A Formalization & Attack Tree for Gasper: Ethereum's Proposal for Proof-of-Stake Consensus

Candidate no. 1051618

Thesis word count: 16,009

A thesis submitted for the degree of Master of Science

Michaelmas 2022

Abstract

Blockchain applications are becoming more prominent at a growing pace, and the stake-based consensus is being adopted by the majority of the current networks. Stake-based blockchains already stepped into many mission-critical sectors such as transportation, health, and defense. It is crucial that security research catches hold of the development. This article introduces Proof-of-Stake consensus protocols and formalizes a specific instance: the "Gasper" protocol that is used to convert the widely used Ethereum into a stake-based network. Taking Gasper as the status quo, the article explores a range of straightforward and more sophisticated ways of challenging the protocol. The identified attack scenarios are then combined in an attack tree which helps our illustration and comparison purposes. We determine that attacking a blockchain application vastly differs for different consensus paradigms. Most work-based attacks are not applicable to stake-based networks and vice versa. The attack space is also significantly affected by the specifics of the consensus algorithms. Overall, we attest that Gasper has a considerably smaller attack space than a general work-based blockchain with the longest chain fork-choice rule.

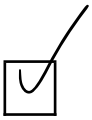
A Formalization & Attack Tree for Gasper: Ethereum's Proposal for Proof-of-Stake Consensus

Candidate no. 1051618

Word count: 16,009

A thesis submitted for the degree of

Master of Science



I hereby certify that this is entirely my own work unless otherwise stated.

Michaelmas 2022

Acknowledgements

Foremost, I would like to acknowledge and give my warmest thanks to my supervisors Prof Michael Goldsmith and Dr Louise Axon. Their continuous support, valuable guidance and seasoned advice carried me through the process of writing my dissertation.

Besides my supervisors, I would like to thank my family for their incredible understanding and moral support.

Finally, I would like to thank all my friends and colleagues who in one way or another shared their support.

Abstract

Blockchain applications are becoming more prominent at a growing pace, and the stake-based consensus is being adopted by the majority of the current networks. Stake-based blockchains already stepped into many mission-critical sectors such as transportation, health, and defense. It is crucial that security research catches hold of the development. This article introduces Proof-of-Stake consensus protocols and formalizes a specific instance: the "Gasper" protocol that is used to convert the widely used Ethereum into a stake-based network. Taking Gasper as the status quo, the article explores a range of straightforward and more sophisticated ways of challenging the protocol. The identified attack scenarios are then combined in an attack tree which helps our illustration and comparison purposes. We determine that attacking a blockchain application vastly differs for different consensus paradigms. Most work-based attacks are not applicable to stake-based networks and vice versa. The attack space is also significantly affected by the specifics of the consensus algorithms. Overall, we attest that Gasper has a considerably smaller attack space than a general work-based blockchain with the longest chain fork-choice rule.

Contents

List of Figures

1 Introduction

Contents

1.1 Aim

This article has two major aims. The first is to introduce Proof-of-Stake (PoS) consensus protocols and formalize a specific instance: the "Gasper" protocol that is used to convert the widely used Ethereum into a stake-based network. We define the building blocks of Gasper under a mathematical framework and analyze the properties of the protocol to truly understand the key components affecting the security guarantees of PoS networks.

Second, we will create an attack tree for the Gasper protocol, originating from a given attack tree for PoW networks. We determine the paths of the tree, i.e., attack vectors, that are only applicable to PoW networks and identify those that specifically arise in PoS networks. We will systematically compare the origin tree with the resulting tree to weigh the two consensus paradigms in terms of certain security metrics.

1.2 Motivation

Blockchain applications are stepping into our daily lives at a growing pace. We recently saw a significant portion of the global financial system occupied by cryptocurrencies. The defense sector started using blockchains to store confidential data securely. Medical records are also stored in blockchains to mitigate integrity and single point of failure issues while guaranteeing transparency.

The pace of security research on the blockchain should match the pace at which we incorporate the technology. Otherwise, drastic side effects would follow. Using a blockchain that is weak in terms of availability in global trade would mean that an attacker could arbitrarily interrupt the exchange of critical goods and services. Similarly, using a blockchain that is not well tested in terms of integrity to store military records would mean that a malicious party could tamper with the records and generate false flags. Hence, security research is not an option, it is not even a need, it is a must. We, therefore, explore the most prominent type of blockchains, PoS networks, comprehensively in terms of security guarantees and vulnerabilities.

1.3 Contribution

Our contributions, in this article, are:

- *Formalizing Gasper* in our own terms by simplifying certain notions in the original Gasper paper, and introducing new notation when we feel the need to do so. We put forward a summary of Gasper such that the definitions follow each other in a progressive manner leading to the final protocol.

