On the security of Bitcoin and DAG-based performance solutions

Danai Balla
AL1.19.0017

Examination committee:
Aris Pagourtzis, School of Electrical and Computer Engineering, National Technical University of Athens.
Lefteris Kokoris-Kogias, Institute of Science and Technology Austria.
Nikos Leonardos, Department of Informatics and Telecommunications, National and Kapodistrian University of Athens.

Supervisor:
Aris Pagourtzis, Professor, School of Electrical and Computer Engineering, National Technical University of Athens.

Μεταπτυχιακό Πρόγραμμα «Αλγόριθμοι, Λογική και Διακριτά Μαθηματικά» – 2016

ABSTRACT

The Bitcoin protocol has been proposed as a decentralized payment system in which any participant can verify their transactions. However, it performs significantly worse than its centralized counterparts, and many modifications have been proposed since its inception with the aim of improving performance. In this thesis, we explore a category of protocols that use an alternative chain selection rule or use a directed acyclic graph of blocks instead of a blockchain, with a focus on their security guarantees.

We begin by describing the Bitcoin protocol and presenting an overview of its security proofs over time, with the aim of understanding the proof methods and the resulting performance limitations. Then, we describe GHOST, which uses the GHOST rule instead of the longest chain rule of Bitcoin and serves as a stepping stone for DAG-based protocols such as PHANTOM, GHOSTDAG, SPECTRE, and Conflux. We also present an attempt for an alternative proof of the security of GHOST. Finally, we describe these DAG-based protocols and present their security guarantees.

ΣΥΝΟΨΗ

Το πρωτόκολλο Bitcoin έχει προταθεί ως αποκεντρωμένο σύστημα πληρωμών στο οποίο οποιοσδήποτε συμμετέχων μπορεί να επαληθεύσει τις συναλλαγές του. Όμως, η απόδοση του είναι σημαντικά μικρότερη από τα αντίστοιχα κεντροποιημένα συστήματα, και από τη σύλληψή του έχουν προταθεί πολλές τροποποιήσεις του που έχουν στόχο να αυξήσουν την απόδοση του. Σε αυτή τη διπλωματική, εξερευνούμε μια κατηγορία πρωτοκόλλων τα οποία χρησιμοποιούν διαφορετικό κανόνα επιλογής αλυσίδας ή χρησιμοποιούν κατευθυνόμενο ακυκλικό γράφημα αντί για αλυσίδα από block, εστιάζοντας στις εγγυήσεις ασφάλειας τους.

Αρχικά περιγράφουμε το πρωτόκολλο Bitcoin και παρουσιάζουμε μια επισκόπηση των αποδείξεων της ασφάλειάς του, με σκοπό να κατανοήσουμε τις μεθόδους που χρησιμοποιήθηκαν και τους περιορισμούς στην απόδοση που προκύπτουν από αυτές. Στη συνέχεια, περιγράφουμε το πρωτόκολλο GHOST, το οποίο στη θέση του κανόνα της μεγαλύτερης αλυσίδας του Bitcoin χρησιμοποιεί τον κανόνα GHOST, και είναι η βάση για πρωτόκολλα που βασίζονται σε κατευθυνόμενο ακυκλικό γράφημα όπως τα PHANTOM, GHOSTDAG, SPECTRE και Conflux. Παρουσιάζουμε επίσης μια απόπειρα για μια εναλλακτική απόδειξη της ασφάλειας του GHOST. Τέλος, περιγράφουμε τα παραπάνω πρωτόκολλα που βασίζονται σε κατευθυνόμενα ακυκλικά γραφήματα και παρουσιάζουμε τις εγγυήσεις ασφάλειάς τους.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα αρχικά να ευχαριστήσω τον επιβλέποντα καθηγητή μου Άρη Παγουρτζή για την πολύτιμη καθοδήγηση και στήριξη που μου προσέφερε κατά τη διάρκεια εκπόνησης αυτής της διπλωματικής, και για το γεγονός ότι μέσα από τα μαθήματα του με κινητοποίησε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα. Επίσης, θα ήθελα να ευχαριστήσω την υποψήφια διδάκτορα Pourandokht Behrouz για τις ιδέες που μοιραστήκαμε και τον χρόνο που αφιέρωσε στις συζητήσεις μας, οι οποίες ήταν πολύτιμες για την εκπόνηση αυτής της διπλωματικής.

Θα ήθελα επίσης να ευχαριστήσω τους συμφοιτητές μου στο ΑΛΜΑ με τους οποίους ξεπεράσαμε μαζί κάθε δυσκολία που αντιμετωπίσαμε στην κατανόηση του απαιτητικού αλλά γοητευτικού πεδίου της Θεωρητικής Πληροφορικής. Ιδιαίτερα θα ήθελα να ευχαριστήσω τον Φίλιππο, με τον οποίο μεγαλώσαμε μαζί ως μαθηματικοί και μοιραστήκαμε μαθηματικές ανησυχίες τα τελευταία 8 χρόνια.

Τέλος, θα ήθελα να ευχαριστήσω τον Π. ο οποίος ήταν πάντα εκεί για να με βοηθήσει όταν τα μαθηματικά μου δεν πήγαιναν καλά, και τον Δημήτρη για τη στήριξη που μου έχει προσφέρει τα τελευταία χρόνια και την υπομονή του να ακούει τις ιδέες και τις σκέψεις μου για κάθε τι που μελετάω.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Bitcoin, its performance, and Directed Acyclic Graphs

The Bitcoin protocol [2] was proposed in 2008 as a *decentralized* and *permissionless* payment system. Decentralized means that multiple entities run the protocol, and permissionless means that any entity can join or leave the protocol without the need for permission. Transactions are stored in a *distributed ledger*, a database that is shared and syncrhonized by all participants. Before transactions are added to the distributed ledger, they are organized into a *blockchain*, and the *longest chain rule* is used to distinguish between valid and invalid transactions. The blockchain is a sequence of chunks of data called *blocks*, and every block is linked to a previous one, forming a chain. The longest chain rule dictates that when having two distinct blockchain structures, the valid data is the data that appears in the longest one. Participants of the protocol, called *miners*, collect transactions into a block, link the block to the last block of the longest chain, and attempt to solve a moderately hard computational puzzle. If they succeed, the block is considered valid, and they broadcast it to the network.

In order for a distributed ledger protocol to be used as a payment system, it should have good performance and security. The performance of distributed ledger protocols is measured using three aspects: *scalability*, *throughput*, and *latency* [21]. Scalability is the ability of the protocol to add greater amount of data into the ledger as the number of its users grows, throughput measures the amount of data adder per unit of time and latency measures the amount of time that should elapse from the moment data is added to the ledger until the data is considered permanently added. Regarding security of distributed ledger protocols, one concern that arises is that at the same moment in time two participants may have different views of the ledger. Furthermore, it might be the case that some participants do not follow the protocol rules and may deviate from the protocol arbitrarily. The security of a distributed ledger protocol guarantees that despite the different views of the ledger and the deviation from the protocol rules by some participants, participants will eventually agree on the same transaction history and they are able to add transactions to the ledger.

Unfortunately, the Bitcoin protocol has low performance compared to popular

means of payment such as Visa. On average, one block is produced every 10 minutes regardless of the number of the participants, and in order for a transaction inside a block to be considered confirmed, the block should be buried by 6 blocks in the chain. This is equivalent to a throughput of approximately 7 transactions per second, one hour latency and no scalability. This performance is not sufficient to support current needs from electronic payment systems.

In terms of security, theoretical security analysis has shown that as long as the participants that follow the protocol hold roughly more computational power than the number of participants that deviate, confirmed transactions are permanent and new transactions added will eventually become confirmed. However, it has been shown that the rate in which blocks are produced cannot be significantly increased without serious loss in security [25].

Many modifications of the Bitcoin protocol have been proposed that aim to allow the block creation rate to be increased and avoid security tradeoffs. A family of them uses either alternative chain selection rules, such as the GHOST protocol [7], or use a Directed Acyclic Graph of blocks (blockDAG) instead of a blockchain [10, 34, 19]. Regarding the Bitcoin protocol, the reason that the block creation rate cannot be increased is that the percentage of the adversarial power that it can tolerate is inversely proportional to the network delay [25], and it should be ensured that when a block is mined it should have fully propagated before the next block is mined [4]. The aim of these protocols is to make the adversarial power tolerance independent of the network delay, so that the block creation rate can be increased without loss in security.

1.2 Thesis outline

In this thesis we present the protocols that in order to improve on performance diverge from the Bitcoin protocol by either using alternative chain selection rules or using blockDAGs as their underlying data structure. We focus on the security guarantees and techniques used in security proofs of these protocols.

In Chapter 2 we give an overview of the preliminary cryptographic tools that are commonly used in blockchain protocols.

In Chapter 3, we present the Bitcoin protocol and some of the arguments and proofs for its security since its appearance, starting from the synchronous setting in which network delay is abstracted from the model [4] and finally presenting a recent tight bound for the adversarial power tolerance in the setting where network delay is present and upper-bounded by a constant [25].

In Chapter 4 we describe the GHOST rule [7]. The GHOST rule was designed to avoid the secret mining attack that affects the Bitcoin protocol, but it was later found that a balance attack applies [9], which we describe. Furthermore, we discuss the methodology used and bounds obtained in the security analyses of GHOST. Finally, we present an attempt for an alternative proof for an already proven bound on the adversarial power that GHOST can tolerate using techniques present in the analyses of Bitcoin.

In Chapter 5 we present the PHANTOM and GHOSTDAG protocols [34], which are protocols that use a blockDAG as their underlying data structure. The PHANTOM protocol uses an NP-hard problem to distinguish between honest and adversarial blocks, and GHOSTDAG uses an approximation algorithm for the same problem. The GHOSTDAG protocol is presented by its creators as a fix of the GHOST protocol [34]. We also present the security guarantees of GHOSTDAG.

In Chapter 6 we present two other protocols that use a blockDAG as their underlying data structure, SPECTRE [10] and Conflux [29], and we discuss their security guarantees.

Finally, in Chapter 7 we discuss the security of the protocols that we have presented and we propose directions for further research.

CHAPTER 2

BACKGROUND

In this chapter we give an overview of the cryptographic tools that are commonly used in blockchain protocols.

2.1 Negligible functions

A function $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is called *negligible* if f tends to 0 as $n \to \infty$, but faster than the inverse of any polynomial. Formally:

Definition 2.1 (negligible function [24])**.** A function $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is called *negligible* if for all $c \in \mathbb{R}_{>0}$ there exists $n_0 \in \mathbb{Z}_{\geq 1}$ such that for every $n \geq n_0$, we have $|f(n)| < \frac{1}{n^c}$.

2.2 Hash functions

A hash function is a function that takes as input a string of arbitrary length and compresses it into a shorter string. Given an input for the hash function, its value should be computable in polynomial time. More formally:

Definition 2.2 (hash function-syntax [26])**.** A *hash function* is a pair of probabilistic polynomial-time algorithms (Gen, H) that satisfy the following properties.

1. Gen is a probabilistic algorithm which takes as input a security parameter 1^n and outputs a key s.

2. There exists a polynomial l such that H is a deterministic polynomial time algorithm that takes as input a key s and any string $x \in \{0,1\}^*$ and outputs a string $H^s(x) \in \{0,1\}^{l(n)}$.

If for every n and s the hash function H^s is defined only over inputs of length $l(n)$ and $l'(n)$ where $l'(n) > l(n)$, then we say that (Gen, H) is a *fixed-length hash function* with length parameter l'.

Hash functions used in cryptography are often called *cryptographic hash functions*.

A *collision* in a hash function H is a pair of values x and x', such that $x \neq x'$ and $H(x) = H(x')$. A hash function is said to be *collision resistant* if it is infeasible for a probabilistic polynomial time algorithm to find a collision. For a hash function $\Pi = (\mathsf{Gen}, H)$, the experiment for finding a collision is presented in Algorithm 1. Using Algorithm 1, the formal definition of a collision resistant hash function is given in Definition 2.3.

Algorithm 1 The collision-finding experiment [26].

1: **function** $\mathsf{Hash} - \mathsf{Col}_{\Pi,\mathcal{A}}(\mathrm{n})$
2: A key s is chosen.
3: The adversary $\mathcal{A}$ is given s and outputs a pair of values x and x'. Formally, $(x, x') \leftarrow \mathcal{A}(s)$.
4: The output of the experiment is 1 if and only if $x' \neq x$ and $H^s(x) = H^s(x')$. In such a case we say that $\mathcal{A}$ has found a collision.
5: **end function**

Definition 2.3 (collision resistant hash function [26])**.** A hash function $\Pi = (\mathsf{Gen}, H)$ is *collision resistant* if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function $negl$ such that

$$Pr[\mathsf{Hash} - \mathsf{Col}_{\Pi,\mathcal{A}}(n) = 1] \leq negl(n).$$

An example of a cryptographic hash function widely used in practice as a collision resistant hash function is $\mathsf{SHA256}$ [26].

Collision resistance is a strong security requirement, and in some applications weaker notions of security suffice. There are usually three notions of security for hash functions.

1. *Collision resistance* (Definition 2.3).
2. *Second preimage resistance*. Informally, a hash function is *second preimage resistant* if given a key s and a string x it is hard for a probabilistic polynomial-time adversary to find a string x' such that $H^s(x) = H^s(x')$.
3. *Preimage resistance*. Informally, a hash function is *preimage resistant* if given a key s and some value y it is hard for a probabilistic polynomial-time adversary to find a value x' such that $H^s(x') = y$.

The formal definitions for the second preimage resistance and preimage resistance properties are similar to Definition 2.3, by defining similar experiments as the experiment of Algorithm 1. It is important to note that a hash function that is collision resistant is also second preimage resistant, and a hash function that is second preimage resistant is also preimage resistant.

2.3 The random oracle model

A common methodology used in the design and analysis of protocols which use hash functions is the *random oracle model*. The random oracle model assumes the existence of a public, randomly chosen function H that can be evaluated only by querying an

oracle, that can be thought as a "magic box", and on input x, returns $H(x)$. The random oracle model methodology consists of the following two steps [26].

1. The scheme is designed and proven secure using the assumption that the world contains a random oracle.
2. When the scheme is implemented in practice, the random oracle is instantiated with a cryptographic hash function $\hat{H}$. At each point where the scheme dictates that a party should use the random oracle H, the hash function $\hat{H}$ is used instead.

The purpose of this methodology is to abstract the details of the specific hash function used in practice, with the hope that the security proof given in the first step will carry over to the real world implementation.

2.4 Merkle trees

Consider n items, $x_1, \ldots, x_n$ which belong to a space $\mathcal{X}$, for which we wish to compute a short hash and later be able to validate each item. One solution to this problem is to hash every item using a hash function h and store the tuple $(h(x_1), \ldots, h(x_n))$. This approach uses linear space. Merkle trees provide a space-efficient solution to this problem.

Assume that n is a power of 2, if n is not a power of 2 we can extend the tuple $(x_1, \ldots, x_n)$ with dummy elements to the closest power of 2. The definition of a Merkle tree hash is given in Definition 2.4.

Definition 2.4 (Merkle tree hash)**.** Given a hash function $h : \mathcal{X} \to \mathcal{Y}$ and an integer n that is a power of 2, the Merkle tree hash derived from h is a hash function $H : \mathcal{X}^n \to \mathcal{Y}$ which, given as input a tuple $(x_1, \ldots, x_n) \in \mathcal{X}^n$, its value is the output of Algorithm 2.

Algorithm 2 Merkle tree hash calculation [24].

Input: $x_1, \ldots, x_n \in \mathcal{X}$, where n is a power of 2
Output: $y \in \mathcal{Y}$
1: **for** $i = 1$ to n **do**
2: $\quad y_i \leftarrow h(x_i)$
3: **end for**
4: **for** $i = 1$ to $n - 1$ **do**
5: $\quad y_{i+n} \leftarrow h(y_{2i-1} || y_{2i})$
6: **end for**
7: **return** y_{2n-1}

It can be proven that if a hash function h is collision resistant, then the Merkle tree hash induced by h is also collision resistant.

An example of a Merkle tree is given in Figure 2.1

Given a Merkle tree hash value $y = H(x_1, \ldots, x_n)$, we can efficiently prove that a $x \in \mathcal{X}$ is in the position i in the tuple $T = (x_1, \ldots, x_n)$ as follows. One outputs as the proof π all the intermediate hashes that are the siblings of nodes on the path from the leaf number i to the root of the tree. This proof π contains exactly $\log_2 n$ elements in $\mathcal{Y}$. For example, to prove membership of the element x_3 in Figure 2.1, one provides the proof $\pi = (y_4, y_0, y_{14})$. To verify, one uses π to compute the Merkle root $\hat{y}_{15}$, and accepts if $y = \hat{y}_{15}$.

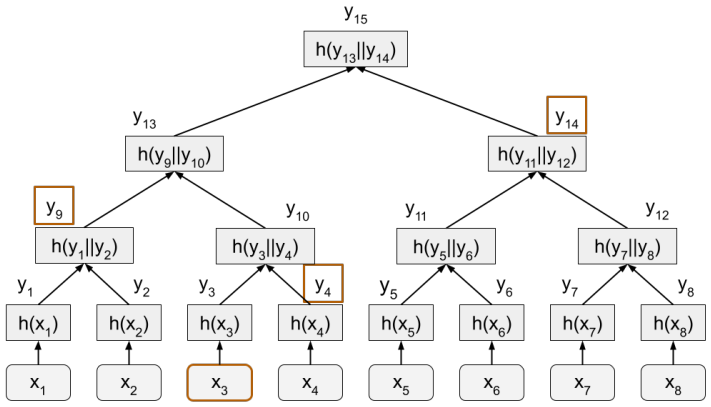


Figure 2.1: A Merkle tree with eight leaves. To prove membership of x_3, one provides the elements highlighted in orange.

