be significantly shortened by connecting additional nodes to the victim that do not forward blocks but always advertise block hashes quickly. This cuts down the median time until the attack is successful to 2.6 minutes (with a maximum of 11.6 minutes).

Possible Countermeasures

- Request block from other peers that have sent the NewBlockHashes message (instead of just 1 random peer), if the first asked peer does not respond. This would decrease the number of missed blocks.
- Allow to synchronise to more than one node. This has to be considered carefully, however, due to the increased load on the network.

4.1.3 Other Vulnerabilities

While investigating the Eclipse attack described above, we found other vulnerabilities with a lower impact and relatively simple fixes. These vulnerabilities are described below.

Synchronising to shorter chain with higher total difficulty

This vulnerability is present in the official golang implementation of the Ethereum client (`geth`), at least up to version 1.4.3. It is likely also present in other implementations.

A bug in the implementation of the block synchronisation makes it impossible for a node to synchronise to a shorter chain with higher total difficulty than the currently accepted chain.

When a node N synchronises the blockchain with a peer P, it requests 256 blocks from P, starting at block $n - 256$, where n is the height of Ns chain (see Section 4.1.1). If N does not receive any blocks in response, it will disconnect from P and will not continue with the synchronisation. While this seems reasonable at first, since a chain can have a higher difficulty than another longer chain, this could potentially be used in an attack on a newly joining node as follows:

1. P mines a chain that is longer than the valid chain by at least 256 blocks (which is feasible, see section 4.1.2).
2. P advertises a higher total difficulty than the valid chain.
3. P connects to newly joining node N.
4. N synchronises with node P and receives his chain.
5. Once Ns chain is at least 256 blocks longer than the valid chain, N will not be able to synchronise with a non malicious node.

Possible Countermeasure:

Request blocks starting at block $\min(n, n') - 256$, where n is the length of the chain at node N and n' is the length of the chain at node P.

Increasing the MinimumDifficulty Parameter in geth

This vulnerability consists of an implementation bug in the official golang implementation of the Ethereum client (`geth`), at least up to version 1.4.3. The bug in the block difficulty calculation can cause the MinimumDifficulty Parameter to increase and possibly cause a node to no longer accept the valid chain.

The bug is part of the function to calculate the difficulty of a block (in package `core`, i.e. the function `CalcDifficulty` in versions $< 1.3.4$, and the functions `calcDifficultyFrontier` and `calcDifficultyHomestead` in versions $\geq 1.3.4$). The functions first calculate the difficulty by subtracting or adding a fraction of the parent difficulty to the parent difficulty, depending on the difference in the timestamp to the parent. If this calculated difficulty is lower than `MinimumDifficulty`, the difficulty is set to `MinimumDifficulty`. Starting at block 200000, an exponential component is added to the difficulty afterwards ($2^{n/100000-2}$, where n is the block number).

The listing below shows the implementation of `calcDifficultyFrontier` (the implementation for `calcDifficultyHomestead` is analogous):

```
1  func calcDifficultyFrontier(time, parentTime uint64, parentNumber,
       parentDiff *big.Int) *big.Int {
2      diff := new(big.Int)
3      adjust := new(big.Int).Div(parentDiff, params.DifficultyBoundDivisor
           )
4      bigTime := new(big.Int)
5      bigParentTime := new(big.Int)
6
7      bigTime.SetUint64(time)
8      bigParentTime.SetUint64(parentTime)
9
10     if bigTime.Sub(bigTime, bigParentTime).Cmp(params.DurationLimit) < 0
           {
11         diff.Add(parentDiff, adjust)
12     } else {
13         diff.Sub(parentDiff, adjust)
14     }
15     if diff.Cmp(params.MinimumDifficulty) < 0 {
16         diff = params.MinimumDifficulty
17     }
18
19     periodCount := new(big.Int).Add(parentNumber, common.Big1)
20     periodCount.Div(periodCount, ExpDiffPeriod)
21     if periodCount.Cmp(common.Big1) > 0 {
22         // diff = diff + 2^(periodCount - 2)
23         expDiff := periodCount.Sub(periodCount, common.Big2)
24         expDiff.Exp(common.Big2, expDiff, nil)
25         diff.Add(diff, expDiff)
26         diff = common.BigMax(diff, params.MinimumDifficulty)
27     }
28
29     return diff
30 }
```

In line 16 of the listing, `params.MinimumDifficulty` is assigned to `diff`. The value is not copied however. Instead, `diff` and `params.MinimumDifficulty` now reference the same object. In line 25 of the listing, the exponential component is added to `diff`, i.e. the sum is written to the object referenced by `diff` and thus by `params.MinimumDifficulty`.