- *Building a Gasper-specific Attack Tree.* We build an attack tree that enumerates attack vectors, i.e., ways of attacking the blockchain application backed by the Gasper protocol. We present a semantic categorization of attack types in our tree, as well as specific exploits needed to advance the attacks. The tree also includes countermeasures to certain attacks. We later compare this tree to a general PoW Attack Tree to identify attacks specific to the consensus paradigm and evaluate the paradigms in conjunction with each other.

- *Explore the State-of-the-art Attacks on PoS Networks.* The article can be viewed as a comprehensive exploration of attack possibilities on PoS networks and known defenses. Hence, it includes descriptions of the state-of-the-art attacks exploiting specific vulnerabilities of the underlying consensus algorithms. We reviewed the large set of attacks on PoS-networks and, specifically, Gasper; vectorized each of them, and put together a summarizing attack tree.

2 Background

Contents

2.1 Background

A blockchain is essentially a distributed ledger, a database that is consensually shared among multiple network participants. Cryptographic primitives such as hash functions, digital signatures, and distributed consensus protocols ensure that a record entered into the ledger can only be altered with the consensus of the entire network. Consequently, data in the distributed ledger can be verified in a decentralized manner, and blockchain applications can operate in trustless environments.

The distributed consensus protocols constitute the backbone of a blockchain network. It dictates the network's security in terms of the ledger's integrity and resiliency and the network's performance in terms of speed and energy efficiency. The very first blockchain application was introduced in "Satoshi Nakamoto"'s famous cryptocurrency paper "Bitcoin: A Peer-to-Peer Electronic Cash System"

[1] and proposed the proof-of-work (PoW) consensus protocol. Therefore, the vast majority of the current blockchain networks rely on the PoW consensus protocol. However, the PoW protocol includes an intensive mining process that leads to several limitations such as energy inefficiency, network delays, and certain security vulnerabilities. A newer consensus protocol, proof-of-stake (PoS) has been proposed to overcome the limitations of PoW.

The PoS consensus protocol achieves consensus by requiring the network participants to stake capital, i.e., deposit digital tokens, the staked capital acts as collateral that incentivizes the participants to behave honestly. Many of the new blockchain applications are deploying the PoS protocol and many existing blockchain applications are mitigating to it because of the substantial efficiency and security improvements. The PoS protocol is expected to be the foundational building block for future blockchain networks and applications.

2.1.1 Consensus Protocols

In distributed environments network participants must agree on the state of the network to be able to act deterministically and simultaneously. Otherwise, the participants can get involved in faulty or malicious behaviors, resulting in a corrupt network. The consensus protocol is therefore vital for a reliable blockchain network in such trustless environments. The protocol incentivizes network participants to behave properly as well as governs how the transactions are added to the distributed ledger. We will now briefly explore the PoW and PoS consensus protocols in comparison with each other.

Proof-of-Work Consensus

The proof-of-work consensus protocol is foundational to early blockchain networks and applications due to its reputation gained with Bitcoin. The participants, i.e., the nodes in a PoW network reach consensus by participating in a puzzle-solving process in which a node competes against others to find a *nonce* (a number that is used once) for the candidate block it proposes. More precisely, the puzzle is

the problem of finding a special number, a nonce, such that the nonce together with the readily available previous block's hash and transactions contained in the candidate block produces an output contained in a target range when given as input to a secure hash function, e.g., SHA-256. If a node succeeds to find a nonce that maps the hash output to the target range, then its candidate block is accepted and broadcasted to the network. This process is also known as the mining of a block. If the broadcasted block is verified to be the first block mined after the last block in the chain, then it is attached to the chain as the new last block.

Keeping a consensual and distributed database is still not that straightforward. Network delays may cause several versions, i.e., forks of the ledger due to two or more miners finding a suitable candidate block simultaneously. This possibility becomes negligible as the mined blocks go deeper in the blockchain, the probability chain rule ensures that the possibility of two valid forks of the ledger existing simultaneously decreases exponentially with the chain length. This result simply follows from the fact that each puzzle-solving attempt is an independent weighted random coin tossing process. Consequently, a mined block is finalized only when it is k, typically six, blocks deep in the blockchain. This requirement ensures there can only exist one main, canonical, chain, but delays the transaction confirmation dramatically.

Due to the non-reversibility property of the hash function, the only way of solving the nonce puzzle is to perform a brute-force search that exhaustively tries a different number until the hash output is within the desired range. Thus, the only factor contributing to the solution searching process is the hash rate of the participants, i.e., the computational power they possess. The probability of participant i finding a solution to the puzzle is simply the ratio of the participant's hash rate to the total hash rate in the network.

$$p_i = \frac{c_i}{\sum_{j=1}^{N} c_j} \tag{2.1}$$

The PoW protocol places a block reward on the puzzle, the block winner also called the leader, i.e., the participant solving the puzzle, receives a reward backed by the network currency. The reward amount is adjusted in a way that it is in the

participant's best interest to participate in the puzzle-solving process. However, a huge drawback of the PoW protocol is that the participants try to increase their hash rates as it is linearly correlated to their chance of winning the block reward, this, in turn, leads to elevated levels of energy consumption which hurts the environment as well as network scalability.