CHAPTER 3

THE BITCOIN PROTOCOL

3.1 Introduction

The Bitcoin protocol was proposed in 2008 by a person or a group of people called Satoshi Nakamoto as a payment system. Participants transact using Bitcoin, a currency native to the protocol. The Bitcoin protocol is decentralized, meaning that multiple entities run the protocol and maintain the transaction database, and permissionless, allowing participants to participate in the protocol and transact with each other without the need for a trusted third party, as for example a bank.

The Bitcoin protocol is the first decentralized and permissionless protocol that was implemented and used for payments. After its appearance many other decentralized and permissionless payment protocols have been created, aiming to improve on performance [8, 34, 8, 20], anonimity [41], or support more complex transactions [38]. All of them share common mechanisms with the Bitcoin protocol.

In this Chapter we give a description of the mechanism of the Bitcoin protocol and we discuss its security, with the aim of understanding how security is proven and how it is shown from the security proofs that the block creation rate cannot be increased.

3.2 Protocol details

3.2.1 Blocks

In the Bitcoin protocol, transactions are broadcasted, collected, and organized into *blocks*. A block is a data structure that consist of two main parts: a list of transactions and a header. It may also contain other values such as the software version or a timestamp [37].

The first block ever created is called the *genesis block*, which is hardcoded in the Bitcoin software. Every other block is created via a procedure called *mining*.

Block Headers

Every block header, except from the block header of the genesis block, consists of three main parts.

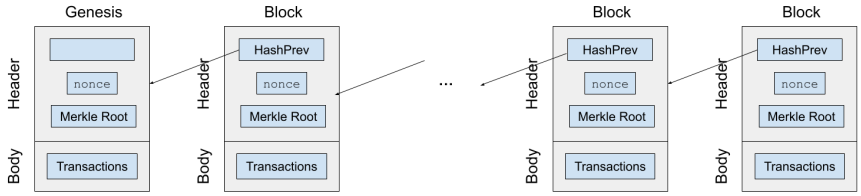


Figure 3.1: A high level overview of a blockchain.

1. The Merkle tree hash of the transactions of the block, which ensures the immutability of the transactions inside the block.

2. The hash of the previous block header using the SHA256 hash function [2].

3. A value nonce, which ensures that enough computational effort has been made in order for the block to be accepted by the network.

The block header of the genesis block does not contain the hash of some other block header.

Since in a block header the hash of some other block header is included, blocks are linked. A chain of blocks is called a *blockchain*. The genesis block is the root of the blockchain. By construction, a blockchain is a timely ordered sequence of blocks: a block has necessarily been created after its predecessor.

A high level overview of a blockchain appears in Figure 3.1.

Mining

Mining is the procedure in which new blocks are created, and participants that attempt to create new blocks are called *miners*[1] In order for a block to be considered valid and accepted by the network, the miner should find a *proof-of-work* for the block. The proof-of-work is a value for the nonce such that when the string containing the data of the block header is hashed using the SHA256 hash function, the hash value is lower than a threshold **T**. Proof-of-work ensures that enough computational power has been spent, since SHA256 is used as a collision resistant hash function. Upon successful mining of a block, the block is broadcasted to the network by the miner.

The Proof-of-Work mechanism is the mechanism that enables the Bitcoin protocol to be a permissionless distributed ledger protocol. Anyone can join or leave the protocol and create as many identities as they wish, since the rate in which a participant can produce blocks depends on their computational (CPU) power and not on the number of identities they possess.

.

The threshold **T** is set so that 1 block per 10 minutes is produced [2] by the network. The reason for the choice of rate is to ensure that a block has propagated to the whole network before a new blocks is mined. Since the amount of time needed by the network to mine a new block depends on the network's total computational power, the threshold

[1] The reason for being called miners is that when a new block is mined, new Bitcoins are created and sent to the address of the miner as a reward for the resources they have spent. For the rest of this work we will use the terms *participant*, *player* and *party* interchangeably to refer to a miner.

T is recalculated every 2016 blocks [37] to preserve the 1 block per 10 minutes rate. This rate is equivalent to 7 transactions per second [28], which is a low rate compared to Visa that can support 65,000 transactions per second[2], and potentially cannot serve current needs. For this reason, debates on the block size of the Bitcoin protocol and variants with larger block size have emerged [22].

A mathematical definition of a block is given below.

Definition 3.1 (block [4])**.** Let $G(\cdot), H(\cdot)$ be cryptographic hash functions with output in $\{0,1\}^\kappa$. A block is any triple of the form $B = \langle s, x, ctr \rangle$ where $s \in \{0,1\}^\kappa, x \in \{0,1\}^*, ctr \in \mathbb{N}$ are such that satisfy predicate $validblock^q_{\mathbf{T}}(B)$ defined as

$$(H(ctr, G(s,x)) < \mathbf{T}) \wedge (ctr \leq q).$$

3.2.2 Forks and the longest chain rule

A *fork* is the result of two distinct blocks pointing to the same block. Forks are created because of two main reasons: network delay and adversarial behavior, as illustrated in Figure 3.2. When forks appear, conflicting transactions may be included in the different branches of the tree and it is necessary that all miners include only one and the same one in their local ledger.

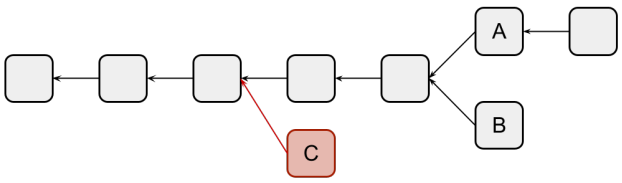


Figure 3.2: Forks in the Bitcoin protocol. Blocks A and B are created from honest miners, who due to network delay did not receive each other's block when mining. Block C is created by the adversary who may be aware of more recent blocks than the block C is pointing to, but chose to create a fork.

The way to resolve forks is by using the *longest chain rule*: the valid blockchain is the longest one and blocks that are not in the longest chain are ignored. Furthermore, when miners create new blocks, their blocks should point to the longest chain of blocks the miner observes and thus increasing its length.

3.3 Security of the Bitcoin protocol

3.3.1 Nakamoto's calculations

In the Bitcoin Whitepaper [2], the security of the Bitcoin protocol was discussed by considering the scenario of an attacker performing a *secret mining attack*[3]. In this attack, the adversary publishes a transaction in the honest chain and starts secretly mining an

[2] source: visa fact sheet `https://www.visa.co.uk/dam/VCOM/download/corporate/media/visanet-technology/aboutvisafactsheet.pdf`

[3] In the Bitcoin Whitepaper [2] this attack is called a *double spending* attack. We chose to call it a *secret mining* attack since as we will see in Chapter 4, there are other attacks that result in double spending, but the strategy of the attack is to secretly mine a chain.

alternative chain that contains a conflicting transaction. After the recipient of the transaction in the honest chain accepts it, the adversary broadcasts their secretly mined chain to the network. If the adversarial chain is longer than the honest chain, the adversary is able to reverse the transaction and the attack succeeds.

Using the following arguments, it was concluded that if the honest parties possess more computational power than the attacker, the attack will not succeed. As we will see in the following sections, this argument is not correct when network delay is present. In this case an attacker with less than half of the total computational power can succeed in the secret mining attack, and actually the network delay works in their favor.

The race between the honest and the adversarial chain is modelled as a Simple Random Walk [39]. The success event $(+1)$ is the event that the honest parties mine the next block and thus the difference in the length of the two chains increases by one, and the failure event (-1) is the event that the adversary mines the next block and thus the difference decreases by one.

If p, q are the probabilities that an honest party and the attacker find the next block, respectively, then similarly to the "Monkey falls of the cliff" problem [1], the probability q_m that the attacker will ever create a longer chain than the honest parties if the honest chain is m blocks longer than the attacker chain is

$$q_m = \begin{cases} 1 & \text{if } p \leq q \\ \left(\frac{p}{q}\right)^m & \text{if } p > q \end{cases}$$

The seller accepts the transaction when the honest chain has been extended by k blocks from the block containing the transaction. Assuming that these k blocks took the average time per block to be mined, the number of blocks the attacker will have mined on his secret chain in the same time follows a Poisson distribution with expected value $\lambda = k\frac{p}{q}$. Then, the probability that the attacker can succeed in the attack is calculated as follows.

$$\sum_{m=0}^{\infty} Pr[\text{attacker mines } m \text{ blocks}] \cdot Pr[\text{ attacker can beat } (k-m) \text{ difference in length}]$$

The mathematical equation is Equation 3.1.

$$\sum_{m=0}^{\infty} \frac{\lambda^m e^{-\lambda}}{m!} f_k(m) \tag{3.1}$$

where

$$f_k(m) = \begin{cases} \left(\frac{q}{p}\right)^{(k-m)} & \text{if } m \leq k \\ 1 & \text{if } m > k \end{cases}$$

Then, by calculating the values of Equation 3.1 for several values of q and k, it was concluded that the probability that the attacker will succeed drops exponentially as k increases.

3.3.2 Bitcoin in the synchronous setting

The first security analysis of the Bitcoin protocol under any adversarial strategy was by Garay et al. [4]. The protocol used to distributively form and maintain the the

blockchain was separated from the application, which is using the blockchain to securely store transactions. The protocol was named *Bitcoin Backbone*, and the application was named a *transaction ledger*. A model and a security proof of the Bitcoin backbone protocol under any adversarial strategy was given, showing that it satisfies two fundamental properties, *Common Prefix* and *Chain Quality*. Furthermore, it was shown that under the *Honest Majority Assumption*, the Bitcoin Backbone protocol can be used to implement a robust transaction ledger, where robustness means that it satisfies *Persistence* and *Liveness*.

In the analysis, a static number n of participants are assumed, among which t are adversarial. The hash function is modelled as a random oracle with a probability of success p for each query. Communication was assumed *syncrhonous*, in the following sense. Time is divided in rounds. At the beginning of each round, each party receives the longest chains broadcasted by other parties in the previous round and potentially updates its local longest chain. Then, it attempts to mine a block, having q queries to the random oracle, and if successful before the q queries are exhausted, updates its local chain with the new block. At the end of the round, every honest party broadcasts to all parties its local longest chain. The adversary has the ability to send different blocks to different parties, however, they cannot alter the content or prevent the delivery of messages sent by honest parties. Since each party has q queries in each round, each one successful with probability p, proof-of-work mining is modelled as a Binomial random variable. The security of the Bitcoin protocol depends on the number of *uniquely successful rounds*, namely, rounds in which exactly one honest block is mined.

Using the notation of table 3.1, we give the definitions of honest majority assumption, Common Prefix, and Chain Growth properties below.

p	probability that a query to the random oracle is successful
λ	tail-bounds parameter
n	total number of parties mining
t	number of parties controlled by the adversary
q	number of queries each party has to the random oracle
δ	advantage of honest parties, $(t/(n-t) \leq 1-\delta)$
α	probability at least one honest party succeeds in finding a POW in a round $\alpha = 1-(1-p)^{q(n-t)}$
ϵ	quality of concentration of random variables in typical executions, cf. Definition 3.9
k	number of blocks for the common prefix property
l	number of blocks for the chain quality property
μ	chain quality parameter
s	number of rounds for the chain growth property
τ	chain growth parameter

Table 3.1: Notation

Definition 3.2 (Honest Majority Assumption [4])**.** A number of t out of n parties are corrupted such that $t \leq (1-\delta)(n-t)$, where $3\alpha + 3\epsilon < \delta \leq 1$

Definition 3.3 (Common Prefix [4], simplified form)**.** The common prefix property Q_{cp} with parameter $k \in \mathbb{N}$ states that for any pair of honest parties P_1, P_2 adopting the chains $\mathcal{C}_1, \mathcal{C}_2$ at rounds $r_1 \leq r_2$ respectively, it holds that $\mathcal{C}_1^{\lceil k}$ [4] is a prefix of $\mathcal{C}_2$.

[4] $\mathcal{C}^{\lceil k}$ is the chain resulting from $\mathcal{C}$ by removing the last k blocks.

Definition 3.4 (Chain Quality [4], simplified form)**.** The chain quality property Q_{cq} with parameters $\mu \in \mathbb{R}$ and $l \in \mathbb{N}$ states that for any honest party P with chain $\mathcal{C}$, it holds that for any l consecutive blocks of $\mathcal{C}$, the ratio of honest blocks is at least μ.

In a subsequent work by Kiayias and Panagiotakos [5] another fundamental property of the Bitcoin backbone was identified, the *Chain Growth* property. The Chain Growth property was later stated explicitly into the analysis of Garay et al.[4].

Definition 3.5 (Chain Growth [4], simplified form)**.** The chain growth property Qcg with parameters $\tau \in \mathbb{R}$ and $s \in \mathbb{N}$ states that for any honest party P that has a chain $\mathcal{C}$, it holds that after any s consecutive rounds it adopts a chain that is at least $\tau \cdot s$ blocks longer than $\mathcal{C}$.

The three properties together are sufficient to prove in a black box manner that Nakamoto's protocol can be used to implement a transaction ledger that satisfies Persistence and Liveness. The Persistence property ensures that once a transaction is included in a block of the longest chain which is buried by k blocks, it will remain in the same position in the ledger. The Liveness property ensures that if honest parties attempt to include a transaction in the ledger, the transaction will be eventually added and will eventually be confirmed.

Definition 3.6 (Persistence [4])**.** Parameterized by $k \in \mathbb{N}$ (the "depth" parameter), if in a certain round an honest party reports a ledger that contains a transaction tx in a block more than k blocks away from the end of the ledger, then tx will always be reported in the same position in the ledger by any honest party from this round on.

Definition 3.7 (Liveness [4])**.** Parameterized by $u, k \in \mathbb{N}$ (the "wait time" and "depth" parameters, resp.), provided that a valid transaction is given as input to all honest parties continuously for u consecutive rounds, then there exists an honest party who will report this transaction at a block more than k blocks from the end of the ledger.

More specifically, the Common Prefix property is sufficient to prove Persistence, and the Chain Growth and Chain Quality properties is sufficient to prove Liveness. Intuitively, it is straightforward why the Common Prefix property implies Persistence. Regarding the Liveness property, the Chain Growth property ensures that new blocks are added to the chains of honest parties, and the Chain Quality property ensures that some of the blocks that are added are honest, thus eventually if honest participants attempt to include a transaction in the ledger, the transaction will be added and confirmed.

It is important to note that the technique of using the Common Prefix, Chain Growth and Chain Quality properties to prove Persistence and Liveness was later used in the security proofs of the family of Ouroboros protocols [12, 16, 15] and the Prism protocol [20].

The security of the Bitcoin protocol depends on *uniquely successful rounds*. A round is called uniquely successful if in this round exactly one honest party succeeds in mining a block. Another type of rounds defined are the *successful rounds*. A round is called successful if at least one honest party succeeds in mining a block. The following random variables are introduced to capture whether rounds are successful or uniquely successful, and also to count the number of adversarial blocks mined in one round or in a set of rounds.

Definition 3.8 ([4])**.** For a round i, let

- Random variable X_i be equal to 1, if round i is successful, and 0 otherwise.

- Random variable Y_i be equal to 1, if round i is uniquely successful, and 0 otherwise.
- Random variable Z_i be equal to the number of blocks the adversary mines in round i, formally defined as the sum of the successful queries in round i of all the parties controlled by the adversary.

Furthermore, for a set of rounds S, let $X(S) = \sum_{i \in S} X_i$ and $Y(S)$ and $Z(S)$ defined similarly.

To explain the results of Garay et al.[4], the definition of a *typical execution* is needed. A typical execution is an execution of the protocol in which during a sufficient number of consecutive rounds, inversely proportional to the probability that a round is successful, the random variables of Definition 3.8 are close to expectation, and the number of adversarial blocks is less than the number of successful rounds. Using the notation of Table 3.1, a typical execution is defined as follows.

Definition 3.9 (typical execution, [4])**.** An execution is (ϵ, λ)-typical (or just typical), for $\epsilon \in (0, 1)$ and integer $\lambda \geq \frac{2}{\alpha}$, if, for any set S of at least λ consecutive rounds, the following hold.

(a) $(1 - \epsilon)E[X(S)] < X(S) < (1 + \epsilon)E[X(S)]$ and $(1 - \epsilon)E[Y(S)] < Y(S)$

(b) $Z(S) < E[Z(S)] + \epsilon E[X(S)]$

(c) No insertions, no copies, and no predictions occured, where

- an *insertion* occurs when, given a chain $\mathcal{C}$ with two consecutive blocks B_1 and B_2, a block B_3 created after B_2 is such that B_1, B_3, B_2 form three consecutive blocks of a valid chain,
- a *copy* occurs if the same block exists in two different positions, and
- a prediction occurs when a block extends one which was computed at a later round.

Typical executions are the most likely to happen.

Theorem 3.10 ([4])**.** *An execution is typical with overwhelming probability.*

Furthermore, in a typical execution, for any set S of at least λ consecutive rounds, it holds that

$$Y(S) > Z(S). \tag{3.2}$$

That is, during a sufficient amount of consecutive rounds, the number of those that are uniquely successful is greater than the number of adversarial blocks produced during these rounds. Equation 3.2 is the core equation used in the proofs of Persistence and Liveness stated below, showing the dependency of the security of the Bitcoin protocol to uniquely successful rounds.