This could potentially be used for an attack on a node N by an attacker P as follows:

1. P mines a long chain at minimum difficulty (using the flawed implementation to calculate the difficulty).
2. P connects to N, advertising a higher difficulty than Ns chain.
3. N synchronises with P.
4. Since N checks the difficulty for each block received by P, for every block, `params.MinimumDifficulty` is increased by the exponential component.
5. After syncing enough blocks, Ns `params.MinimumDifficulty` is larger than the difficulty of the current head of the blockchain.
6. N can no longer synchronise with the valid blockchain.

Possible Countermeasure:

Replace

```
diff = params.MinimumDifficulty
```

with

```
diff.Set(params.MinimumDifficulty)
```

(and analogously for `calcDifficultyHomestead`).

4.1.4 Ethereum vs. GHOST

GHOST [6] is an alternative to the longest chain rule for establishing consensus in proof of work based block chains and aims to alleviate the negative impacts of stale blocks. Instead of accepting the blocks of the longest chain, GHOST considers the whole tree of blocks and chooses on each height the block with the heaviest subtree, i.e. the mining power spent on stale blocks is not wasted but instead also strengthens their ancestors. This allows a blockchain to shorten block intervals or increase the block size without compromising security against 50% attacks [6].

While Ethereum claims to use a protocol derived from GHOST, it has substantial differences and thus the security guarantees of GHOST are not valid for Ethereum. Ethereum allows miners to include uncles, stale blocks that

are children of one of the last 6 ancestor blocks, in mined blocks. While uncle blocks that are included in a block receive a reward and increase the reward for the block that includes them, they do not count towards the total difficulty of a chain, i.e. Ethereum uses a longest chain rule with added rewards for stale blocks.

4.2 Stellar

Stellar is a payment network and cryptocurrency that is not based on Proof of Work and instead uses the Stellar Consensus Protocol (SCP, cf. Section 4.2.1) to ensure consensus.

4.2.1 Stellar Consensus Protocol

The Stellar Consensus Protocol (SCP) is a protocol for Federated Byzantine Agreement (FBA), a new consensus model introduced in the SCP paper [25]. In contrast to other Byzantine Agreement models, FBA does not require an a priori defined and unanimously agreed upon membership list. Instead, every node decides which other participants it considers important and bases the decision of the validity of a statement on the messages received from these important participants.

A quorum in FBA consists of a *quorum slice* for every participating node. A quorum slice for node v is a set of nodes that, by asserting a statement, convinces v that no functioning node will contradict that statement. A quorum is a set of nodes containing a quorum slice for each node in the set.

Agreement can only be guaranteed by a protocol in an FBA system if the quorum slices satisfy *quorum intersection* after deleting byzantine nodes, i.e. if any two quorums share at least one honest node.

4.2.2 Security of the Stellar Network

The Stellar Consensus Protocol bases its safety guarantees on the ability of its users to choose quorum slices correctly. If the quorum intersection property is not satisfied, safety is trivially not guaranteed since two disjoint quorums can accept contradicting statements. Even if quorum intersection is satisfied, it is possible that only few entities need to be compromised to convince different nodes of contradicting statements, especially since one entity can control multiple nodes.

The state of the current network is not ideal. At the time of writing[3], only 10 validator nodes form a quorum with quorum intersection. Some additional

[3] i.e. 2016-02-26

nodes are used as validators but do not themselves have quorum sets on which they base their decisions.