Another problem with the PoW protocol is the formation of mining pools. A mining pool is the collection of computing resources provided by network participants who want to get higher opportunities to win block rewards by collaboratively solving the nonce puzzles. Mining pools guarantee more stable incomes to their participants due to the increased chances of mining a block. However, they threaten the decentralization premise of blockchain technology. Up to 62.7% of the total hash rate of the entire Bitcoin network is held by the largest five mining pools [2].

One of the most significant vulnerabilities of the PoW protocol is the chance of a 51% attack. In such a scenario, a single party controlling more than 51% of the total hash rate can maliciously alter the chain by launching double spending attacks or avoiding transactions submitted by other participants. The 51% attack is neglectable for large networks, but they need to be taken into consideration as centralized entities such as mining pools grow larger within the networks. Small and newly established networks should particularly take this attack into consideration.

Proof-of-Stake Consensus

The proof-of-stake (PoS) consensus protocols are energy-saving alternatives to PoW consensus protocols. The name of the protocol is self-explaining. The fundamental difference between the two types of protocols is that in PoS the leader selection process is dictated by the stakes contributed by the participants instead of the computational work they have performed. The stake of a node is the number of tokens, i.e., the native currency of the network, which the node deposits upfront. The leader is, again, the participant that mines the new block and appends it to the chain, and it is selected based on the stakes.

The Follow-the-Satoshi (FTS) algorithm is, generally, used in the leader selection process. The FTS algorithm is essentially a hash function that takes a seed, typically pseudo-random, and outputs a token index. Given the output token index, the algorithm searches the deposit history and finds the owner of the output token. The owner is set to be the leader and awarded with the block reward as in the PoW protocol. However, the probability of participant i winning the block reward is now associated with the proportion of the total stakes it holds. The more stakes a participant has, the higher chance it has of winning the block reward.

$$p_i = \frac{s_i}{\sum_{j=1}^{N} s_j} \tag{2.2}$$

The PoS protocol implements an incentive mechanism that combines rewards and penalties to incentivize consensus participation by means of staking while avoiding adversarial behaviors. The protocol penalizes the adversarial participant by taking away his stakes when malicious behavior is detected. The incentive mechanism is a crucial element of any PoS protocol, it allows analyzing network security under a mathematical framework by means of game theory. A PoS-based network is not secure without an incentive mechanism with a provably secure equilibrium.

The transaction confirmation speed is another strength point of the PoS protocol over the PoW protocol. The transaction confirmation depends on the transaction throughput and the block confirmation time. Transaction throughput, Tx/s, is the number of transactions processed in the network per second. It is calculated by dividing the number of transactions contained in a block by the block time, which is the average time needed to increase the chain length by one.

$$Tx/s = \frac{Block_{size}}{Tx_{size} \times Block_{time}} \tag{2.3}$$

The Tx/s dictates how fast transactions are added to the ledger. The block confirmation time, on the other hand, dictates how fast the added transactions are being finalized. It depends on the block time and the finality of the protocol. The finality is, in general, the depth in terms of blocks a transaction must go in the chain to be finalized, different protocols may have different finality notions.

Block time is much shorter in PoS-based networks as a deterministic selection process replaces the puzzle-solving competition, also PoS-based networks typically have a larger block size, thus the transaction throughput is much higher in PoS than in PoW. Most PoS-based networks achieve immediate finality by voting to confirm a block after each round of block addition, which means that the transactions are finalized right after the respective block is added to the chain. Other PoS-based networks implement variants of the longest chain rule which maintains forks of the ledger up to a certain depth, finality is delayed in such networks so they have a longer block confirmation time.

Another factor affecting the security of PoS-based networks is network synchrony. The leader selection process is oftentimes implemented using sub-processes such as voting and message passing. These sub-processes are fallible due to network delays and network congestion. Some networks provide results showing that they are secure under relaxed synchrony requirements, such as partial synchrony or asynchrony.

2.1.2 Attacks

We will briefly touch on the most common attacks that must be considered when building secure blockchain structures. Most of the following attacks are used as sub-processes within more sophisticated attacks. Describing these common attacks will provide an insight into the weaknesses and vulnerabilities of different blockchain types.