Lemma 3.11 (Persistence [4], simplified form)**.** *Under the Honest Majority Assumption, it holds that the transaction ledger induced by Nakamoto's protocol satisfies Persistence (cf. Definition 3.6) with parameter $k = \lceil 2\lambda\alpha \rceil$ with probability at least $1 - e^{-\Omega(\epsilon^2 \lambda \alpha)}$.*

Lemma 3.12 (Liveness [4], simplified form)**.** *Under the Honest Majority Assumption, it holds thatthe transaction ledger induced by Nakamoto's protocol satisfies Liveness (cf. Definition 3.7) with parameters* $u = \lceil \frac{4\lambda}{1-\epsilon} \rceil$ *and* $k = \lceil 2\lambda\alpha \rceil$*, with probability at least* $1 - e^{-\Omega(\epsilon^2 \lambda \alpha)}$.

Finally, Lemmas 3.11 and 3.12 hold under the Honest Majority assumption. The relation $3\alpha + 3\epsilon < \delta \leq 1$ of Defintion 3.2 shows the relation between the proof-of-work difficulty and the adversarial power tolerance of the protocol. If the probability p that a query to the random oracle is successful is small, the probability α that at least one honest party succeeds in finding a proof-of-work in a round increases, and thus the lower bound on δ increases, making the protocol resistant only to smaller adversarial fractions.

3.3.3 Bitcoin in the bounded network delay setting

Dependence of adversarial power tolerance to network delay

There are two ways in which the Bitcoin protocol could increase its throughput: either increasing the block creation rate, or increasing the block size. However, both of these approaches result in a decrease in the adversarial power the Bitcoin protocol can tolerate securely. That is because the adversarial power tolerance of the Bitcoin protocol is dependent on the network delay [7].

First, if the block creation rate increases, an adversary with less than 50% of the total computational power can succeed in the secret mining attack. When the block creation rate is increased and network delay is present, more forks will be created, since honest miners that have not yet received the last addition to the chain will contribute to a less updated chain. Thus, more blocks are created, but they are forming forks instead of contributing to the longest chain. If the adversary is assumed to suffer from little or no delay, he can secretly mine a longer chain than the honest parties while having less computational power, and thus the secret mining attack will succeed (see Figure 3.3).

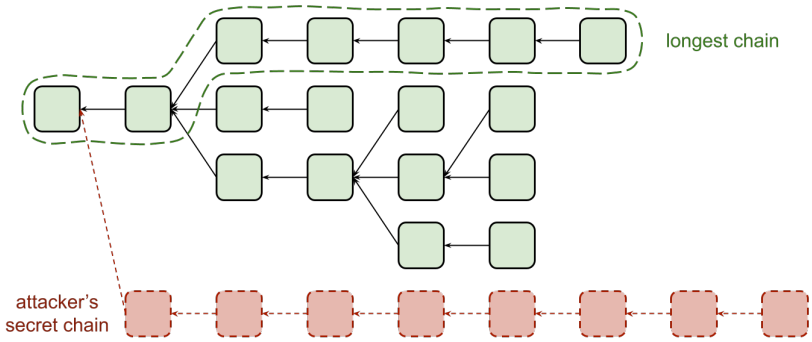


Figure 3.3: Bitcoin under high block creation rate. Blue blocks are honest blocks and red dashed blocks form the attacker's secretly mined chain. Although the attacker has less computational power than the honest parties, since he suffers from no delay he is able to secretly mine a longer chain than the honest parties.

Secondly, increasing the block size also leads to decrease in security. Using data provided by Decker and Watenhoffer [3], Sompolinksy and Zohar [7] observed that an increase in block size leads to a linear increase in delay (Figure 3.4). This means that if the block size is increased, again more forks will be created and the secret mining attack will succeed. Thus, in order to preserve security, when increasing block creation rates, the block size should be lowered so that the delay reduces, and when increasing the block size the block creation rate should be lowered.

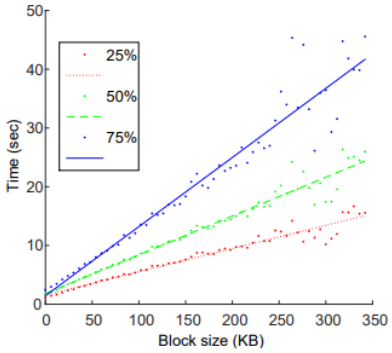


Figure 3.4: The relation between the block size and the time it took to reach 25% (red), 50% (green), and 75% (blue) of monitored network nodes. Figure taken from Sompolinksy and Zohar [7].

In conclusion, increasing the block size and increasing the block creation rate can be seen as equivalent approaches, and none of them solves the performance issue of Bitcoin. This is because the adversarial power tolerance of the Bitcoin protocol is dependent on the rate in which the longest chain grows, which is in turn dependent on the network delay. This means that if the adversary manages to mine blocks faster than the rate in which the longest chain grows, they can succeed in the secret mining attack. Namely, if τ is the rate in which the longest chain grows, and $q \cdot \lambda_h$ is the computational power of the adversary, the *security threshold* of a chain protocol is defined as $\frac{\tau}{q \cdot \lambda_h}$ [7]. As a result, if we want the Bitcoin protocol to tolerate an adversary that possesses a fraction of the computational power close to 50%, the block creation rate should be kept low enough so that the chain of honest parties' grows in a rate close to the rate the honest blocks are produced.

Discrete time analysis

Pass et al. [14] analyzed the security of the Bitcoin protocol in the random oracle model, in a setting where network delay is controlled by the adversary and upper bounded by a constant $\Delta > 0$. As in the analysis of Garay et al. [4], time is divided in rounds, however, in this analysis, the arrival of a block mined in a round can be delayed for up to Δ rounds. Under this model, a bound on the adversarial power tolerance of the Bitcoin protocol parameterized by the upper bound on the network delay Δ is given. It is also shown that the security of the Bitcoin protocol breaks in a setting where network delay is unbounded.

For a transaction ledger to be robust, it should satisfy Persistence and Liveness, similarly to the analysis of Garay et al[4].

Definition 3.13 (Persistence [14])**.** The persistence property stipulates that if some honest party P outputs a message m at position i in its local ledger, then

1. message m is the only message that can ever be output at position i of any other honest party's ledger, and

2. every other honest party will eventually output m at position i.

Definition 3.14 (Liveness [14])**.** The liveness property stipulates that from any given round r, if a sufficiently long period of time t elapses-we refer to this time as the wait-time of the ledger-every honest party will output a message m as part of their (local) ledger, where m was provided as an input to some honest party between rounds r and $r + t$.

In order for a transaction ledger which is built on top of a blockchain protocol to satisfy Persistence and Liveness, it is sufficient for the blockchain protocol to satisfy the following properties.

Definition 3.15 ([14])**.** A secure blockchain protocol should satisfy the following properties.

(a) *consistency*: At any point, the chains of two honest parties can differ only in the last k blocks, with overwhelming probability in k.

(b) *future self-consistence*: At any two rounds r, s the chains of any honest party at r and s differ only within the last k blocks, with overwhelming probability in k.

(c) τ*-chain growth*: Parameterized by $\tau \in \mathbb{R}$, at any point in the execution, the chain of honest parties grows by at least k blocks in the last $\frac{k}{\tau}$ rounds with overwhelming probability in k; τ is called the chain-growth of the protocol.

(d) μ*-chain quality*: Parameterized by $\mu \in \mathbb{R}$, for any k consecutive blocks in any chain held by some honest party, the fraction of blocks that were contributed by honest parties is at least μ, with overwhelming probability in k.

If the adversarial power tolerance is appropriately bounded, the Bitcoin protocol is a secure blockchain protocol as in Definition 3.15. Using the notation of Table 3.2, the result is described in the following theorem.

n	total number of parties
ρ	fraction of parties that are controlled by the adversary
p	probability that a query to the random oracle is successful
α	probability that some honest miner finds a block in one round, $\alpha = 1 - (1-p)^{(1-\rho)n}$
β	expected number of blocks mined by the adversary in one round $\beta = \rho np$
γ	discounted version of α due to network delay $\gamma = \frac{\alpha}{1+\Delta\alpha}$

Table 3.2: Notation

Theorem 3.16 ([14]). *Assume there exists some $\delta > 0$ such that*

$$\alpha(1 - (2\Delta + 1)\alpha) \geq (1 + \delta)\beta \tag{3.3}$$

Let $\tau = \frac{\gamma}{1+\delta}$ and $\mu = 1 - (1 + \delta)\frac{\beta}{\gamma}$. Then Nakamoto's protocol satisfies consistency, future self consistency, μ-chain quality and τ-chain growth.

Equation 3.3 in Theorem 3.16 gives an upper bound for the adversarial power the Bitcoin protocol can tolerate securely, parameterized both by the computational power of the honest parties and the network delay. Furthermore, the network delay affects the parameters of the chain growth and chain quality properties, since both are dependent on γ, which is a discounted version of α due to network delay.

It is proven that every blockchain protocol that satisfies the properties of Definition 3.15 can be used to maintain a public ledger that satisfies Persistence and Liveness, as in the following theorem.

Theorem 3.17 ([14], simplified form). *Let $R(\cdot)$ be a strictly positive, super-constant polynomial and suppose that a blockchain protocol satisfies consistency, τ-chain growth and μ-chain quality, where μ and τ are strictly positive. Then, for every $\delta > 0$, the public ledger $R(\cdot)$-induced by the blockchain protocol [5] satisfies persistence and liveness with wait-time $t = (1 + \delta)\frac{R(\kappa)}{\tau}$.*

Observe that the chain-quality parameter μ does not appear as a parameter of the wait time. The reason is that it is sufficient that μ is strictly positive [14].

It is also shown that as long as the network delay is bounded by a constant $\Delta > 0$, even if Δ is large, we can lower the mining rate and achieve security, as in the following corollary.

Corollary 3.18 ([14], simplified form). *Assume $\rho < \frac{1}{2}$. Then for every n, Δ, there exists some sufficiently small $p_0 = \Theta\left(\frac{1}{\Delta n}\right)$ such that Nakamoto's protocol with mining parameter $p \leq p_0$ satisfies consistency, future self consistency, $\left(1 - \frac{1}{1-\rho}\right)$-chain quality and $\frac{pn}{2}$-chain growth.*

In Corollary 3.18 the upper bound p_0 on p is inversely proportional to the network delay Δ and the number of participants n. This means that to maintain security when having a large number of participants or a large network delay, p should be small, and thus have a small block creation rate.This in turn reduces the value of the chain growth parameter, and thus the throughput of the protocol decreases.

On the contrary, if the network delay is unbounded then the security of the Bitcoin protocol breaks.

Theorem 3.19 (Inconsistency of Nakamoto's protocol with Unbounded Delays [14], simplified form). *For every $0 < \delta < \frac{1}{2}$, $0 < \rho < 1$ and every inverse polynomial $p(\cdot)$, Nakamoto's protocol does not satisfy neither consistency nor liveness for $n = \frac{2}{\rho^2}$ and $\Delta = \frac{1+\delta}{\rho np}$.*

Kiffer et al. [18] also provided a bound for the adversarial power tolerance for the Persistence property (named *Consistency* in their work) of the Bitcoin protocol under network delays, adopting the model of Pass et al [14]. A neater form of this bound was

[5] Given a blockchain protocol, the public ledger $R(\kappa)$-induced by the blockchain protocol extracts the messages/transactions that are inside the blocks of the blockhain, truncates the last $R(\kappa)$ records of it, and outputs the results. ([14], simplified form)

Δ	upper bound on network delay
p	probability that a query to the random oracle is successful
n	total number of miners, each with identical computing power
ρ	fraction of corrupted miners
α	probability that at least one honest miner mines a block in one round, $\alpha = 1 - (1-p)^{(1-\rho)n}$
γ	probability that exactly one honest miner mines a block in one round, $\gamma = (1-\rho)pn(1-p)^{(1-\rho)n-1}$
β	expected number of blocks the adversary can mine in one round, $\beta = p\rho n$

Table 3.3: Notation

later given by Zhao et al. [31], demonstrating that the bound is superior to that of Pass et al.[14].

Using the notation of Table 3.3, the result of Zhao et al. is presented in Theorem 3.20.

Theorem 3.20 ([31])**.** *Nakamoto's blockchain protocol satisfies consistency if there exists a positive constant $\delta > 0$ such that*

$$(1-\alpha)^{2\Delta}\gamma \geq (1+\delta)\beta. \tag{3.4}$$

Continuous time analysis

Ren [23] also studied the Bitcoin protocol in a setting in which adversarial network delays are upper bounded by $\Delta > 0$. Instead of dividing time in rounds, he studied the security of the Bitcoin protocol in the continuous time setting, modelling mining as a Poisson process. The security proof presented was much simpler than previous works.

Considering that an honest party considers a block B *committed* if B is buried at least k blocks deep in its adopted chain, *safety* and *liveness* properties are defined as follows.

- *safety*[6]: Honest parties do not commit different blocks at the same height.
- *liveness*: Every transaction is eventually committed by all honest parties.

Let α be the collective mining rate of honest parties, β be the collective mining rate of adversarial parties and Δ be an upper bound on the network delay. Let also $g = e^{-\alpha\Delta}$; the term g can be thought as a discounted version of α due to network delay. The Safety and Liveness results are given in the following theorems.

Theorem 3.21 (Safety [23])**.** *Let $g = e^{-\alpha\Delta}$ and suppose $g^2\alpha > (1+\delta)\beta$. Consider any time t and any block B that is considered committed by some honest party at time t. Except for $e^{-\Omega(\delta^2 g^2 k)}$ probability, for all time $t' \geq t$, no honest party commits a block $B' \neq B$ at the height of B.*

Theorem 3.22 (Liveness [23])**.** *Let $g = e^{-\alpha\Delta}$ and suppose $g\alpha > (1+\delta)\beta$. At time t, except for $e^{-\Omega(\delta^2 g\alpha t)}$ probability, every honest party commits at least $\frac{\delta}{6}g\alpha t - k - 1$ honest blocks.*

[6]Similar to Persistence of Garay et al. [4] with a different name.

The relation $g^2\alpha > (1+\delta)\beta$ for the Safety property (Theorem 3.21) is the same bound as that of Zhao et al. (Theorem 3.20) expressed in the continuous-time setting [23].

Tight consistency bound

Lastly, a tight bound for the consistency property of the Bitcoin protocol was given by Gaži et al. [25]. In this work, proof-of-work mining is modelled as a Poisson process, and the discrete time approximation of a Poisson Process was used. If r_h, r_a is the expected number of honest and adversarial blocks in unit time, respectively, and Δ is an upper bound on the network delay, then the Bitcoin protocol is secure if

$$r_a < \frac{1}{\Delta + \frac{1}{r_h}} \tag{3.5}$$

Equation 3.5 is tight in the sense that if r_a exceeds the given threshold, the adversary can beat the rate in which the longest chain grows and the secret mining attack applies, but below the threshold the Bitcoin protocol is proven to be secure.

Concerning the bound on the proof-of-work difficulty, or equivalently the probability that a query to the random oracle is successful, the result is as follows. For a time interval of length L, divide it into $\frac{L}{s}$ sufficiently small intervals of length s each, such that at most one block can be mined in any interval. Let $p_h = r_h s$ and $p_a = r_a s$ be the probability that an honest and an adversarial party mines a block in an interval of length s, respectively. In this discretized setting, the Bitcoin protocol is secure if

$$p_a < \frac{1}{\Delta - 1 + \frac{1}{p_h}}$$

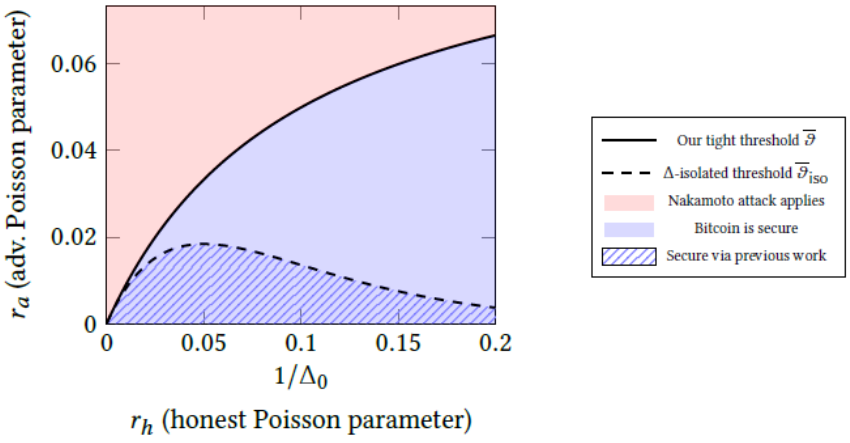


Figure 3.5: Figure taken from Gaži et al. [25]. The solid dashed line is the threshold of Equation 3.5 for $\Delta = \Delta_0 = 10$. The red area represents the pairs (r_a, r_h) for which the secret mining attack applies, and the blue area represents the pairs for which the Bitcoin protocol is secure. The dashed blue area represents the pairs for which the Bitcoin protocol was proven secure via previous work [23].

CHAPTER 4

THE GHOST PROTOCOL

4.1 Introduction

The GHOST protocol was proposed by Sompolinsky and Zohar [7] as a modification of the Bitcoin protocol. Instead of the longest chain selection rule, it uses the Greedy Heaviest Observed Sub Tree (GHOST) rule to select a chain of blocks. The intuition behind GHOST is that blocks that may not be in the final selected chain should be counted in the chain selection proccess. The GHOST protocol aimed to support higher throughput (transactions per second) than the longest chain rule by avoiding the secret mining attack discussed in Chapter 3. Even though it avoids the secret mining attack, a balance attack was found that makes the throughput of the GHOST protocol dependent on the network delay [13].

In this chapter we describe the GHOST rule, the balance attack and security proofs for GHOST. We also present our contribution, which is an alternative proof of security for GHOST.