The quorum set relation is shown in Figure 4.3. The nodes are identified by their shortened public key. With only two byzantine nodes, two quorums can exist without containing a common honest node. Figure 4.4 shows the graph of the quorum set relation after deleting the nodes "GD5DJ" and "GDPJ4". Deleting the two nodes creates a disjoint graph, visualising the two quorums that can achieve contradictory consensus.

According to a list of validators from the stellar github repository[4], the two nodes belong to the same entity (strllar.org), i.e. compromising a single entity is enough to compromise consensus. This shows that the Stellar network is not secure in its current state.

[4] `https://github.com/stellar/docs/blob/master/validators.md` (retrieved 2016-02-26)

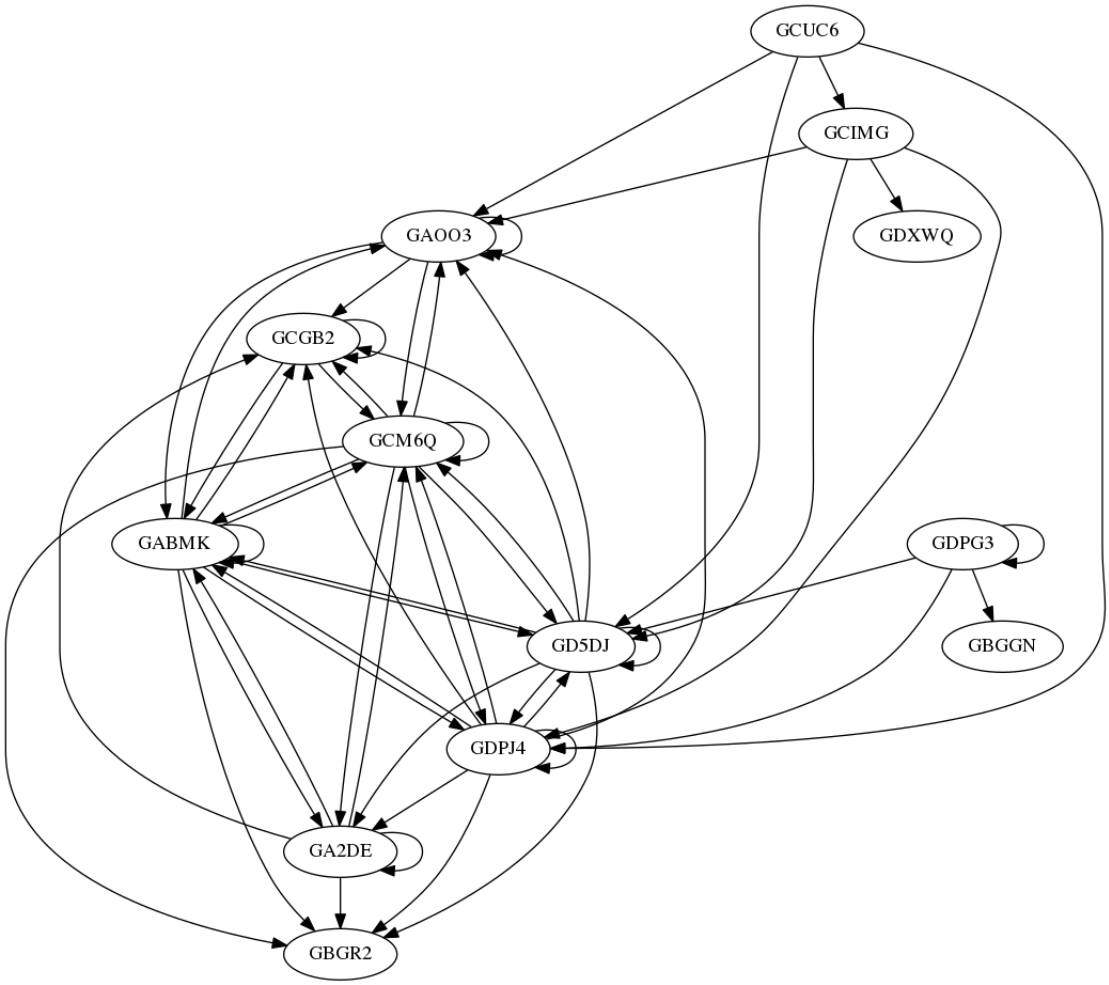


Figure 4.3: Quorum Set Relation in the current Stellar Network

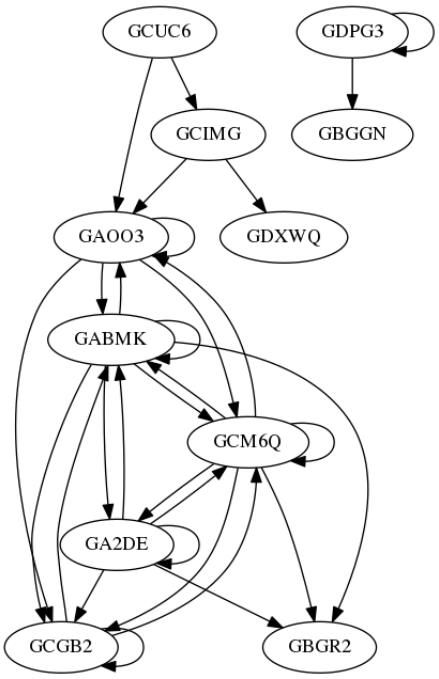


Figure 4.4: Quorum Set Relation in the current Stellar Network after removing two nodes

Chapter 5

Related Work

Several contributions consider double-spending attacks in Bitcoin [26, 4] but do not take into account optimal adversarial strategies. Karame *et al.* [5] investigate double-spending of unconfirmed transactions.

Eyal and Sirer [7] show that a selfish miner can increase his relative mining revenue by keeping some mined blocks private and only selectively publishing them. Similarly, Courtois and Bahack [27] study subversive mining strategies. Our work is closely related to the work of Sapirshtein *et al.* [11], where the authors analyse optimal adversarial strategies for selfish mining in Bitcoin. Unlike [11], our work however captures parameters such as network delays and different block sizes through the stale block rate and allows us to devise optimal adversarial selfish mining strategies for any chain-based PoW-powered blockchain. Additionally, we analyse optimal double-spending strategies to quantify the required value that an adversary needs to double-spend in order to be profitable, given additional parameters such as the number of required confirmations and the adversarial mining costs. For both, selfish mining and double-spending, our model also considers eclipse attacks.