Chen et al. [3] propose a clear categorization of the blockchain attacks. They focus on (1) mining pool attacks; which include attacks such as the 51%, pool hopping, selfish mining, and fork after withholding (FoW), (2) consensus excitation attacks; such as distributed denial-of-service (DDos), Sybil, eclipse, and reentrancy, (3) middle protocol attacks; mainly attacks targeting identity privacy and transaction information.

Attacks on blockchain have already caused substantial damage and proved that security and privacy are one of the most substantial issues in blockchain technology. There are several concrete examples of mining pool attacks where hackers launch

attacks on the mining pool to increase the expectation over their revenues. On May 16, 2018, Bitcoin Gold (BTG) adopted the EquiHash mining algorithm and supported graphic card mining, hackers took advantage and launched a 51% attack by renting sufficiently many graphic cards, the incident resulted in 12,239 stolen gold bits [4]. A similar 51% incident took place on January 5, 2019, where Ethereum Classic (ETC) had been attacked by hackers, again, renting the computing power of the graphic cards, resulting in a loss of $1.1 million US dollars [5].

Hackers utilize network communication attacks to interrupt the timely communication between the nodes and hurt certain types of cryptocurrencies. Back on September 22, 2016, Ethereum suffered greatly from a distributed denial of service (DDoS) which significantly reduced the transaction confirmation speed resulting in two hard forks of ETH [6]. A hard fork forces all nodes to upgrade to the latest version of a blockchain, which makes certain valid transactions invalid and validates certain pending transactions. Hard forks mark unstable time periods for cryptocurrencies and hackers do insider trading to generate revenues.

Hackers also launch attacks on the blockchain application layer where they find loopholes and steal coins staked in smart contracts. Smart contracts are essentially scripts stored, executed, and verified on the blockchain. A famous example of smart contract attacks is the *decentralized autonomous organization* (DAO) attack on June 17, 2016, when hackers detected loopholes in the smart contract and stole >3 million ETH, again forcing a hard fork [7]. Smart contract attacks are very hard to resolve in terms of legal sanctions, as smart contracts are immutable once they are deployed, and hacking a smart contract essentially means exploiting a bug that is completely the same as interacting with the smart contract in a usual way.

Other examples of practical attacks include attacks targeting servers of big trading platforms to steal the identity and transaction information of users. Such a privacy theft attack occurred on March 7, 2018, when hackers performed malicious transactions on the Binance exchange after stealing private user information, the attack significantly affected the price of the cryptocurrencies being traded on the platform [8].

The number of such attacks goes far beyond the examples provided above, and they constantly happen. This necessitates a comprehensive analysis where attacks are categorized, enumerated, and linked.

51% Attack

The 51% attack is arguably one of the most famous mining pool attacks as it is a direct violation of the core concept of decentralization. This attack is also known as the majority attack and occurs when a single miner or a group of miners controls over 50% of a network's computational power or total stakes.

Attackers achieve this by renting powerful computational resources such as graphic cards or dedicated mining devices from third-party providers.

Provided that an attacker controls more than 50% of the overall resources, he can block incoming transactions, rewrite parts of the existing chain, and change the order of his transactions resulting in double-spending. Ultimately, a successful 51% attack is extremely destructive to the integrity of the blockchain. However, the 51% attack has its limitations, the attacker cannot prevent others from broadcasting their transactions, reverse the order of others' transactions, or steal assets from unrelated parties. The likelihood of a 51% attack also decreases linearly with the network size as it gets more costly for the attacker to gather the needed computational resources, or stakes, as the network gets bigger.

Pool Hopping Attack

A pool hopping attack means abusing a mining pool. Mining pools implement different reward mechanisms, in some mining pools rewards are higher at certain times than at other times. A miner is expected to contribute to the pool equally through the good and bad times and their reward would be the statistical average of the collected block rewards over their contributions. Pool hoppers, however, hop into and out of the pool in a way that they only contribute during the good times and leave during the bad times, resulting in increased rewards at the expense of other miners.

For example, mining pools that implement the proportional method where a block reward is distributed between miners in proportion to the number of shares

they have submitted since the previous block are open to Pool Hopping attacks. In other words, in such pools, the reward per share equals the block reward divided by the number of shares in that round. Consequently, the reward of a share submitted is determined by the number of shares that have already been submitted since the last block, so a share submitted earlier in the round will yield a higher reward than a share submitted later. It can be shown that attackers can mine in a proportional pool until the number of shares reaches a certain threshold and hop out to join another pool at the threshold to have extra profits. Modern defenses to pool hoppers ensure that the reward per share depends only on the future states of the pool, not its past.

Block Withholding Attack

Block Withholding Attack falls under the category of mining pool attacks. There are two forms of block withholding, "Sabotage" and "Lie in Wait" as Rosenfeld names them [9].

In Sabotage withholding, the attacker in a mining pool withholds the blocks to harm the mining pool he had joined. This form of withholding brings no profit to the attacker but harms other pool participants.