4.2 Protocol Details

Contrary to the Bitcoin protocol in which miners only store the most recent longest chain of blocks they have received, in the GHOST protocol miners keep all valid blocks they receive and then use the GHOST rule to find the last block of the GHOST chain. Then, they attempt to mine a new block that points to the last block of the GHOST chain instead of the last block of the longest chain.

We will call the data structure formed by valid blocks a *blockTree*, since if blocks are considered as vertices and hash references as edges, the data structure has the form of a tree. We first give our definition of a blockTree. Next we give the definition of the weight of a block in a blockTree, and then we describe the GHOST rule in Algorithm 3.

Definition 4.1 (BlockTree)**.** A *blockTree* is a directed rooted tree $T = (V, E)$, where V represents blocks and E represents hash references.

Definition 4.2 (Weight of a subtree)**.** For a blockTree $T = (V, E)$ and $B \in V$, denote

by $T(B)$ the subtree of T rooted at B. The *weight* of $T(B)$, denoted $w(T(B))$, is the number of blocks in $T(B)$.

Algorithm 3 The GHOST rule[7].

Input: BlockTree $T = (V, E)$
Output: Last block of the GHOST chain of T
1: $B \leftarrow$ genesis block
2: **while** $children_T(B) \neq \emptyset$ **do**
3: $\quad B \leftarrow argmax_{B' \in children_T(B)} w(T(B'))$
4: **end while**
5: **return** B

As presented in Algorithm 3, the GHOST chain is calculated by starting from the genesis block and at each step choosing the block that has the biggest weight, until reaching a leaf. An example of the execution of Algorithm 3 in a blockTree is given in Figure 4.1.

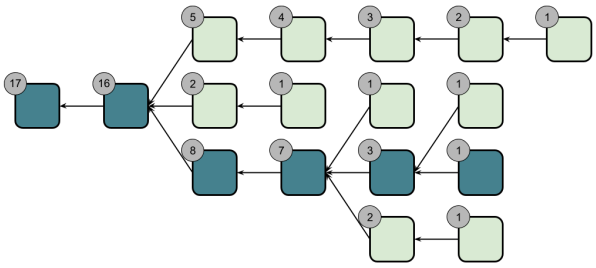


Figure 4.1: An example of the execution of Algorithm 3 in a blockTree. The algorithm returns the last block of the GHOST chain (blue blocks). The weight of each block appears in the top left circle in each block.

That is, the higher the weight of the subtree of a block, the bigger its chances that it will be selected. For a block in the GHOST protocol to be considered confirmed, it should be in the GHOST chain and its subtree should have a sufficient weight.

4.3 The Balance Attack

Although the GHOST protocol avoids the secret mining attack, it cannot support unlimited throughput when network delay is present. An attack called *balance attack* applies to GHOST and makes its security dependent on the network delay [13]. This attack also applies to Bitcoin.

Let $H = (V, E)$ be the graph of the communication network in which the GHOST protocol is executed. The adversary partitions V into sets $(V_1, \ldots, V_r)$ such that the computational power of network nodes in each V_i is almost the same. Let $H_i = H[V_i] = (V_i, E_i)$ be the subgraph of H induced by V_i. Let also $E_0 = E \setminus \bigcup_{i=1}^{r} \{E_i\}$.

If the adversary is able to disrupt communications in E_0 for a sufficient amount of time, he is able to double-spend.

The attack works as follows. The adversary disrupts communications in E_0 for Δ_0 seconds and sends a transaction tx crediting a merchant to V_i for some $i \in \{1, \ldots, r\}$. Let $view(H_i)$ be the set of blocks miners in V_i can receive. After sending tx, the adversary starts mining on top of a block B that appears in $view(H_j)$ for some $j \in \{1, \ldots, r\}$ but does not appear in $view(H_i)$, and broadcasts his blocks only in the nodes in V_j. When Δ_0 seconds have elapsed, he stops disrupting communications on E_0 and he can issue a transaction tx' that double-spends the coins of tx. If Δ_0 is big enough, the block containing tx will be considered committed by the merchant after time Δ_0 has elapsed.

Assuming for simplicity that $r = 2$, and assume that during time Δ_0, all honest miners mine blocks. Using the notation of Table 4.1, each set of miners in V_1 and V_2 performs $m = \frac{1-\rho}{2} f \Delta_0$ Bernoulli trials, each one with probability of success p. If X_i is the number of blocks mined by miners in $V_i, i = 1, 2$, then

$$\mu_h = E[X_i] = mp = \frac{1-\rho}{2} f \Delta_0 p.$$

Similarly, the expected number of blocks the adversary can mine in time Δ_0 is

$$\mu_a = \rho f \Delta_0 p.$$

f	total mining power of the system (number of Hashes tested per second)
p	probability that a hash is successful
ρ	fraction of mining power owned by the adversary
r	number of communication subgraphs
Δ_0	disruption time between communication subgraphs (in seconds)
μ_h	expected number of blocks mined by honest miners in V_i in a time interval of length Δ_0
μ_a	expected number of blocks mined by the adversary in a time interval of length Δ_0

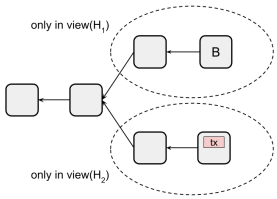
Table 4.1: Notation

The success of the balance attack relies on the adversary being able to mine more blocks than the difference between the number of blocks mined by the miners in V_1 and V_2, since then the blockTree branch that does not contain his transaction will be selected.

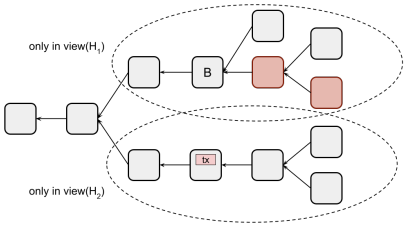
Theorem 4.3 ([9]). *If $\Delta_0 \geq \frac{(1-\rho)6log(\frac{4}{\varepsilon})}{\rho^2 fp}$, then after the communication is re-enabled, $Pr[\mu_a > |X_1 - X_2|] > 1 - \varepsilon$.*

Theorem 4.3 shows the relation between the proof-of-work hardness, the network delay and the adversarial power tolerance of the GHOST protocol. If the probability p that a hash is successful, the total mining power f, or the fraction of adversarial mining power increases, the lower bound on Δ_0 decreases, making it easier for the adversary to succeed in the attack. On the other hand, if the network delay is big enough, it is more likely to be larger than this lower bound.

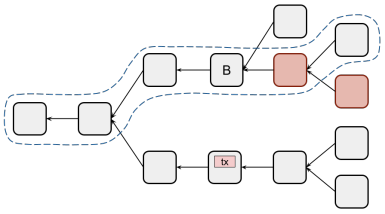
An example of the execution of the attack for $r = 2$ and 3 blocks needed for confirmation is illustrated in Figure 4.2.



(a) Attacker starts disrupting communications between H_1 and H_2 and issues a transaction in H_2.



(b) Attaker starts mining on top of block B in $view(H_1)$. At the same time interval, 3 blocks are mined on top of the block containing the malicious transaction, thus the merchant considers the transaction confirmed.



(c) Attacker stops disrupting communications. The GHOST chain (in blue) does not contain the block with the malicious transaction, thus the adversary can double-spend.

Figure 4.2: An example of the execution of a balance attack, when the number of blocks needed to confirm a block are 3.

4.4 Security of the GHOST Protocol

4.4.1 Resilience under secret mining attacks

Sompolinsky and Zohar [7] provided an initial security analysis of GHOST and proved that the GHOST protocol avoids the secret mining attack of the Bitcoin protocol (Figure 4.3).

Modelling the block creation as a Poisson process, let λ_h be the honest block creation rate. For a block B, let $time(B)$ be the moment in time B was created.

Proposition 4.4 (GHOST is resilient to secret mining attacks [7])**.** *Assume the attacker's block creation rate is $q \cdot \lambda_h$, and $0 \leq q < 1$. The probability that a block B will be off the main chain sometime after $time(B) + s$, given that it was in the main chain at $time(B) + s$, goes to zero as s goes to infinity.*

The proof of Proposition 4.4 is based on the fact that the attacker possesses less computational power than the honest parties. If a block B is in the main chain of honest parties at time $time(B) + s$, the difference in the size of the subtree rooted at B and the attacker's tree grows, as s goes to infinity, by the law of large numbers.

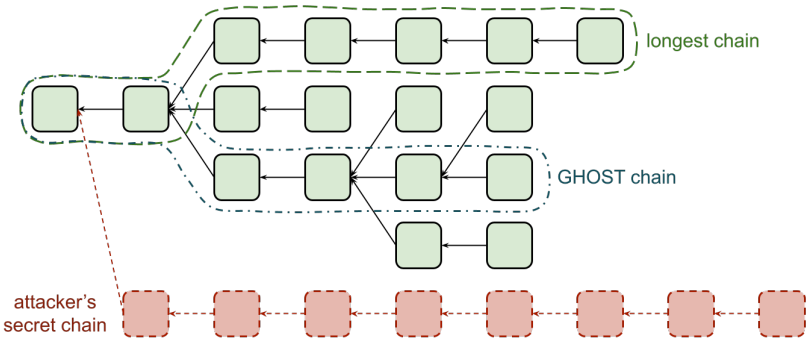


Figure 4.3: The attacker's secretly mined chain is able to win over the longest chain, but not over the GHOST chain.

4.4.2 GHOST in the synchronous setting

Although Sompolinsky and Zohar [7] analyzed the security of the GHOST protocol, the analysis was not as general as the security analyses of Bitcoin (for example as the analysis of Garay et al. [4]), since it did not include the possibility that the adversary may send conflicting information to different parties. For this reason, Kiayias and Panagiotakos [11] modified the model of Garay et al. [4] proposing a model that is based on trees rather than on chains and proved the security of the GHOST protocol in the synchronous setting.

In their model, they expressed the notion of a block being d-dominant, which intuitively means that the block is "stronger" than its siblings by a quantity of d.

Definition 4.5 ([11])**.** Let w be a function defined on trees which takes only non-negative values. For a directed blockTree T, let $siblings(B)$ denote the set of vertices in T that share the same parent with B. Let also $T(B)$ be the subrtree of T rooted at B. Then, block B is d-dominant in T if

$$w(T(B)) \geq d \text{ and } \forall u \in siblings(B) : w(T(B)) \leq w(T(B)) + d.$$

Using the notion of d-dominant blocks, a unified description of the Bitcoin and GHOST backbone protocols can be obtained as follows. Let for a blocktree T, $w(T)$ to be equal to the height of T, and assume that miners in the Bitcoin protocol keep all valid blocks they receive instead of keeping only the blocks of the longest chain. We can describe the procedure that a miner follows to calculate the longest chain as the procedure in which the miner starts from the $genesis$ block, and then at each step picks one of the 0-dominant children, until reaching a leaf. We can describe in the same way the procedure to find the GHOST chain in the GHOST protocol, by setting $w(T)$ to be the number of blocks in T.

To prove the security of GHOST, Kiayias and Panagiotakos, similarly to Garay et al. [4], separate the protocol used to maintain the data structure, which they named *GHOST backbone*, from the distributed ledger application. They show that the GHOST backbone satisfies the *Fresh Block lemma*. Using the notation of Table 4.2, the Fresh Block lemma is described as follows.

p	probability that a query to the random oracle is successful
λ	tail-bounds parameter
n	total number of parties mining
t	number of parties controlled by the adversary
q	number of queries each party has to the random oracle
δ	advantage of honest parties, $(t/(n-t) \leq 1-\delta)$
α	probability at least one honest party succeeds in finding a POW in a round $\alpha = 1-(1-p)^{q(n-t)}$
γ	probability that exactly one honest party succeeds in finding a POW in a round
β	expected number of blocks the adversary can mine in one round $\beta = pqt$
f	total mining power $f = \alpha + \beta < 1$

Table 4.2: Notation

Lemma 4.6 (Fresh Block [11])**.** *Assume $\gamma \geq (1+\delta)\beta$, for some real $\delta \in (0,1)$ and $f < 1$. Then, for all $s \in \mathbb{N}$ and $r \geq s$ it holds that there exists a block mined by an honest player on and after round $r-s$, that is contained in the chain which any honest player adopts on and after round r with probability $1 - e^{-\Omega(\delta^2 s)}$.*

The Fresh Block lemma is sufficient to prove in a black box manner that the public ledger built on top of the GHOST protocol satisfies Persistence and Liveness. The definitions of Persistence and Liveness are similar to those of Garay et al. for the Bitcoin protocol (Definitions 3.6 and 3.7), with the difference that the "depth" parameter is a "weight" parameter: for both properties, blocks should have a weight of at least k instead of being k blocks deep in the ledger.

Theorem 4.7 (Persistence [11], simplified form)**.** *Suppose $\gamma \geq (1+\delta)\beta$ and $(1+\delta)f \leq 1$, for some $\delta \in (0,1)$. Then, for all $k \in \mathbb{N}$, the public ledger built on top of the GHOST protocol satisfies Persistence with depth parameter k, with probability $1 - e^{-\Omega(\delta^2 k)}$.*

Theorem 4.8 (Liveness [11], simplified form)**.** *Assume $\gamma \geq (1+\delta)\beta$, for some $\delta \in (0,1)$ and $f < 1$. Then, for all $k \in \mathbb{N}$, the public ledger built on top of the GHOST protocol satisfies Liveness with wait time $u = k + \frac{k}{(1-\delta)\alpha}$ rounds and depth parameter k, with probability $1 - e^{-\Omega(\delta^2 k)}$.*

The Fresh Block lemma is sufficient to prove Persistance and Liveness. For Persistence, honest parties wait for a sufficient amount of time after a block has been included in the GHOST chain to consider it committed, such that in this amount of time the Fresh Block lemma is applicable, and an honest block that is a descendant of the block containing the transaction appears in the GHOST chain. As a result, the block containing the transaction will be in every honest party's chain. For Liveness, the u rounds for wait time are set so as to be sufficient for the Fresh Block lemma to apply, and similarly to Persistence, the block containing the transaction will be a descendant of an honest block that is in the chain of every honest party. Then, all honest parties will mine on a chain containing the block with the transaction, and thus the block will eventually be commited.

4.4.3 GHOST in the bounded network delay setting

Discrete time analysis of GHOST

Kiffer et al. [18] proved that GHOST satisfies Persistence, with the same bound as that of Bitcoin. Namely, GHOST is secure if Equation 3.4 holds.

Continuous time analysis of GHOST

In this section we present an attempt for a proof of the Safety property of GHOST in the continuous time setting where proof-of-work is modelled as a Poisson process, using techniques of Ren [23] and Kiayias and Panagiotakos.[11]. Our proof attempt implies that GHOST is as secure as Bitcoin, in the sense that the adversarial power tolerance for the Safety property to hold is the same for both protocols.

The theorem that we are interested in is the following, and is similar to that of Ren [23] for Bitcoin.

Theorem 4.9 (informal)**.** *Let $g = e^{-\alpha\Delta}$ and δ be any positive constant. The GHOST protocol with the k-confirmation rule satisfies consistency except for $e^{-\Omega(\delta^2 g^2 k)}$ probability if*

$$g^2\alpha > (1+\delta)\beta.$$

In Theorem 4.9, the values α, β are the total mining rate of honest and adversarial parties, respectively, and Δ is an upper bound on the network delay. Furthermore, the k-confirmation rule dictates that a party considers a block committed if it is in the GHOST chain of that party and its subtree is composed of more than k blocks.

Essentially, Theorem 4.9 states that the bound of Ren [23] on the adversarial power tolerance of Bitcoin also holds for GHOST. This bound, which is proven in the continuous time setting, is equivalent to that of Kiffer et al. [18] in which mining is modelled as a Bernoulli random variable. Thus, our proof attempt does not imply a better bound than the ones already proven for GHOST. Our methodology consists of proving Safety using a *Common Prefix* methodology, similar to that of Bitcoin [4, 23, 14].In the Bitcoin protocol, apart from the use in proving Safety, the Common Prefix methodology is often required for compositions [17] and in security proofs of protocols built on top of it as for example light client constructions [33]. Thus, Safety is only one of the many potential applications of the Common Prefix methodology. We hope that the common prefix methodology for the GHOST protocol can aid in the construction of similar protocols using the GHOST protocol as a base. The techniques that we use were present in the work of Kiayias and Panagiotakos [11], however, a lot of them were not stated explicitly, but rather used in the Fresh Block lemma proof.

Model. Every miner stores a local blockTree, follows the GHOST rule on their blockTree and mine on top of the tip of the GHOST chain, which they find using Algorithm 3. Ties are broken arbitrarily. Upon mining or receiving a new block, miners broadcast the block to other miners. When receiving a new block, a miner adds it to their local tree only if they have received all its ancestors, in other case they add it after receiving them. We assume that the network suffers a delay which is upper bounded by a constant Δ, this means that if an honest miner broadcasts a block at time t, by time $t + \Delta$ every other honest miner should have received it, but it is possible that they may have not have received it earlier than $t+\Delta$. Adversarial miners do not suffer from any delay, meaning that if an adversarial miner broadcasts a block at time t, honest and adversarial miners

can receive it at time t. The adversary has the ability to send different blocks to different parties or send blocks only to specific parties. However, honest parties broadcast the blocks they receive thus in the case the adversary sends blocks to a specific party at time t every honest party will have received it at time $t + \Delta$. The adversary also controls the delay of the honest miners. If a miner broadcasts a block, the adversary can control the arrival time of that block to other miners, however, he cannot delay its arrival by more than Δ. However, the adversary cannot alter the content of the messages sent by honest miners.