Related to the vulnerability discovered in Ethereum that allows an attacker to eclipse a victim (cf. Section 4.1.2), Gervais *et al.* [13] show that a resource-constrained attacker can perform (partial) eclipse attacks on Bitcoin with only one TCP connection. While our MDP model does not capture such partial eclipse attacks, we do consider stronger full eclipse attacks.

GHOST [28] is an alternative to the longest chain rule for establishing consensus in PoW based blockchains in which stale blocks also contribute to the security of the chain and in which stale blocks therefore do not impact the security.

There also exist many alternatives to Proof of Work. In *Proof of Stake* (PoS) [2], nodes "stake" some value and based on the staked amount get a share of the

vote of whether a block is valid. *Proof of Burn* (PoB) is a proposal to replace PoW by burning coins, i.e. sending them to an address that is verifiably unspendable, such that they can no longer be spent. However, existing PoB-based blockchains rely on burning coins from PoW blockchains in order to create blocks and therefore can not stand on their own.

Many additional proposals [29, 30, 31, 8] rely on classical Byzantine Fault Tolerant consensus protocols in order to improve consensus efficiency and scale to a larger number of transactions. Such protocols are however often not as open to participation (e.g. permissioned blockchains such as [31]) or suffer from centralisation problems (cf. Section 4.2). Recent studies propose to combine the use of PoW with BFT protocols to implement highly-performant open consensus protocols (Byzcoin [32]). Intel recently proposed Proof of Elapsed Time (PoET) [3], which allows participants to achieve consensus by leveraging Intel's SGX platform.

Chapter 6

Conclusions

In this thesis, we presented a framework to quantify the security of different Proof of Work blockchains, taking into account parameters such as network delays and the dollar value of block rewards. By evaluating the impact of different parameters on the security of PoW blockchains, we can objectively compare the security provisions of different blockchain instances regarding their resilience to optimal selfish mining and double-spending strategies.