In the second form of withholding, namely, Lie in Wait, the attacker keeps the block he mined and keeps mining the next block secretly. Then, the attacker releases more than one block making other miners waste their computational resources. By postponing submitting the blocks, the attacker earns more block rewards than he mines honestly. This is only possible provided that the attacker is in possession of sufficient computational power or sufficiently many stakes in the context of PoS-based networks. The Lie in Wait withholding is also known as Selfish Mining.

There are enhanced versions of the block withholding attacks such as the Fork After Withholding (FAW) attack combining Sabotage and Lie in Wait withholding where the attacker's profit is bounded below by the expected profit if he had mined selfishly [10].

DDoS Attack

DDoS Attack stands for "Distributed Denial-of-Service" attack, it is a notorious network attack as it assists many other attacks and acts as a subsidiary. The core idea behind the DDoS is to overwhelm the target node or the network infrastructure and disrupt normal network traffic.

A DDoS attack involves launching denial-of-service (DoS) attacks from many sources. The goal of the DoS attacks is to overwhelm individual nodes, this can be achieved by flooding the nodes with messages that require heavy processing. Launching from many sources makes the attack distributed, the attack source becomes harder to detect and the effects become harder to avoid [11].

DDoS attackers' aim is oftentimes to damage a competitor's business or to ask for a ransom to restore the network functionality. DDoS attacks are very common and most of the downtime incidents are associated with them.

DDoS attacks are slightly different in the blockchain context as decentralized networks eliminate single points of failure. DDoS attackers target the protocol layer by transaction flooding, spam transactions fill the limited-sized blocks and hinder legitimate transactions from being confirmed. This, if not detected early, results in network failure.

Common defenses to DDoS attacks targeting blockchain networks are to ensure that all nodes have adequate storage, processing power, and network bandwidth and then add fail-safes into the protocol code.

Eclipse Attack

An eclipse Attack is a network attack that aims to isolate a node and manipulate it into malicious acts. By isolating a node from its legitimate neighbors and surrounding it with artificial adversarial nodes, the eclipse attack can cause illegitimate transaction confirmations.

The success of the eclipse attacks depends heavily on the network topology and the underlying communication scheme as the attack is based on exploiting the target node's neighbors. The decentralized protocols of most blockchain networks

make it very hard for an attacker to create an artificial environment for a target node [12], however, eclipse attacks still rarely happen due to bandwidth constraints forcing a node to connect to a limited set of neighboring nodes instead of all of the network nodes.

Attackers often use botnets as attacker-controlled neighbor nodes and repeatedly force the target node to connect to the botnet by utilizing the DDoS attack. Once the target node has been compromised the attacker can misdirect the node to accept an invalid transaction or a transaction that has already been validated resulting in double-spending. The attacker can also make the victim node hide the fact that a block has been mined and effectively increase his relative hash rate to prepare for a 51% attack.

Defense mechanisms such as randomization, increasing node connections, and new node constraints make eclipse attacks even harder to happen in practice.

Sybil Attack

A Sybil attack makes a single node create and operate multiple fake identities, also known as the Sybil identities. The Sybil identities are used to undermine the authority in the network by gaining the majority of the influence [13].

A successful Sybil attack against a PoS-based network may create sufficiently many Sybil nodes which out-vote the honest nodes in the block confirmation process and avoid legitimate transactions.

Identity validation should be implemented o prevent Sybil attacks. The local network entities must query a central authority to perform reverse lookups on the remote entities. The look-up can use several identity verification techniques such as phone number, IP address, or credit card verification. These identity verification methods help reduce the risk of Sybil attacks but at the cost of sacrificing anonymity.

More sophisticated prevention mechanisms exist such as Social Trust Graphs where the network topology and connectivity are continuously analyzed to identify suspected Sybil clusters while maintaining the anonymity of the nodes [14].

Identity Privacy Attack

Anonymity is an important promise of blockchain technology. It is how transactions are kept confidential while being publicly available. Attackers try to obtain user privacy information to steal profits or impersonate users to fake transactions. Blockchain protocols rely heavily on cryptography primitives and the worst thing an attacker could do in a privacy theft is to get hold of a user's private key. Attackers steal private keys via spyware or social engineering methods, once they succeed they get complete control of the user's address. Private key thefts are commonly known as key attacks.

There are other ways in which the attacker can attempt to steal profits without being in possession of a user's private key. The attacker can perform a replay attack where he intercepts the transaction data and sends a fake packet received by the destination. The attacker can also perform an impersonation attack where he pretends to be a legitimate user.