Preliminary definitions. For a blockTree $T = (V, E)$ and a block $B \in V$, let $w_T(B)$ be the number of vertices of the subtree of T rooted at B. When T is clear from context, we will write $w(B)$ instead of $w_T(B)$. For a party P and a moment in time t let T_t^P be the local tree of party P at time t. The k-confirmation rule dictates that an honest party P considers a block B committed at time t if B is in the GHOST chain of T_t^P and $w_{T_t^P}(B) \geq k$.

Our proof attempt implies an upper bound on the adversarial power tolerance of the GHOST protocol in order to satisfy *Safety* with overwhelming probability.

Definition 4.10 (Safety)**.** Parameterized by $k \in \mathbb{N}$, if for an honest party P at time t a block B is considered committed, then for every honest player P' at time $t' \geq t$ block B is considered committed.

We will use the notion of *dominance* of a block, introduced by Kiayias and Panagiotakos. Let $T = (V, E)$ be a blocktree and $B \in V$. We denote by $siblings_T(B)$ the set of blocks in T that have the same parent as B. Dominance is defined as follows.

Definition 4.11 ([11], rephrased)**.** Let $T = (V, E)$ be a blockTree and $B \in V$. For $d \geq 0$, we say that the block B is d-dominant in T if:

- $w(B) \geq d$, if $siblings_T(B) = \emptyset$
- $w(B) \geq d$ and $w(B) \geq w(B') + d$ for every $B' \in siblings_T(B)$, if $siblings_T(B) \neq \emptyset$.

We observe that if a block is $(d + 1)$-dominant then it is d-dominant. Also, using the above definition, the GHOST chain in the local tree of a player P at time t is a 0-dominant root-leaf path in the local tree of player P at time t.

Definition 4.12. Let $T = (V, E)$ be a blockTree and $B \in V$. The *rooted path up to* B is the path from the genesis block to block B (including B). We say that the rooted path up to B is d-dominant in T if every block in the rooted path up to B is d-dominant in T. Furthermore, if B is in the GHOST chain of this tree, we say instead that the GHOST chain up to B is d-dominant in T.

The same definition as Definition 4.12 with different notation is present in the work of Kiayias and Panagiotakos [11].

We observe that if the GHOST chain of a player up to block B is d-dominant, it means that at least d blocks should be added to that player's local tree in order for a block of the GHOST chain up to B to be potentially kicked out of the GHOST chain for this player (the block will become 0-dominant, so then it depends on tie-breaking). It is necessary that these blocks affect the dominance of the block in order for it to be kicked out of the chain. But where do these blocks should point to in order to affect the dominance of the block? From the definition of the dominance of a block, we can observe the following.

Observation 4.13. Assume that a block B is d-dominant in a tree T, and that k new blocks are added to T. Then:

(a) the dominance of B will be reduced by k if and only if these k new blocks are descendants of $parent(B)$ but are not descendants of B.

(b) The dominance of B will be increased by k if and only if these k new blocks are descendants of B.

Furthermore, B will remain d-dominant if and only if new blocks (potentially 0) are added to T and these blocks are not descendants of $parent(B)$.

Preliminary lemmas. We transfer the main lemmas for the synchronous setting of Kiayias and Panagiotakos [11] to the bounded delay setting. All the proofs that we provide are similar to theirs.

The fist lemma states that if in a time period the adversary broadcasts less blocks that the number of loner blocks mined in this period, after the end of the period there will exist a loner block that will be in the GHOST chain of every honest party. Furthermore, the dominance of the GHOST chain will be equal to the difference between the number of loner blocks mined and the number of adversarial blocks broadcasted in this time period.

Lemma 4.14. *Assume that exactly m loner blocks are mined in $[t_0 + \Delta, t_1 - \Delta]$, and assume that the adversary broadcasts $k < m$ blocks in $[t_0, t_1]$. Then, there exists a loner block B, mined in $[t_0 + \Delta, t_1 - \Delta]$ such that at time t_1, block B is in the GHOST chain of every honest party and the GHOST chain up to B is $(m - k)$-dominant.*

Proof. We will prove the lemma by induction on m.

If $m = 1$, then $k = 0$. This means that the adversary does not broadcast any block in $[t_0, t_1]$. Let B be the loner block mined in $[t_0 + \Delta, t_1 - \Delta]$ and let t be the time it was mined. Since B is a loner block, no other honest block is mined in $[t - \Delta, t + \Delta]$ and no adversarial block is broadcasted in the same interval. Thus, all parties have the same view of the ledger at time t. The honest block will be mined on the GHOST chain, and by time $t + \Delta$ everyone will have received it, and the GHOST chain up to B will be 1-dominant. In $(t + \Delta, t_1]$ every honest block will be a descendant of B and no adversarial block is broadcasted, thus the dominance of the GHOST chain up to B can only increase. We conclude that at time t_1 block B is in the GHOST chain of every honest party and the GHOST chain up to B is 1-dominant for every honest party.

Assume that the lemma holds for $m - 1$ loner blocks.

For m loner blocks, let t'_1 be the time that the $(m - 1)$-th loner block was mined, and t be the time that the m-th loner block was mined. Let also k_1 be the number of adversarial blocks broadcasted in $(t_0, t'_1 + \Delta]$. We divide into two cases.

1. If $k_1 = m - 1$, then the adversary does not broadcast any block in $(t'_1 + \Delta, t_1]$. Since at times t'_1 and t loner blocks are mined we have $t'_1 + \Delta < t$, thus before time t all honest parties have received all adversarial blocks. Furthermore, no honest block is mined in $[t - \Delta, t)$. Thus all parties will have the same view of the ledger by time t. Let B be the m-th loner block. The honest party that will mine B will mine it on top of the GHOST chain and by time $t + \Delta$ every honest party will have received B. Because of B the GHOST chain up to block B will be 1-dominant, and in $(t + \Delta, t_1]$ since the adversary does not broadcast any blocks, every other honest block will be a descendant of B, thus the dominance

of the GHOST chain up to B can only increase. This means that at time t_1 block B is in the GHOST chain of every honest party and the GHOST chain up to B is 1-dominant.

2. If $k_1 < m - 1$, by the induction hypothesis there exists a loner block B mined in $[t_0 + \Delta, t_1']$ such that at time $t_1' + \Delta$ block B is in the GHOST chain of every honest party and the GHOST chain up to B is $(m-k_1)$-dominant for every honest party.

   Let

   - k_2 be the number of adversarial blocks broadcasted in $(t_1' + \Delta, t]$,
   - k_3 be the number of adversarial blocks broadcasted in $(t, t + \Delta]$, and
   - k_4 be the number of adversarial blocks broadcasted in $(t + \Delta, t_1]$.

   We divide into two cases.

   (a) If $k_2 = m - 1 - k_1$, then either B remains in the GHOST chain of every honest player in $(t_1' + \Delta, t]$ or the chain up to B can become 0-dominant for at least one honest player. However, all blocks that have height less or equal to the height of B can be at most 0-dominant for every honest player in $(t_1 + \Delta, t]$. The honest player that will mine the m-th loner block B' will mine it on his local GHOST chain, thus his local GHOST chain up to B' will be 1-dominant.
   Since $k_2 = m-1-k_1$, we have $k_3 = k_4 = 0$. Thus, the adversary will not broadcast any block in $[t, t_1]$ and similarly to the induction base, at time t_1 block B' will be in the GHOST chain of every honest party and the GHOST chain up to B' will be 1-dominant.

   (b) If $k_2 < m - 1 - k_1$, then in $(t_1' + \Delta, t]$ block B will remain in the GHOST chain of every honest party and the GHOST chain up to B will be $(m - 1 - k_1 - k_2)$-dominant. This means that the m-th loner block will be a descendant of B. In $(t, t + \Delta]$ no honest block is mined and the adversary broadcasts $k_3 \leq m - 1 - k_1 - k_2$ blocks. Since at time $t + \Delta$ every honest party will have received the m-th loner block, block B is in the GHOST chain of every honest party and the GHOST chain up to B is $(m - k_1 - k_2 - k_3)$-dominant. Since $k_4 < m - k_1 - k_2 - k_3$, the dominance of the GHOST chain up to B can decrease by at most k_4, thus B will remain in the GHOST chain of every honest party in $(t+\Delta, t_1]$ Thus at time t_1, block B is in the GHOST chain of every honest party and the GHOST chain up to B is $(m - k_1 - k_2 - k_3 - k_4)$-dominant.

□

For the next lemma, we need the following definition.

Definition 4.15. For a moment in time t, let $T_t = \bigcup_{P:P \text{ honest}} T_t^P$. If the adversary broadcasts blocks at time t, we assume that T_t contains these blocks.

Since we are arguing about security in the continuous time setting, we need to define random variables for the adversarial and honest blocks in a continuous time interval.

Definition 4.16. For any continuous time interval I we define Y_I and Z_I to be the number of loners and malicious blocks mined in the interval, respectively. Let also Z_I^{bd} be the number of blocks the adversary broadcasts in the interval I.

The following lemma is a corollary of Lemma 4.14, and states that in order for the adversary to make honest parties switch chains, he has for a period of time to broadcast more blocks that the number of loner blocks in that period.

Lemma 4.17. *Let $B_1, B_2, \ldots, B_r$ where $r \geq 2$ be a sequence of loner blocks such that block B_i is mined in time t_i and $t_1 < t_2 < \cdots < t_r$. Assume that B_i is the first loner block in $[t_1, t_r]$ that is not a descendant of B_{i-1} for $i \geq 2$. In order for this sequence of loner blocks to exist, the inequality $Z^{bd}_{[t_1-\Delta,t_2]} \geq Y_{[t_1,t_r-\Delta)}$ should hold. Furthermore, if $Z^{bd}_{[t_1-\Delta,t_r]} = Y_{[t_1,t_r-\Delta)}$, then block B_r is in a 1-dominant path in T_{t_r}.*

Proof Sketch. The lemma follows by induction. We provide a proof for the induction base, for which we transfer the corresponding induction base proof for the similar lemma appearing in the work of Kiayias and Panagiotakos [11]. The induction step follows similarly.

For $r = 2$, assume for the purpose of contradiction that $Z^{bd}_{[t_1-\Delta,t_2]} < Y_{[t_1,t_2-\Delta)}$ Let t be the time the last loner block was mined in $[t_1, t_2 - \Delta]$, then we have $Y_{[t_1,t_2-\Delta)} = Y_{[t_1,t]}$. We have assumed $Z^{bd}_{[t_1-\Delta,t_2]} < Y_{[t_1,t_2-\Delta)}$ and since $Z^{bd}_{[t_1-\Delta,t+\Delta]} \leq Z^{bd}_{[t_1-\Delta,t_2]}$, we have $Z^{bd}_{[t_1-\Delta,t+\Delta]} < Y_{[t_1,t]}$. By applying Lemma 4.14 for the interval $[t_1, t]$, we have that there exists a loner block B, mined in $[t_1, t]$ such that B is in the GHOST chain of every honest party at time $t + \Delta$, and the GHOST chain up to B is $(Y_{[t_1,t_2-\Delta]} - Z^{bd}_{[t_1-\Delta,t+\Delta]})$-dominant at time $t + \Delta$.

Since $Z^{bd}_{(t+\Delta,t_2]} < Y_{[t_1,t_2-\Delta)} - Z^{bd}_{[t_1-\Delta,t+\Delta]}$, block B will remain in the GHOST chain of every honest party in $(t + \Delta, t_2]$ and thus block B_2 that will be mined at time t_2 will be a descendant of B. However, B is a descendant of B_1 by the definition of B_2 and B_2 is a descendant of B, thus B_2 is a descendant of B_1 which is a contradiction. Thus, $Z^{bd}_{[t_1-\Delta,t_2]} \geq Y_{[t_1,t_2-\Delta)}$.

Assume now that $Z^{bd}_{[t_1-\Delta,t_2]} = Y_{[t_1,t_2-\Delta)}$. We divide into two cases.

1. Assume that $Z^{bd}_{[t_1-\Delta,t+\Delta]} = Y_{[t_1,t]}$.

   If $t + \Delta < t_2 - \Delta$ then no adversarial block is mined in $[t_2 - \Delta, t]$, and no loner block is mined. Thus every honest party will have the same view of the tree at time t_2, and the honest party that will mine block B_2 will mine it on top of a 0-dominant rooted path, making the path 1-dominant in T_{t_2}.

   Else, if $t + \Delta \geq t_2 - \Delta$, then let p be the rooted path up to B_1 in $T_{t+\Delta}$, and let $B \in p$.

   For every $B' \in siblings_{T_{t+\Delta}}(B)$,

   $$w_{T_{t+\Delta}}(B) \geq Y_{[t_1,t-\Delta]} = Y_{[t_1,t_2-\Delta]} = Z^{bd}_{[t_1-\Delta,t+\Delta]} \geq w_{T_{t+\Delta}}(B') \qquad (4.1)$$

   Furthermore, T_{t_2} is the tree composed by adding block B_2 to $T_{t+\Delta}$. Block B_2 cannot be a descendant of block B_1 by the theorem statement. Thus B_2 will be mined on top of a path that by Equation 4.1 will be 0-dominant in $T_{t+\Delta}$, and will become 1-dominant in T_{t_2}.

2. If $Z^{bd}_{[t_1-\Delta,t+\Delta]} < Y_{[t_1,t]}$, then as previously discussed, there exists a loner block B, mined in $[t_1, t]$ that is in the GHOST chain of all honest parties at time $t + \Delta$, and the GHOST chain up to B is $(Y_{[t_1,t_2-\Delta]} - Z^{bd}_{[t_1-\Delta,t+\Delta]})$-dominant at time $t + \Delta$ for every honest party. Let p' be the GHOST chain up to B. Since

$Z^{bd}_{(t_1+\Delta,t_2]} = Y_{[t_1,t_2-\Delta]} - Z^{bd}_{[t_1-\Delta,t+\Delta]}$ in the worst case the adversary can decrease the dominance of every block in p' to 0 for some honest parties, however, he cannot make the blocks of a rooted path that does not contain p' as a prefix 1-dominant. Block B_2 cannot a descendant of B as previously discussed, thus it will be mined on top of a path that is 0-dominant for the player mining B_2.

□

In GHOST, the length of the chains of honest parties does not necessarily increase monotonically, and the adversary can use very old blocks to switch the chain of honest parties. The following observation gives a lower bound on the blocks the adversary may broadcast to do so, and is essential to argue about the Safety property of GHOST using a Common Prefix lemma format. This obseration was also used in the work of Kiayias and Panagiotakos for the Fresh Block lemma proof.

Observation 4.18. Assume that there exists an honest block B^*, mined in $t^* \leq t_0$, such that

1. every loner block in $[t_0 + \Delta, t_1 - \Delta]$ is a descendant of this block, and
2. block B^* is in the GHOST chain of every honest party at time t_1.

Then, Lemma 4.14 still holds if we further assume that the adversary broadcasts as many blocks as they want, but only k of them are descendants of B^*, as long as all loner blocks are descendants of B^* and B^* is in the chain of every honest party at time t_1. The reason is that blocks that are not descendants of B^* will not affect the dominance of blocks that are descendants of B^*. Equivalently, we can ignore the blocks the adversary broadcasts that are mined before t^*, since these blocks will not be descendants of B^*. Since Lemma 4.17 is a corollary of Lemma 4.14, Lemma 4.17 also holds under that assumption. Although block B^* is chosen by the adversary, the genesis block satisfies these properties, so such a block always exists.

Consequences of Common Prefix violation. The following lemma states that if a block is considered committed by an honest party and at some later moment in time another party has a chain that does not include this block, there should exist a period of time in which the adversary mined at least as many blocks as the number of loner blocks in this period. This argument is essentially Equation 3.2 in Chapter 3 which is used in many security proofs for Persistence and Liveness of Bitcoin [4, 23] and of GHOST [11].

Theorem 4.19 (Consequence of common prefix violation)**.** *Consider any time t and block B that is considered committed by some honest party at time t. If at time $t' \geq t$ an honest party adopts a chain which does not contain B, then there exist times t_0, t_1 where $t_0 < t \leq t_1 \leq t'$ such that $Z_{[t_0,t_1]} \geq Y_{[t_0+\Delta,t_1-\Delta]}$. Furthermore, in t_0 an honest block B_0 is mined which is also considered committed at time t by the honest party.*

Proof. Let t_1 be the first time after B was committed that an honest party adopts such a chain. Take the most recent honest block B_0 that is an ancestor of B and mined at some time $t_0 < t$ that satisfies the following properties:

1. every loner block in $[t_0 + \Delta, t_1 - \Delta]$ is a descendant of B_0, and

2. block B_0 is in the chain of every honest party at time t_1.

Such a block is well defined since the genesis block satisfies this property. Also, since B_0 is an ancestor of B, at time t the weight of B_0 for the honest party that considers B committed is greater than k, thus B_0 is also considered committed by the honest party.