For example, we find that, in order to match the security of 6 confirmations in Bitcoin, Ethereum needs at least 37 confirmations (given an adversary with 30% of the mining power). By quantitatively comparing the consensus protocols of Bitcoin and Ethereum, we also find that Bitcoin's protocol offers more security than Ethereums, which rewards stale blocks and performs uniform tie breaking for fork resolutions. Our results also show that the value of the block reward has a strong influence on the double-spending resilience of a chain. This implies, for example, that the halving of the block reward in Bitcoin weakens the security of Bitcoin if the value of one Bitcoin remains the same.

Our work quantitatively evaluates the impact of the stale block rate on optimal selfish mining and double-spending in PoW blockchains (cf. Figure 3.8 and Figure 3.2. Our results allow merchants to set the number of required confirmations depending on the value of the transaction and quantify the risk of double-spending, without simply relying on the famous required six block confirmation. Our insights also help miners in quantifying any blockchain's resilience against selfish mining.

In our practical security analysis on Ethereum, we discovered three vulnerabilities in Ethereum's peer-to-peer protocol and in one Ethereum client. We also show that Stellar's network is currently heavily centralised and is, in its current state, vulnerable to a network split that would allow a single entity to perform successful double-spending attacks.

References

[1] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[2] QuantumMechanic. Proof of stake. Available from: `https://bitcointalk.org/index.php?topic=27787.0`.

[3] Intel. Proof of elapsed time (poet). Available from: `http://intelledger.github.io/`.

[4] Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014.

[5] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, New York, NY, USA, 2012. ACM.

[6] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. *IACR Cryptology ePrint Archive*, 2013:881, 2013.

[7] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[8] D. Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. Available from: `https://www.stellar.org/papers/stellar-consensus-protocol.pdf`.

[9] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[10] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[11] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *arXiv preprint arXiv:1507.06183*, 2015.

[12] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. 2015.

[13] Arthur Gervais, Hubert Ritzdorf, Ghassan O Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 692–705. ACM, 2015.

[14] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, and Emin Gün. On scaling decentralized blockchains.

[15] Visa Inc. at a Glance, 2015. Available from: `https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf`.

[16] Matt Corallo. Bitcoin relay network. Available from: `http://bitcoinrelaynetwork.org/`.

[17] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. Cryptology ePrint Archive, Report 2016/555, 2016. `https://eprint.iacr.org/2016/555`.

[18] Ethereum. ethstats. Available from: `https://ethstats.net/`.

[19] Ethereum. ethernodes. Available from: `https://www.ethernodes.org/network/1`.

[20] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. Technical report, IACR Cryptology ePrint Archive 2015, 2015.

[21] Ronald A Howard. *Dynamic Probabilistic Systems, Volume I: Markov Models*, volume 1. Courier Corporation, 2012.

[22] John G Kemeny, J Laurie Snell, and Gerald L Thompson. Finite mathematics. *DC Murdoch, Linear Algebra for Undergraduates*, 1974.

[23] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[24] Ethereum. Ethereum tie breaking. Available from: `https://github.com/ethereum/go-ethereum/commit/bcf565730b1816304947021080981245d084a930`.

[25] DAVID MAZIERES. The stellar consensus protocol: A federated model for internet-level consensus.

[26] The Finney Attack, 2013. Available from: `https://en.bitcoin.it/wiki/Weaknesses#The_.22Finney.22_attack`.

[27] Nicolas T. Courtois and Lear Bahack. On subversive miner strategies and block withholding attack in bitcoin digital currency. *CoRR*, abs/1402.1718, 2014.

[28] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[29] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. *arXiv preprint arXiv:1602.06997*, 2016.

[30] Marko Vukolic. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *Proceedings of the IFIP WG 11.4 Workshop iNetSec 2015*. 2015.

[31] IBM. Ibm openblockchain. Available from: `http://www.ibm.com/blockchain/`.

[32] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. *CoRR*, abs/1602.06997, 2016.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Security of Blockchain Technologies

Authored by (in block letters):

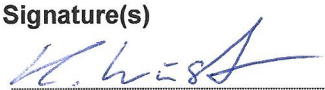
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Wüst	Karl

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 2016-07-01	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.