The more information about the user's identity gets revealed, the stronger the impersonation attack becomes. Several studies propose frameworks that analyze transaction patterns to group addresses that likely belong to the same user in a cluster [15] [16], privacy is becoming a growing concern as such pattern recognition tools get better. Techniques such as zero-knowledge proofs proposed by S. Goldwasser et al. [17] significantly reduce the amount of information leakage, hence helping networks defend against de-anonymization.

2.1.3 Attack Graphs

Attack graphs are data structures used in cybersecurity to represent all possible paths of attack against a system. A path illustrates a sequence of states where the adversary ends up succeeding in a breach. One of the most common forms of attack graphs is the directed graph where nodes represent system states and edges represent exploits, i.e., parts of the attack that transform one state to another. As we walk through a path the transformed states are always more compromised than the preceding states, since edges are the exploits. Each path is a monotonically

increasing sequence of states in terms of the damage level. There are many variants of attack graphs such as probabilistic and Bayesian variants where uncertainty is incorporated into the formalization.

The use of attack graphs is substantial in building secure blockchain networks. Blockchain engineers must continuously and comprehensively test the system defenses to protect the network. The networks are inherently complex as they operate in multiple layers of abstraction. The complexity of these networks poses a real challenge when attempting to test them. Attack graphs help reduce this inherent complexity as they formally mark the vulnerabilities of a system and link them to those they enable. For a defender, understanding the attack graph topology is key to protecting critical states and avoiding risky paths.

Once an attack graph has been formed, defenders can analyze certain metrics of the graph to gain insight into the potential attacks, interpret the graph, and compare the modeled network to another that has also been formalized by an attack graph. An attack sequence is a generalization of a path, it is a sequence of exploits carried out in order by the adversary. Even simple metrics such as the shortest attack sequence and the number of attack sequences give useful information. The shortest attack sequence correlates to the easiness to attack the network as cybersecurity assumes that the network is as strong as its weakest link. The number of attack sequences correlates to the number of options available for an attacker as each attack sequence corresponds to a way of penetrating the network. There are as many more metrics as the number of graph properties and each has an insightful interpretation, there lies the representative power of attack graphs.

Attack Trees

An Attack Tree is a connected Attack Graph without any cycle. The edges of an Attack Tree are directed from the parent node to the child node. attack trees are simplified attack graphs, they are relatively easier to understand and suitable for scenarios where cyclic attack vectors are not common.

We use an Attack Tree with two types of nodes and two types of edges in our modeling: attack nodes, defense nodes, attack edges, and defense edges respectively. Attack nodes describe the state of the system as well as the action required to advance to the next state. Attack edges connect the attack nodes. Defense nodes describe countermeasures to the parent attack or further steps of the parent defense depending on the type of edge that connects the node to its parent. Defense edges connect the defense nodes and they connect attack nodes to their defense node children. We draw attack nodes and attack edges using straight lines, and defense nodes and defense edges using dashed lines.

3

Gasper: The Proof-of-Stake Consensus Protocol

Contents

3.1 Basics

We must narrow our discussion down to a particular PoS-based consensus protocol as security analysis is heavily dependent on the specifics of the protocols. Generalizing over protocols is possible to some extent but requires essential details to be abstracted.

We choose the *"Gasper"* protocol, as described in the original paper, an idealized version of the Ethereum 2.0 beacon chain [18]. The Ethereum 2.0 merge is arguably one of the most awaited events in the cryptocurrency world as Ethereum occupies a significant share of the market volume. Consequently, a lot of effort goes into the development of Ethereum's PoS-based protocol, and Gasper is presented directly by the members of the founding team. Hence, we find it sensible to focus our discussion on Gasper as we expect future blockchains to either adapt it or build on top of it.

Gasper combines a finality tool, *Casper FFG (the Friendly Finality Gadget)*, with a fork-choice rule, *LMD GHOST (Latest Message Driven Greediest Heaviest Observer Sub-Tree)*, while introducing subtle adaptations to both.

Casper FFG is not a standalone protocol in the sense that it serves as a gadget operating on top of a functioning blockchain. Casper FFG marks certain blocks as finalized so that nodes with partial information can still be fully confident that the marked blocks are part of the *canonical chain*, i.e., the chain that is consensually viewed as the main one.

LMD GHOST is a *fork-choice rule*, just like the longest chain rule, where validators *attest* to certain blocks, i.e., support certain blocks by means of voting.

Gasper is, again, an ideal abstraction of the Ethereum 2.0 beacon chain, so the planned Ethereum implementation differs from Gasper in some ways such as delays in attestations and finalization, or the validator sets being dynamic.

We try to keep up with the notation used in the original paper as we study the Gasper protocol, to keep the bookkeeping in this article easy and consistent.