Let $B_1, B_2, \ldots, B_r$ be the longest possible sequence of loner blocks such that block B_i is mined in time s_i, where $t_0 + \Delta \leq s_1 < s_2 < \cdots < s_r \leq t_1 - \Delta$, block B_1 is the first loner block mined in $[t_0 + \Delta, t_1 - \Delta]$, and B_i is the first loner block in $[s_1, s_r]$ that is not a descendant of B_{i-1} for $i \geq 2$. Then:

- For the interval $[t_0 + \Delta, s_r]$, by applying Lemma 4.17 for the interval $[s_1, s_r]$, we have that $Z^{bd}_{[s_1-\Delta, s_r]} \geq Y_{[s_1, s_r-\Delta]}$. Since s_1 is the time the first loner block was mined in $[t_0 + \Delta, t_1 - \Delta]$, we have that $Z^{bd}_{[t_0,s_r]} \geq Z^{bd}_{[s_1-\Delta,s_r]}$, thus

$$Z^{bd}_{[t_0,s_r]} \geq Y_{[t_0+\Delta, s_r-\Delta]}. \tag{4.2}$$

- For the interval $(s_r, t_1 - \Delta]$, we have that the loner blocks mined form a chain and are descendants on B_r. Assume that $Z^{bd}_{(s_r,t_1]} < Y_{(s_r+\Delta, t_1-\Delta]}$. Then, by Lemma 4.14 we have that there exists a loner block B' mined in $[s_r + \Delta, t_1 - \Delta]$ such that B' is in the chain of every honest party at time t_1. If B' is an ancestor of B, since all loner blocks mined after B' are descendants of B' from the fact that $B_1, \ldots, B_r$ was the longest possible sequence, we obtain a contradiction, since B_0 was the most recent block with this property. If B' is a descendant of block B, then B is in the chain of every honest party at time t_1, which is a contradiction. If B' is nor an ancestor nor a descendant of B, then since all loner blocks after B' form a chain and are in the chain of every honest party at time t_1, we have that there are earlier times than t_1 for which honest parties have chains that do not contain B, which is a contradiction. Thus,

$$Z^{bd}_{(s_r,t_1]} \geq Y_{(s_r+\Delta, t_1-\Delta]}. \tag{4.3}$$

We have that

$$Z^{bd}_{t_0,t_1} = Z^{bd}_{t_0,s_r} + Z^{bd}_{(s_r,t_1]} \tag{4.4}$$

and

$$Y_{[t_0+\Delta, t_1-\Delta]} = Y_{[t_0, s_r-\Delta]} + Y_{(s_r+\Delta, t_1-\Delta]} + 1. \tag{4.5}$$

We will show that equality cannot hold in both Equations 4.2 and 4.3, and thus proving

$$Z^{bd}_{[t_0,t_1]} \geq Y_{[t_0+\Delta, t_1-\Delta]}.$$

Assume that $Z^{bd}_{[t_0,s_r]} = Y_{[t_0+\Delta, s_r-\Delta]}$, and let

$$T_{s_r} = \bigcup_{P:P \text{ honest}} T^P_{s_r}.$$

Then, $B_r \in T_{s_r}$. Let p be the rooted path up to B_r in T_{s_r}, then by Lemma 4.17, every block in p is 1-dominant in T_{s_r}. This means that for all $B' \in p$ and for all $B'' \in siblings_{T_{s_r}}(B')$ we have

$$w_{T_{s_r}}(B') \geq w_{T_{s_r}}(B'') + 1. \tag{4.6}$$

Let also $T_{t_1-\Delta} = \bigcup_{P:P \text{ honest}} T^P_{t_1-\Delta}$ and assume $Y_{(s_r+\Delta,t_1-\Delta]} = Z^{bd}_{(s_r,t_1-\Delta)}$. Assume that at least Δ time has passed from $s_r + \Delta$ until $t_1 - \Delta$, else $Y_{(s_r+\Delta,t_1-\Delta]} = 0$ and this case is trivial. Then, we have that the blocks that the honest players have in their local trees at time s_r will have fully propagated to all honest players at time $t_1 - \Delta$, thus $T_{s_r} \subseteq T_{t_1-\Delta}$.

We will abuse notation and for a block $B' \in T_{t_1-\Delta}$, if $B' \notin T_{t_{s_r}}$ we will behave as $B' \in T_{s_r}$ but $w_{T_{s_r}}(B') = 0$.

Then in $T_{t_1-\Delta}$, for every $B' \in p$, since loner blocks mined after B_r are all in the same chain, we have that $w_{T_{t_1-\Delta}}(B') \geq w_{T_{s_r}}(B') + Y_{(s_r+\Delta,t_1-\Delta]}$, and since $Z^{bd}_{[s_r,t_1-\Delta]} = Y_{(s_r+\Delta,t_1-\Delta]}$ we have that

$$w_{T_{t_1-\Delta}}(B') \geq w_{T_{s_r}}(B') + Z^{bd}_{(s_r,t_1-\Delta]}. \tag{4.7}$$

Furthermore, for every block B' and every block $B'' \in siblings_{T_{t_1-\Delta}}(B')$ we have

$$w_{T_{t_1-\Delta}}(B'') \leq w_{T_{s_r}}(B'') + Z^{bd}_{(s_r,t_1-\Delta]}. \tag{4.8}$$

Combining Equations 4.6, 4.7 and 4.8, we have that for every $B'' \in siblings_{T_{t_1-\Delta}}(B')$,

$$w_{T_{t_1-\Delta}}(B') \geq w_{T_{t_1-\Delta}}(B'') + 1,$$

which means that p is 1-dominant in $T_{(t_1-\Delta]}$. At time t_1 every honest party will have received every block of $T_{t_1-\Delta}$, and since $Z^{bd}_{t_1-\Delta,t_1} = 0$, at time t_1 p will remain 1-dominant for every honest player P and thus p will be a prefix of the GHOST chain of every player p. However, $B_r \in p$, thus B_r is in the chain of every honest party at time t_1 and all loner blocks mined after time s_r are descendants of B_r. Furthermore, if block B_r is mined after time t, then it should be a descendant of B, since t_1 is the first time after t that an honest party has a chain that does not contain B. But then, since B_r is in the chain of every honest party at time t_1, the same should hold for B, which is a contradiction. Thus, B_r is mined before time t. However, this contradicts the definition of B_0.

Thus, we showed that equality cannot hold in both Equations 4.2 and 4.3. Combining this fact together with equations 4.5 and 4.4 we have that

$$Z^{bd}_{[t_0,t_1]} \geq Y_{[t_0+\Delta,t_1-\Delta]}.$$

Since we used Lemma 4.14 and 4.17, by applying Observation 4.18 we can ignore the blocks the adversary broadcasts that are mined before t_0 and the inequality $Z^{bd}_{[t_0,t_1]} \geq Y_{[t_0+\Delta,t_1-\Delta]}$ will still hold. Thus $Z_{[t_0,t_1]} \geq Z^{bd}_{[t_0,t_1]}$ (the adversary may mine more blocks than the ones they broadcast) and thus

$$Z_{[t_0,t_1]} \geq Y_{[t_0+\Delta,t_1-\Delta]}.$$

Also, since B_0 is an ancestor of B, at time t the weight of B_0 for the honest party that considers B committed is greater than k, thus B_0 is also considered committed by the honest party. □

Safety Using Theorem 4.19, the Safety property of GHOST can be proven as follows.

Theorem 4.20 (Safety)**.** *Let $g = e^{-\alpha\Delta}$ and consider any $0 < \delta < 1$ such that $g^2\alpha > (1+\delta)\beta$. Consider any time t and any block B that is considered committed by some honest party at time t. Except for $e^{-\Omega(\delta^2 g^2 k)}$, for all time $t' \geq t$, no honest party has a chain that does not contain B.*

Proof Sketch. Consider a time t and a block B that is considered committed by some honest party at time t. Furthermore, let $\mathcal{E}$ be the event that at time $t' \geq t$ an honest party has a chain that does not contain B. By applying Theorem 4.19, we obtain the following.

(a) There exist times t_0 and t_1 such that $t_0 < t$ and $t \leq t_1 \leq t'$ such that $Z_{[t_0,t_1]} \geq Y_{[t_0+\Delta,t_1-\Delta]}$.

(b) At time t_0 an honest block B_0 is mined that is also considered committed at time t by the honest party that considered B committed at time t.

Using points (a) and (b) above, the proof that $\mathcal{E}$ happens with probability $e^{-\Omega(\delta^2 g^2 k)}$ is identical to the proof of Ren [23] for Bitcoin. □

CHAPTER 5

THE PHANTOM AND GHOSTDAG PROTOCOLS

5.1 Introduction

5.1.1 From trees to DAGs

As discussed in Chapter 3, when increasing block creation rates in the Bitcoin protocol, more forks are created. Apart from the effect on security, this also means that in protocols that use some chain selection rule, among blocks that are created in parallel only one of them is selected although many of them could have been created by honest miners. This means that a linear increase in block creation rate does not necessarily result in a linear increase in throughput, since the transactions inside blocks that could be created by honest miners are not included into the ledger. Figure 5.1 illustrates this situation.

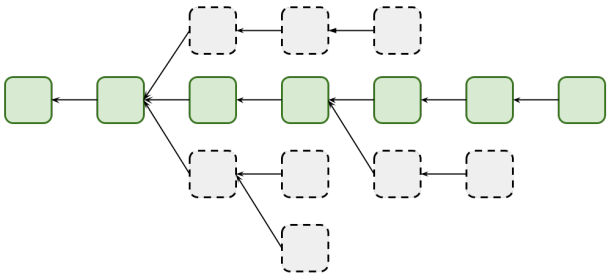


Figure 5.1: Bitcoin under high block creation rate. Only one block is selected out of every fork and as a consequence a lot of transactions in blocks created by honest miners are not included in the ledger.

To solve this problem, Lewenberg et al. [6] proposed to modify the Bitcoin protocol and use a Directed Acyclic Graph data structure in place of a blockchain. The idea is that when a new block is created, it should reference all blocks that in the view of the

miner are not already referenced by some other block, contrary to referencing only one block as in chain based protocols such as Bitcoin and GHOST. Having more references results in the ability to extract more information about the relation between the creation times of blocks. The idea of using a DAG instead of a chain has been present in many other distributed ledger protocols [20, 17, 35, 32, 30].

The data structure created when restructuring the blockchain into a DAG is commonly called a blockDAG [6], [34]. Although if we consider the blocks as vertices and the references as edges the resulting data structure is a DAG, not every DAG can be the resulting data structure of this reconstruction. This means that the class of blockDAGs is a subclass of the class of DAGs. The following definition characterizes blockDAGs.

Definition 5.1 (blockDAG, [36])**.** A *blockDAG* is a Directed Acyclic Graph $G = (V, E)$ in which vertices represent blocks and edges represent hash references, and satisfies the following properties.

- There exists only one sink (i.e., a vertex with out-degree 0), and
- if $(u, v_1), (u, v_2) \in E$ then there is no directed path from v_1 to v_2.

An example of a blockDAG appears in Figure 5.2

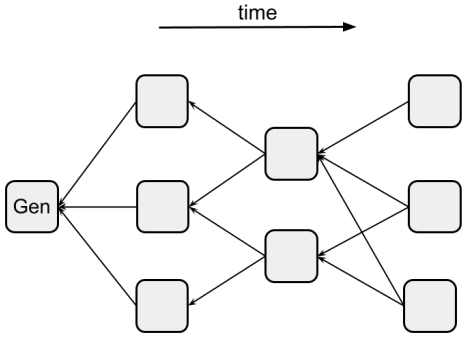


Figure 5.2: An example of a blockDAG.

5.1.2 The PHANTOM and GHOSTDAG protocols

The PHANTOM protocol and its optimization, GHOSTDAG [34], are DAG-based protocols that generalize Nakamoto's longest chain protocol in order to achieve scalability. In contrast with other protocols such as SPECTRE [10], PHANTOM and GHOSTDAG provide a total ordering of transactions, and thus have the capability of supporting smart contracts. The GHOSTDAG protocol is said by the authors to be a fix to the GHOST protocol discussed in Chapter 4 and has been implemented in the KASPA network [40].

In the following sections we will give an overview of the PHANTOM and GHOSTDAG protocols and discuss the security of GHOSTDAG.

5.2 The PHANTOM Protocol

5.2.1 Intuition

As discussed in previous chapters, the throughput of the Bitcoin protocol is suppressed since its security is dependent on the number of blocks that are able to fully propagate before a new block is mined. This means that in an interval of length $2 \cdot \Delta$, where Δ is an upper bound on the network delay, fewer than one block should be mined in order for the protocol to be secure. The core idea behind PHANTOM is that we could use a blockDAG structure, allow multiple blocks to be mined in an interval of length $2 \cdot \Delta$, and distinguish between honest an malicious blocks using information from the structure of the formed blockDAG. Furthermore, the longest chain rule results as a special case of the procedure used to distinguish between honest and malicious blocks.

5.2.2 Mining protocol

When mining a new block, a miner has to follow two rules.

1. Create the block such that it points to the tips of the DAG.
2. Publish the block immediately.

5.2.3 The maximum k-cluster subDAG problem

The maximum k-cluster subDAG problem is the core problem used in the PHANTOM protocol in order to distinguish between honest and malicious blocks.

The intuition behind the maximum k-cluster subDAG problem is the following. In proof-of-work protocols, the number of blocks created in an interval of a specific length is only dependent on the block creation rate. Furthermore, for a given block creation rate, if Δ is an upper bound on the network delay and δ is an error tolerance parameter, by modelling proof-of-work mining as a Poisson process one can calculate an upper bound for the number of blocks created in an interval of length $2 \cdot \Delta$, so that their number is below the upper bound with probability at least $1 - \delta$. This means that for a block B created at some time t we can find a parameter k such that the number of additional blocks created in $[t - \Delta, t + \Delta]$ is upper bounded by k with probability at least $1 - \delta$. With this notion of k, it turns out that the subDAG created by honest miners has a specific structure, and is captured by the notion of a k-cluster.

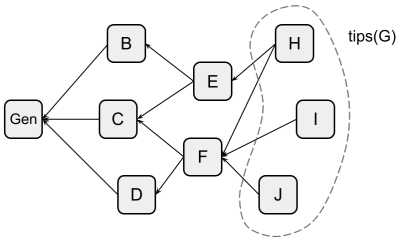
Before we give the definition of a k-cluster, we give the following preliminary definition.

Definition 5.2 ([34])**.** Let $G = (V, E)$ be a blockDAG and $B \in V$. Then,

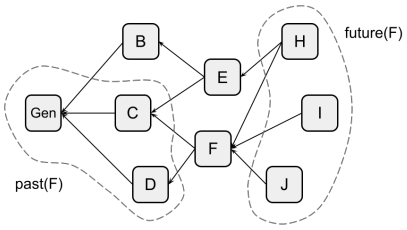
- The *tips* of G are defined as $tips(G) := \{B' \in V \mid deg_{in}(B') = 0\}$, where $deg_{in}(B')$ is the in-degree of $B' \in V$.
- The *past* of B with respect to G is defined as $past_G(B) := \{B' \in V \mid B' \neq B$ and $\exists B - B'$ path in $G\}$.
- The *future* of B with respect to G is defined as $future_G(B) := \{B' \in V \mid B' \neq B$ and $\exists B' - B$ path in $G\}$.
- The *cone* of B with respect to G is defined as $cone_G(v) := past_G(B) \cup \{B\} \cup future_G(B)$.

- The *anticone* of B with respect to G is defined as $anticone_G(B) := V \setminus cone_G(B)$.

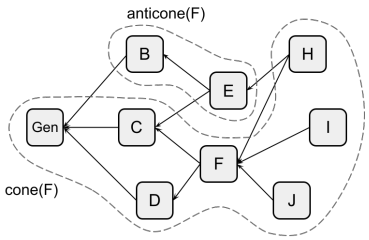
An example of Definition 5.2 is given in Figure 5.3.



(a) $tips(G) = \{H, I, J\}$



(b) $past_G(F) = \{Gen, C, D\}$ and $future_G(F) = \{H, I, J\}$



(c) $cone_G(F) = past_G(F) \cup \{F\} \cup future_G(F) = \{Gen, C, D, F, H, I, J\}$ and $anticone_G(F) = V \setminus cone_G(F) = \{B, E\}$

Figure 5.3: An example of Definition 5.2

Using Definition 5.2 we can define the notion of a k-cluster.

Definition 5.3 ([34])**.** Given a DAG $G = (V, E)$, a subset $S \subseteq B$ is called a *k-cluster* of G if $\forall B \in S, |anticone_G(B) \cap S| \leq k$

The definition of a k-cluster aims to capture the following observation for the graph structure that is formed by honest blocks.

Observation 5.4 ([34])**.** Assume an upper bound Δ on the network delay and assume that a block is mined by an honest miner at time t. Then the block satisfies the following properties.

1. The block should reference directly or indirectly all blocks published before time $t - \Delta$. That is because the miner will have received these blocks at time t.
2. The block should be referenced directly or indirectly by all honest blocks mined after $t + \Delta$. That is because all other miners will have received the block by time $t + \Delta$.

Honest miners are assumed to possess more computational power than malicious miners, and thus create a k-cluster that is larger in size than the malicious miners. Thus, the solution of the following optimization problem contains most honest blocks.

Maximum k-cluster subDAG:
Input: blockDAG $G = (V, E)$
Output: A set $S^* \subseteq V$ of maximum size such that $\forall B \in S^*, |anticone_G(B) \cap S^*| \leq k$.

An example of the maximum 3-cluster in a blockDAG is given in Figure 5.4.

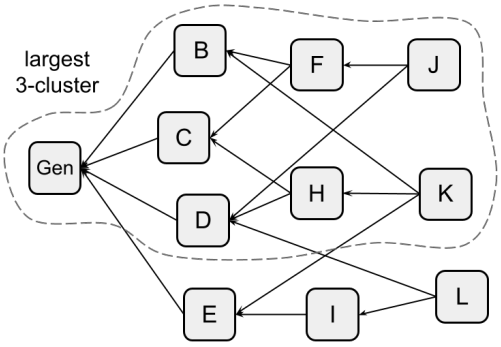


Figure 5.4: An example of the largest 3-cluster in a blockDAG. Example taken from Sompolinsky et al. [34].

For $k = 0$, the maximum 0-cluster subDAG problem is equivalent to finding a longest chain in the DAG. Thus the maximum k-cluster subDAG problem is a generalization of the longest chain rule. However, for $k > 0$, the maximum k-cluster subDAG problem is NP-hard [34], thus PHANTOM cannot be used in practice.

5.2.4 DAG ordering protocol

Since the block creation rate is increased and blocks are created in parallel, different honest miners may create blocks that contain conflicting transactions, thus in order to solve the conflict an ordering between them should be decided. Furthermore, in contrast with protocols such as Bitcoin and GHOST in which a total ordering of transactions can be derived immediately from the ordering of the blocks in the selected chain, an total ordering of transactions in DAG-based protocols is not straightforward.

We first give the definition of an ordering rule of blocks in a blockDAG as described by Sompolinksy et al. [34].

Definition 5.5 (ordering rule, [34])**.** An *ordering rule* ord is an algorithm that takes as input a blockDAG G and outputs a total ordering of blocks $(B_0, B_1, \ldots, B_n)$, where $n = |V|$ and $\{B_1, \ldots, B_n\} = V$.

Transactions in a block can be ordered in the way the appear in the block. Thus, one can extend an ordering rule of blocks to an ordering rule of transactions by first using the ordering rule to order blocks and then ordering transactions inside the blocks by the order they appear. This means that for a DAG-based protocol, an ordering rule of blocks suffices in order to have an ordering rule of transactions.

The ordering rule or PHANTOM is specified as follows. Suppose that S is a solution of the maximum k-cluster subDAG problem in a blockDAG $G = (V, E)$. The ordering

rule orders the blocks of the DAG in a topological order while prioritizing over blocks in the k-cluster.

5.3 The GHOSTDAG Protocol

Since the maximum k-cluster subDAG problem is NP-hard, the GHOSTDAG protocol was created as an optimization of the PHANTOM protocol. The GHOSTDAG protocol uses an approximation algorithm for the maximum k-cluster subDAG problem and thus it can be used in practice.

5.3.1 DAG Ordering protocol

The ordering algorithm of GHOSTDAG is presented in Algorithm 4.

Algorithm 4 The GHOSTDAG ordering algorithm [34].

Input: : block DAG G, anticone parameter k
Output: $BLUE_k(G)$: the Blue set of G; ord: an ordered list containing all blocks in G

1: **function** OrderDAG(G, k)
2: **if** $G == \{genesis\}$ **then**
3: **return** $(\{genesis\}, [genesis])$
4: **end if**
5: **for** $B \in tips(G)$ **do**
6: $(BlueSet_B, OrderedList_B) \leftarrow$ OrderDAG$(past_G(B), k)$
7: **end for**
8: $B_{max} \leftarrow$ argmax$\{|BlueSet_B| : B \in tips(G)\}$ ▷ ties are broken according to lowest hash
9: $BlueSet_G \leftarrow BlueSet_{B_{max}}$
10: $OrderList_G \leftarrow OrderedList_{B_{max}}$
11: add B_{max} to the end of $OrderedList_G$
12: **for** $B \in anticone_G(B_{max})$ **do** in some topological ordering
13: **if** $BlueSet_G \cup \{B\}$ is a k-cluster **then**
14: add B to $BlueSet_G$
15: **end if**
16: add B to the end of $OrderedList_G$
17: **end for**
18: **return** $(BlueSet_G, OrderedList_G)$
19: **end function**

The algorithm takes as input a blockDAG G and the anticone parameter k and outputs $Blue_k(G)$, a set of blocks that are considered honest called *blue set*, and ord, an ordered list of the blocks of the DAG. The algorithm is recursive. For the base case, if the blockDAG contains only the $genesis$ block, a set and a list containing only the genesis block are returned (lines 1-4). For every tip of the blockDAG, the algorithm performs a recursive call to compute the blue set and the ordered list of its past (lines 6-7). The size of the blue set is called the *blue score* of the tip. The algorithm selects the tip with the highest blue score, adds it to the blue set and to the end of the list (lines 9-10). Then, for every block in the anticone of the highest scoring tip (line 12), the

algorithm checks whether adding the block to the blue set preserves the k-cluster property (line 13). If yes, then the block is added to the blue set (line 14). Then, the block is added to the end of the list (line 16).

Essentially, Algorithm 4 calculates an approximation for the size of the maximum k-cluster of $past(B)$, for every block B in the blockDAG. Then, starting from the virtual block, it selects a chain greedily, each time picking the block for which the calculated size of k-cluster in its past is the biggest. Finally, it orders the blockDAG topologically around this chain.

An execution of the iterative version of Algorithm 4 for $k = 3$ is presented in Figures 5.5, 5.6 and 5.7. We present the algorithm as having two procedures, one in which the blue sets and the blue scores are calculated and one in which the total ordering is calculated. In Figure 5.5 we illustrate how the blue score is calculated for each block. Since in every step the highest scoring tip is selected, the selected blocks form a chain, called the *blue chain*. We explain how this chain is selected in Figure 5.6. Having selected the blue chain and assuming the blue set has been calculated for the past of every block, we illustrate how the ordered list is calculated in Figure 5.7.

It is important to note that if we set $k = 0$ in Algorithm 4, the algorithm calculates the longest chain in the blockDAG. Thus, the GHOSTDAG protocol can be seen as a protocol that uses a generalization of the longest chain rule.

Although Algorithm 4 has high computational complexity, in KASPA [40], the implementation of GHOSTDAG, a more efficient version of the algorithm is used [34].

5.3.2 Setting the anticone parameter k

The parameter k is decided before the start of the protocol execution and is hard-coded in the protocol. It is calculated given as input an upper bound on the network delay Δ, a desired block creation rate λ and an error tolerance parameter δ, so that for every block B it holds that $|anticone(B)| \leq k$ with probability at least $1 - \delta$.

More precisely, the calculation of k is described in Equation 5.1.

$$k(\Delta, \lambda, \delta) = \min\{\hat{k} \in \mathbb{N} \mid f(\hat{k}, \Delta, \lambda) < \delta\}, \tag{5.1}$$

where $f(\hat{k}, \Delta, \lambda)$ is defined as

$$f(\hat{k}, \Delta, \lambda) = \max\left\{ \sum_{j=\hat{k}+1}^{\infty} e^{-2\lambda\Delta} \frac{(2\lambda\Delta)^j}{j!}, \frac{2\Delta\lambda}{\hat{k} + 2\Delta\lambda} \right\}. \tag{5.2}$$

Modelling the block creation rate as a Poisson process, if t is the time a block B is created, the term $\sum_{j=\hat{k}+1}^{\infty} e^{-2\lambda\Delta} \frac{(2\lambda\Delta)^j}{j!}$ in Equation (5.2) corresponds to the probability of more than k blocks being created in the interval $[t - \Delta, t + \Delta]$. The term $\frac{2\Delta\lambda}{\hat{k}+2\Delta\lambda}$ comes from the security analysis of GHOSTDAG and is the ratio in which the adversary can reduce the size of the BlueSet.

5.4 Security of the GHOSTDAG protocol

5.4.1 Model

The network consists of nodes (or miners/ parties/ players). There are two types of miners; honest miners, which follow the protocol, and malicious miners that can deviate

arbitrarily. Honest nodes form a connected component in the network's topology. The networks suffers from delay upper bounded by a constant Δ.

Proof-of-work mining is modelled as a Poisson proccess with constant rate λ. The computational power of a miner P is $0 < \alpha_P < 1$ and is the probability that miner P will be the creator of the next block in the system.

5.4.2 Security

In this section we will abuse notation and for a blockDAG $G = (V, E)$ we will write $B \in G$ instead of $B \in V$.

For a player P and a moment in time t, we denote by G_t^P be the blockDAG observed by P at time t. For an ordering rule ord, a blockDAG G and $B, B' \in G$ we write $B \prec_G B'$ if B' comes before B in the output of ord with input the blockDAG G. If $B \in G$ but $B' \notin G$ we will also write $B \prec_G B'$.

Definition 5.6 ([34])**.** Fix an ordering rule ord, a time $t > 0$, a time length $r > 0$ and a player P. For a block B, the *risk* of the player P for the block B at time t and for the time length r is defined as follows.

- If $B \in G_t^P$,

$$Risk_P(B, t, r) := Pr[\exists s > t + r, \exists B' \in G_s^P : B \prec_{G_{t+r}^P} B' \text{ and } B' \prec_{G_s^P} B] \tag{5.3}$$

- If $B \notin G_t^P$,

$$Risk_P(B, t, r) := 1$$

The function $Risk$ for the block B at time t and for the time length r is defined as

$$Risk(B, t, r) := \max_{P: P \text{ is honest}} Risk_P(B, t, r) \tag{5.4}$$

Equation 5.3 is the probability that, for a block B which is in the DAG that is observed by a player P at time t, there exists a block B' that does not precede B in the ordering at time $t + r$ but at a time later than $t + r$ will precede it. The function $Risk$ in equation 5.4 is defined as the maximal risk over all honest players.

Definition 5.7 ([34])**.** For $\beta > 0$, an ordering rule ord is said to $(1 - \beta)$-*converge* if for every $t > 0$, every honest player P and every $B \in G_t^P$,

$$\lim_{r \to \infty} Risk(B, t, r) = 0. \tag{5.5}$$

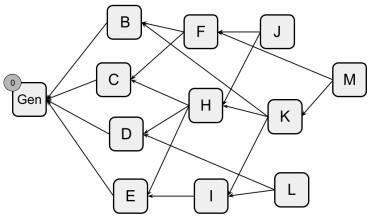
even if a fraction β of the mining power is adversarial.

The convergence property of an ordering rule of Definition 5.7 couples the Safety and Liveness properties required for the security of distributed ledger protocols [34]. If for a block B at time t for some time length r, $Risk(B, t, r) < \varepsilon$, then a transaction inside a block is guaranteed to be irreversible, and a decision to accept this transaction can be made, up to an error probability of ε.

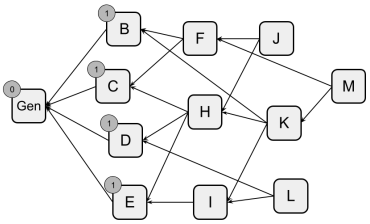
Definition 5.8 ([34])**.** The *security threshold* of an ordering rule ord is defined as the maximal $\beta > 0$ for which it $(1 - \beta)$-converges exponentially fast.

The security of GHOSTDAG is given by the following theorem.

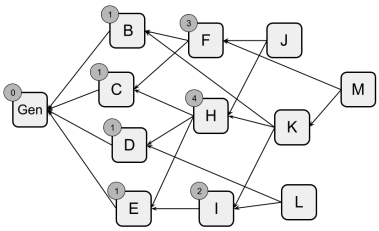
Theorem 5.9 ([34])**.** *Given a block creation rate $\lambda > 0$, an error tolerance parameter $\delta > 0$ and an upper bound on the network delay $\Delta > 0$, the security threshold of GHOSTDAG, parameterized with $k(\Delta, \lambda, \delta)$, is lower bounded by $\frac{1}{2} \cdot (1 - \delta)$.*



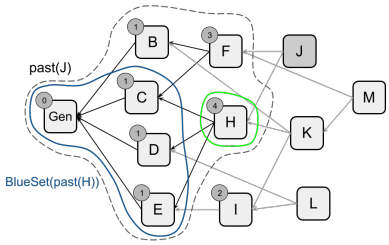
(a) The genesis block has empty past, so it has a score of 0.



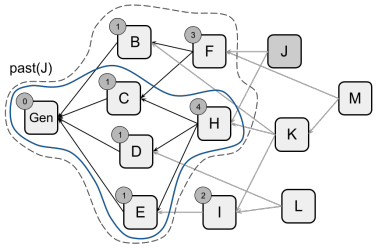
(b) The set $\{Gen\}$ is a 3-cluster, so the score of B, C, D and E is set to 1.



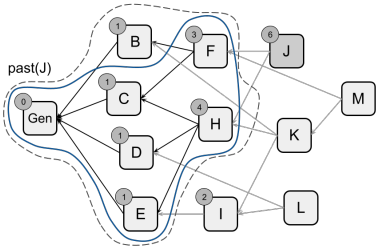
(c) For blocks F, H and I their past set is a 3-cluster, so their score is equal to the size of their past set.



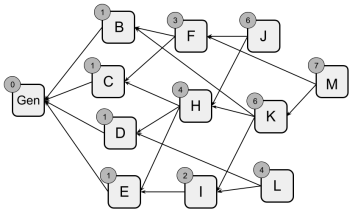
(d) To calculate the score of block J, first H, the tip with the highest score in its past, is selected. In this step, $BlueSet(past(J)) = BlueSet(past(H)) = \{Gen, C, D, E\}$.



(e) Block H is added to $BlueSet(past(J)$. Now $BlueSet(past(J))= \{Gen, C, F, E, H\}$.

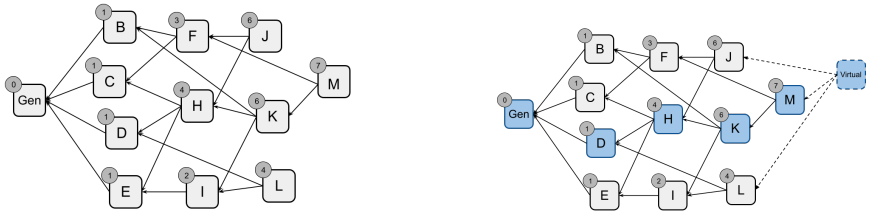


(f) Block B cannot be added to the blueSet since the set would not be a 3-cluster. Then block F is addedd to the blueSet. $BlueSet(past(J)) = \{Gen, C, F, E, F, H\}$ and $score(J) = 6$.



(g) Blue score is calculated analogously for blocks K, L and M.

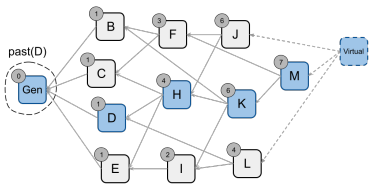
Figure 5.5: Blue score calculation in GHOSTDAG for $k = 3$. The score of each block is equal to the size of the blue set of its past. Example taken and mofidied from Sompolinsky et al. [34].



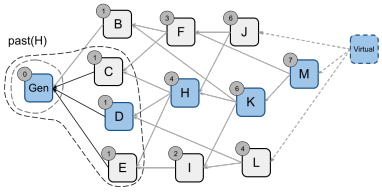
(a) Blue score is calculated for each block.

(b) A virtual block is added that points to the tips of the DAG. Then the Blue Chain is selected greedily, starting from the virtual block and selecting each time the block with the higher blue score.

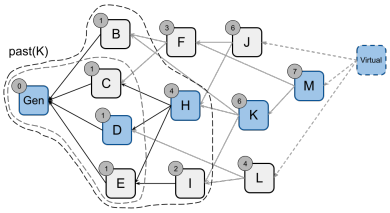
Figure 5.6: Chain selection in GHOSTDAG. Example taken and mofidied from Sompolinsky et al. [34].



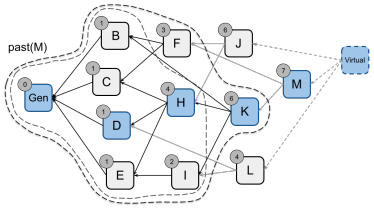
(a) $past(D)$ is ordered. In this step, $OrderderList = [Gen]$.



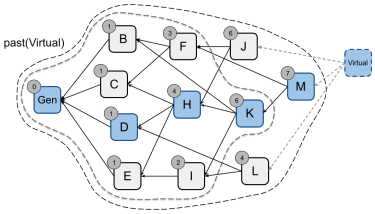
(b) $past(H)$ is ordered. In this step, $OrderedList = [Gen, D, C, E]$.



(c) $past(K)$ is ordered. In this step, $OrderedList = [Gen, D, C, E, H, I, B]$.



(d) $past(M)$ is ordered. In this step, $OrderedList = [Gen, D, C, E, H, I, B, K, F]$.



(e) $past(Virtual)$ is ordered. In this step, $OrderedList = [Gen, D, C, E, H, I, B, K, F, M, J, L]$.

Figure 5.7: Ordering of GHOSTDAG after the chain selection. Starting from the genesis block, the past of every block in the blue chain is ordered. In every step the highest scoring tip is added to $OrderedList$, and then blocks in its anticone are added to $OrderedList$ in topological order. Example taken and modified from Sompolinsky et al. [34].

CHAPTER 6

OTHER DAG-BASED PROTOCOLS

6.1 The Conflux protocol

The Conflux protocol [19] is a proof-of-work DAG-based protocol that uses the GHOST rule in order to achieve high throughput.

Similarly to the PHANTOM and GHOSTDAG protocols, blocks point to all unreferenced blocks the miner observed at the time of creation. However, two types of edges are used.

- *parent edges*. Each block except the *genesis* block has exactly one outgoing parent edge. As a result, the subgraph that contains only the parent edges is a tree.
- *reference edges*. Each block can have multiple outgoing reference edges.

The mining rules are as follows.

1. The new block should reference with a parent edge the tip of the GHOST chain of the tree composed by parent edges.
2. The new block should point to all other tips of the DAG with reference edges.
3. Upon being successful in mining, the miner should broadcast the block immediately.