Definition 3.1.1 (Validators) *V is the set of validators that are connected to each other and able to broadcast messages (typically blocks).*

Definition 3.1.2 (Genesis Block) *B_g is the "genesis block" representing the blank initial state during the cold start.*

Honest reasons such as network latency, as well as malicious reasons such as *Byzantine* validators, a common term in cybersecurity to mean dishonest participants, may result in conflicting blocks in the blockchain history. Thus, we cannot take the entire blockchain history as the common consensual history, we therefore define:

Definition 3.1.3 (History) *History is the choice of a "chain", essentially a linked list that is rooted at B_g and extends to a particular block.*

Definition 3.1.4 (Consensus History) *A history (a chain) that is mutually accepted as correct by all V.*

We assume that a message M sent by an honest validator V is sent to all validators of the network. Messages can be block proposals, attestation notices (supporting blocks), *slashing* (proving malicious acts of others), etc..

We also define:

Definition 3.1.5 (Digital Signatures) *A function $sign()$ exists for all messages M such that $sign(M) = V$ outputs the author of the message.*

The $sign()$ function is implemented using digital signatures in practice and it makes *impersonation* based attacks impotent.

3.1.1 Views & Network View

Validators do not necessarily observe the same blockchain at a given time. A validator may see some messages and miss others. A message is said to be *accepted* if all other messages that it *depends* on are accepted, recursively. We now define:

Definition 3.1.6 (View of a Validator) *$view(V, t)$ is the set of messages accepted by V until time t.*

Definition 3.1.7 (Network View) *NW denotes a hypothetical validator that has accepted all M broadcasted at any time. $view(NW, t)$ is called the network view.*

Property 3.1.1 (Network View encapsulates any View) *By definition, we have $view(V) \subseteq view(NW)$ for all V.*

We usually suppress time and drop the parameter t ($view(V, t) = view(V)$), unless time explicitly matters.

Some important assumptions and observations about views follow:

Property 3.1.2 (Cold Start) *All V start with B_g as the initial message. B_g has no dependencies and it is the only message for which the $sign()$ function is undefined.*

Definition 3.1.8 (Parent of a Block) *Each B that is not B_g has a parent block $P(B) = B'$, and contains a pointer to B' as part of its data. The parent-child relationship is shown using the edge $B \leftarrow B'$. Thus, each $view(V)$ defines a directed tree of parent-child edges, rooted at B_g.*

Definition 3.1.9 (Leaf Block) *Leaf blocks are the blocks having no children, L denotes the set of leaf blocks in a view.*

Definition 3.1.10 (Descendant Relation) *B' is a descendant of B in the chain $B_1 \leftarrow B_2 \leftarrow ...$, if there exists a path from B to B'.*

Definition 3.1.11 (Conflicting Blocks) *Two blocks conflict if neither is a descendant of the other.*

Definition 3.1.12 (Chain of a Block) *For a given block B, $chain(B)$ uniquely defines the path rooted at B_g and ending at B.*

3.1.2 Validators & Stakes

The above definitions and properties also apply for a PoW-based blockchain. We now proceed with PoS-specific definitions where a validator's voting power is proportional to their bonded stake in the network.

Definition 3.1.13 (Stakes) *$\Lambda = \{V_1, ..., V_N\}$ is the set of N validators. Each $V \in \Lambda$ has an amount $stake(V)$ representing the amount of collateral V has, $stake(V)$ is a positive real number. We assume $\sum_{V \in \Lambda} stake(V) = N$. As all operations involving $stake(V)$ are linear this scaling has no effect.*

3.1.3 Key Ingredients of a Consensus Protocol

The key ingredients we need to formalize to be able to define a consensus protocol are:

Definition 3.1.14 (Fork-choice Rule) *A $fork$ function identifies a single leaf block B when given a view W. $fork(W) = chain(B)$ and B is called the head of the chain in view W.*

Definition 3.1.15 (Finality Function) *A finality function F is a deterministic function where $F(W)$ returns the set of finalized blocks in the view W. If $B \in F(W)$, B is part of the consensus history.*

Definition 3.1.16 (Slashing Conditions) *Slashing conditions are rules the honest validators would never violate. If some V violates a slashing condition, $stake(V)$ is burned, or slashed. Slashing conditions are the key ingredients of a protocol's incentive mechanism.*

We say an *attestation* is a vote attached to a message indicating which block is seen as the head of the chain in a view.

3.1.4 Byzantine Validators

A *byzantine* validator is a dishonest validator. Practical Byzantine Fault Tolerance (PBFT) literature assumes that strictly less than $p = 1/3$ of the validators are *byzantine* [19], we stick to this constant.

Definition 3.1.17 (p-slashable) *A PoS-based blockchain is p-slashable if, at any time, there exists a validator V with $view(V) = view(NW)$ that can slash a byzantine validator or a group of byzantine validators with a total of pN stake.*

Being (1/3)-slashable is a very strong property as we cannot guarantee any sound properties as we have too many stakes held by byzantine actors.