The ordering of blocks is calculated as follows. First, the GHOST chain of the tree composed by parent edges, which is called *pivot* chain, is calculated. Let $B_1, \ldots, B_n$ be the blocks of the pivot chain. Then, a partition of the blocks into n sets $S_1, \ldots, S_n$ is made, where $S_i = \{B_i\} \cup \{B' \mid B' \text{ is not in the pivot chain and } B' \notin S_j, \forall j < i\}$.. The set S_i is called the *epoch* of block B_i. Blocks of each epoch are sorted topologically, breaking ties according to the lowest hash. If L_i is the ordered list of the blocks in S_i, the total ordering of blocks is the concatenation of all lists. That is, $L = L_1 || \cdots || L_n$. Finally, as in PHANTOM and GHOSTDAG, transactions inside a block are ordered in the way they appear in the block, and thus a total ordering of transactions results by ordering the blocks, then ordering the transactions inside the blocks, and then in case of conflicting transactions removing all but the first one in the ordering.

An example of a blockDAG created using the mining rules of Conflux is given in Figure 6.1.

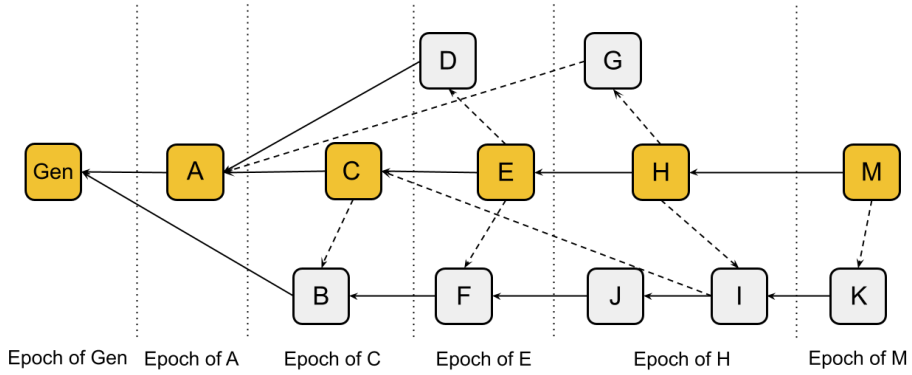


Figure 6.1: An example of a blockDAG in the Conflux protocol. Solid arrows represent parent edges, and dashed arrows represent reference edges. The pivot chain is shown in yellow. Blocks are added to their corresponding epoch as shown in the figure. The total ordering of blocks is $L = [Genesis, A, C, B, E, D, F, H, I, J, G, M, K]$. Example taken and modified from Li et al. [19].

The security of Conflux depends on the security of GHOST. Since GHOST is vulnerable to the balance attack [9], Conflux uses a variant of GHOST called Greedy Heaviest Adaptive SubTree (GHAST) designed to avoid the balance attack [28, 27].

6.2 The SPECTRE protocol

The SPECTRE protocol [10] is a proof-of-work DAG-based protocol that achieves high throughput but provides only a pairwise ordering of transactions. For this reason, SPECTRE can be used for payments but not for smart contract applications. The SPECTRE protocol guarantees *weak liveness*, meaning that the pairwise ordering is robust only for transactions for which no conflicting transaction is published for a short period of time. For honest users of the protocol this property suffices, since they will not publish conflicting transactions.

The mining rules of SPECTRE are identical to those of PHANTOM and GHOSTDAG presented in Chapter 5. When mining a block, the block should reference all the tips of the DAG the miner observes and upon successful mining, the miner should broadcast the block immediately.

Let $G = (V, E)$ be a blockDAG and assume that in blocks $B_1, B_2 \in V$ two conflicting transactions appear. As mentioned previously, SPECTRE provides a partial ordering of transactions. This is acquired by providing a partial ordering of blocks, namely, by making possible to decide whether $B_1 \prec B_2$ or $B_2 \prec B_1$. This is done using the following procedure.

Every block $B \in V$ is considered as a *voter*, whose vote is decided based on the structure of the DAG. Specifically, the vote of block B for the pair B_1, B_2 with respect

to the structure of the blockDAG G is

$$vote_{B_1,B_2}(B,G) = \begin{cases} -1, & \text{which represents } B_1 \prec B_2 \\ 0, & \text{which represents a tie} \\ +1, & \text{which represents } B_2 \prec B_1 \end{cases} \tag{6.1}$$

Let also $virtual(G)$ be a hypothetical block as in Chapter 5 such that $past(virtual(G)) = G$. The hypothetical block $virtual(G)$ is called the *virtual* block and it is associated with a vote as well. Specifically, $vote_{B_1,B_2}(virtual(G), G)$ is equal to the majority of votes in V.

Using Definition 5.2, the rules to calculate the value of $vote_{B_1,B_2}(B,G)$ for a block $B \in V$ are as follows.

1. If $B \in future(B_1) \setminus future(B_2)$ then $vote_{B_1,B_2}(B,G) = -1$. That is, if block B is in $future(B_1)$ but not in $future(B_2)$, B votes in favor of B_1. Similarly, if $B \in future(B_2) \setminus future(B_1)$ then $vote_{B_1,B_2}(B,G) = 1$.

2. If $B \in future(B_1) \cap future(B_2)$ then $vote_{B_1,B_2}(B,G) = vote_{B_1,B_2}(virtual(past(B)), G)$. That is, if B is in both $future(B_1)$ and $future(B_2)$, it votes as the majority of blocks in its past.

3. If $B \notin future(B_1) \cup future(B_2)$, B votes as the majority of blocks in $future(B)$.

4. If $B = B_1$ then $vote_{B_1,B_2}(B,G) = -1$, that is, block B_1 votes for itself. Similarly, if $B = B_2$ then $vote_{B_1,B_2}(B,G) = 1$.

Finally, if $\sum_{B \in V} vote_{B_1,B_2}(B,G) < 0$ then it is decided that $B_1 \prec B_2$, and if $\sum_{B \in V} vote_{B_1,B_2}(B,G) > 0$ then it is decided that $B_2 \prec B_1$.

Concerning ties, they are broken using some deterministic rule, for example in favor of the block with the lowest hash or other information inside the block header.

An example of the voting procedure on a pair of blocks B_1, B_2 is illustrated in Figure 6.2.

Using the above pairwise ordering procedure, a set TxO of accepted transactions is extracted from the blockDAG. A transaction included in a block B_1 is considered *accepted* if the following hold.

1. All of its inputs[1] have been accepted.

2. For any conflicting transaction included in a block $B_2 \in anticone(B_1)$ it holds that $B_1 \prec B_2$.

3. Any conflicting included in a block $B_2 \in past(B_1)$ has been rejected.

Finally, a set $RobustTxO$ of *robustly accepted* transactions is extracted from TxO. The set $RobustTxO$ contains the transactions of TxO that are guaranteed to be accepted forever with an error probability of ϵ, and may not include all the transactions of TxO.

The SPECTRE protocol satisfies the following security properties for a 50%-bounded adversary.

[1] The *inputs* of a transaction refer to the UTXO transaction model. See the Bitcoin whitepaper [2] for more details on this topic.

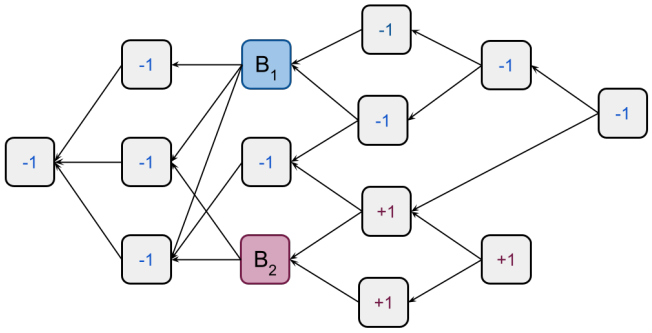


Figure 6.2: An example of the voting procedure on pair of blocks B_1, B_2 in a blockDAG $G = (V, E)$. For $B \in V$, $vote_{B_1,B_2}(B, G) = -1$ if $B_1 \prec B_2$ and $vote_{B_1,B_2}(B, G) = +1$ if $B_2 \prec B_1$. We also have that $vote_{B_1,B_2}(B_1, G) = -1$ and $vote_{B_1,B_2}(B_2, G) = +1$. We have that $\sum_{B \in V} vote_{B_1,B_2}(B, G) < 0$, thus it is decided that $B_1 \prec B_2$. Example taken and modified from Sompolinsky et al. [10].

Definition 6.1 (Consistency [10], simplified form)**.** If a transaction is included in TxO by some honest party then all of its inputs have also been included, and there is no conflicting transaction included.

Definition 6.2 (Safety [10], simplified form)**.** If a transaction is included in $RobustTxO$ by some honest party, then with high probability it will be included in $RobustTxO$ by all honest parties, and the expected waiting time for this event is constant.

Definition 6.3 (Weak Liveness [10], simplified form)**.** If a transaction is published in the ledger, it included in $RobustTxO$ by any honest party after a short while, provided that its inputs are also included in $RobustTxO$ and that no conflicting transactions are published.

CHAPTER 7

CONCLUSION

7.1 Discussion

In this work we presented performance solutions to Bitcoin that use alternative chain selection rules or a Directed Acyclic Graph as their underlying data structure.

We first presented the Bitcoin protocol and discussed its security proofs. Then, we presented the GHOST rule, which instead of the longest chain selects greedily each time the block that is the root of the biggest subtree, and we discussed the arguments and proofs for its security. Finally, we presented our contribution, an attempt for an alternative proof of an already known upper bound on the adversarial power tolerance of GHOST using a Common Prefix technique, that is a reconstruction of previous security proofs.

Regarding DAG-based distributed ledgers, we presented the PHANTOM protocol and its optimization GHOSTDAG, that allow multiple blocks to be created in parallel and use information of the structure of the DAG created to provide a total ordering of transactions. We also presented the main security properties of GHOSTDAG. Finally, we presented the mechanisms of SPECTRE and Conflux protocols and discussed their security guarantees.

Regarding attacks on the aforementioned protocols, the secret mining attack is known to apply to Bitcoin since its construction [2], and it is exactly the attack that prevents the protocol from satisfying consistency if the tight bound is not satisfied [25]. The GHOST protocol was designed to avoid the secret mining attack of Bitcoin, however, it does not avoid all the attacks applicable to Bitcoin since it is vulnerable to the balancing attack. The GHOSTDAG protocol can be seen as a fix of the GHOST rule [34] and the Conflux protocol that initially relied on the GHOST rule now uses a variation of it, GHAST, that avoids the balancing attack [27]. Furthermore, a liveness attack for a previous variant of GHOSTDAG was found [19], which we did not include in this work.

We conclude that in many cases in which a protocol is proposed as a scalability solution to Bitcoin and is designed to avoid a particular attack, another attack appears that affects its security and makes the protocol non-scalable. Furthermore, when arguing about the security of a protocol, the assumptions on the capabilities of the adversary should be as close to real conditions as possible and the definitions of the required se-

curity properties should reflect the real-world requirements. For example, although a security analysis of GHOST was present in the initial paper, the security analysis did not consider the possibility that the adversary could send different blocks to different parties, which is something that can happen in the real-world execution, and is exactly the reason the balance attack succeeds.

7.2 Open problems

There are many problems left open concerning the security of the protocols we presented. Some of the open problems stated in the works we presented are a proof for a tight consistency bound of Bitcoin in the variable difficulty setting [25], light client constructions for GHOST [7], a security analysis of PHANTOM [34], research on tighter methods to calculate the anticone parameter in GHOSTDAG [34] and a reformulation of the security proof of GHOSTDAG that includes Safety and Liveness properties [34].

In the direction of light clients for GHOST, NIPoPoW protocols for Bitcoin [33] could potentially be modified to work for GHOST and their security proof could be based on a modification of our proof attempt presented in Chapter 4. Regarding the security analysis of PHANTOM, a possible approach would be to compare its security to some other DAG-based protocol for which its security is already examined, such as the Parallel Chains protocol [17], using some appropriate mapping.

Directions for further research that we identify in the works we presented are the following. Regarding GHOST, a tight consistency bound could potentially be achieved using the techniques of Gaži et al. for Bitcoin [25] and liveness bounds for GHOST in the bounded delay setting could potentially be achieved by transferring the Fresh Block lemma [11] to the bounded delay setting. We strongly believe that the already known bounds for GHOST can be improved and a comparison of GHOST and Bitcoin could be made. Regarding PHANTOM and GHOSTDAG, a formal proof for the NP-hardness of the maximum k-cluster subDAG problem is needed, and it could also be useful to examine whether there is any tradeoff in security when using an approximation algorithm for an NP-hard problem.

BIBLIOGRAPHY

[1] Sven Erick Alm. ``Simple random walk''. In: *Unpublished manuscript (2002). http://www2. math. uu. se/~ sea/kurser/stokprocmn1/slumpvandring_eng. pdf* (2002).

[2] Satoshi Nakamoto. ``Bitcoin: A peer-to-peer electronic cash system''. In: *Decentralized Business Review* (2008), p. 21260.

[3] Christian Decker and Roger Wattenhofer. ``Information propagation in the bitcoin network''. In: *IEEE P2P 2013 Proceedings*. IEEE. 2013, pp. 1–10.

[4] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. ``The bitcoin backbone protocol: Analysis and applications''. In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2015, pp. 281–310.

[5] Aggelos Kiayias and Giorgos Panagiotakos. ``Speed-security tradeoffs in blockchain protocols''. In: *Cryptology ePrint Archive* (2015).

[6] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. ``Inclusive block chain protocols''. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2015, pp. 528–547.

[7] Yonatan Sompolinsky and Aviv Zohar. ``Secure high-rate transaction processing in bitcoin''. In: *International conference on financial cryptography and data security*. Springer. 2015, pp. 507–527.

[8] Jing Chen and Silvio Micali. ``Algorand''. In: *arXiv preprint arXiv:1607.01341* (2016).

[9] Christopher Natoli and Vincent Gramoli. ``The balance attack against proof-of-work blockchains: The R3 testbed as an example''. In: *arXiv preprint arXiv:1612.09426* (2016).

[10] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. ``Spectre: A fast and scalable cryptocurrency protocol''. In: *Cryptology ePrint Archive* (2016).

[11] Aggelos Kiayias and Giorgos Panagiotakos. ``On trees, chains and fast transactions in the blockchain''. In: *International Conference on Cryptology and Information Security in Latin America*. Springer. 2017, pp. 327–351.

[12] Aggelos Kiayias et al. ``Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *Annual international cryptology conference*. Springer. 2017, pp. 357–388.

[13] Christopher Natoli and Vincent Gramoli. ``The balance attack or why forkable blockchains are ill-suited for consortium". In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2017, pp. 579–590.

[14] Rafael Pass, Lior Seeman, and Abhi Shelat. ``Analysis of the blockchain protocol in asynchronous networks". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2017, pp. 643–673.

[15] Christian Badertscher et al. ``Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 913–930.

[16] Bernardo David et al. ``Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2018, pp. 66–98.

[17] Matthias Fitzi et al. ``Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition". In: *Cryptology ePrint Archive* (2018).

[18] Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. ``A better method to analyze blockchain consistency". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 729–744.

[19] Chenxing Li et al. ``Scaling nakamoto consensus to thousands of transactions per second". In: *arXiv preprint arXiv:1805.03870* (2018).

[20] Vivek Bagaria et al. ``Prism: Deconstructing the blockchain to approach physical limits". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 585–602.

[21] Shehar Bano et al. ``SoK: Consensus in the age of blockchains". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. 2019, pp. 183–198.

[22] Bitcoin Cash. ``Bitcoin cash". In: *Development* 2 (2019).

[23] Ling Ren. ``Analysis of nakamoto consensus". In: *Cryptology ePrint Archive* (2019).

[24] Dan Boneh and Victor Shoup. ``A graduate course in applied cryptography". In: *Draft 0.5* (2020).

[25] Peter Gaži, Aggelos Kiayias, and Alexander Russell. ``Tight consistency bounds for bitcoin". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 819–838.

[26] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[27] Chenxing Li, Fan Long, and Guang Yang. ``GHAST: Breaking confirmation delay barrier in nakamoto consensus via adaptive weighted blocks". In: *arXiv preprint arXiv:2006.01072* (2020).

[28] Chenxing Li and Guang Yang. ``Conflux protocol specification". In: (2020).

[29] Jing Li and Dongning Guo. ``Continuous-time analysis of the bitcoin and prism backbone protocols". In: *arXiv preprint arXiv:2001.05644* (2020).

[30] Wellington Fernandes Silvano and Roderval Marcelino. ``Iota Tangle: A cryptocurrency to communicate Internet-of-Things data". In: *Future Generation Computer Systems* 112 (2020), pp. 307–319.

[31] Jun Zhao et al. ``An analysis of blockchain consistency in asynchronous networks: Deriving a neat bound". In: *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2020, pp. 179–189.

[32] Iddo Bentov et al. ``Tortoise and hares consensus: the meshcash framework for incentive-compatible, scalable cryptocurrencies". In: *International Symposium on Cyber Security Cryptography and Machine Learning*. Springer. 2021, pp. 114–127.

[33] Aggelos Kiayias, Nikos Leonardos, and Dionysis Zindros. ``Mining in Logarithmic Space". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 3487–3501.

[34] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. ``PHANTOM GHOSTDAG: a scalable generalization of Nakamoto consensus: September 2, 2021". In: *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. 2021, pp. 57–70.

[35] George Danezis et al. ``Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus". In: *Proceedings of the Seventeenth European Conference on Computer Systems*. 2022, pp. 34–50.

[36] Tatsuya Yanagita et al. ``Space-Efficient Data Structure for Posets with Applications". In: *18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2022.

[37] *Bitcoin wiki*. url: `https://en.bitcoin.it/wiki/`.

[38] *Ethereum whitepaper*. url: `https://ethereum.org/el/whitepaper/`.

[39] William Feller. ``An introduction to probability theory and its applications". In: *1, 2nd* ().

[40] *Kaspa github repository*. url: `https://github.com/kaspanet/kaspad/`.

[41] *Monero Project*. url: `https://www.getmonero.org/`.