3.1.5 Security Properties

The following properties are important in our construction:

Definition 3.1.18 (Safety) *A protocol is safe if $F(W)$ never contains two conflicting blocks for any view W. Consequently, any finalized view is a subchain of the finalized network view, i.e., $F(W)$ is a subsequence of $F(view(NW))$.*

Definition 3.1.19 (Liveness) *A protocol is live if the set of finalized blocks always grow. There are two variants of the liveness property:*

- ***plausible liveness**: regardless of any unexpected event, the logic of the protocol ensures that the set of finalized blocks grow.*

- ***probabilistic liveness**: regardless of any unexpected event, the set of finalized blocks grows in expectation provided we make probabilistic assumptions about the context of the protocol.*

3.1.6 Slots & Epochs

Definition 3.1.20 (Slot) *A slot is the atomic unit for time.*

Definition 3.1.21 (Epoch) *An epoch is the collection of a constant C slots.*

The time runs as:
$0^{(0)}, 1^{(0)}, ..., C-1^{(0)}, C^{(1)}, C+1^{(1)}, ..., 2C-1^{(1)}, 2C^{(2)}, 2C+1^{(2)}, ..., 3C-1^{(2)}, ...$

where the base denotes the slot number and the superscript denotes the epoch number. The epoch of slot i is simply $epoch(i) = floor(i/C)$. Epochs will help us introduce checkpoints later.

3.1.7 Synchrony

A consensus protocol should not assume that all validators have the same view of time. Therefore, the following synchrony conditions are defined:

Definition 3.1.22 (Synchronous Systems) *A synchronous system has explicit upper bounds for delays in sending and receiving messages.*

Definition 3.1.23 (Asynchronous Systems) *An asynchronous system has no synchrony guarantees.*

Definition 3.1.24 (Partially Synchronous Systems) *A partially synchronous system has explicit upper bounds for delays but they are unknown a priori and can be learned only after a certain unknown time t.*

Having defined the baseline, we now describe Casper FFG and LMD GHOST to build a foundation for Gasper.

3.1.8 Note on our Formalization

We want to clarify the ways in which our formalization differs from the original paper's in this subsection.

- Gasper is an idealized form of Ethereum 2.0's consensus mechanism, however the original paper does not hang back from giving practical details. We add a layer of abstraction and narrow our formalization to the "idealized" world where the assumptions we state along with our definitions always hold.
- The original paper assumes an informed reader, but we do not. We build our formalization from the ground bottom, by defining the primitives in simplest terms.
- We omit parameters and functions specifically defined in the original paper to discuss probabilistic liveness, dynamic validator sets, and extreme cases; again, because of our abstraction to the "idealized" world.

Ultimately, our formalization is a compact summary of key points of Gasper that is easy to follow, even for the uninformed reader.

3.2 Casper FFG

3.2.1 Checkpoints & Attestations

We first define:

Definition 3.2.1 (Height of a Block) *Height of B is the distance between B and B_g in terms of the number of blocks, $height(B) = |chain(B)| - 1$.*

Definition 3.2.2 (Checkpoint Block) *A checkpoint block B is a block with $height(B) = nH$, for some constant H and a nonnegative integer n.*

Definition 3.2.3 (Attestations & Checkpoint Edges) *An attestations is a signed message containing the checkpoint edge $B_1 \rightarrow B_2$ where both ends are checkpoint blocks. An attestation is a vote to move from B_1 to B_2, resulting in finalizing B_1 and justifying B_2. Ideally, $height(B_2) = height(B_1) + 1$, but practically a validator can miss certain blocks and attest to $B_1 \rightarrow B_3$ where $height(B_3) - height(B_1) > H$.*

Definition 3.2.4 (Attestation Weight) *An attestation has a weight which can be thought of as its voting power. The attestation weight is the stake of its author, $weight(\alpha) = stake(sign(\alpha))$.*

3.2.2 Justification & Finalization

Given a view W, $J(W)$ denotes the set of *justified* checkpoint blocks, and $F(W)$ denotes the set of *finalized* checkpoint blocks.

We define the concept of a **supermajortity link**, a key part of Casper.

Definition 3.2.5 (Supermajority Link) *A supermajority link, denoted by* $B' \xrightarrow{J} B$*, exists if the combined weight of attestations to* B *observed in a given view* W *is at least two thirds of the total stake,* $\sum_{\alpha_i:B' \to B} weight(\alpha_i) \geq 2N/3$.

Casper FFG finalizes blocks in a view using the following rules:

- **Justification Rule:** A checkpoint block B is justified in view W, if there exists a supermajority link to B in W.
- **Finalization Rule:** A checkpoint block B is finalized in view W, if there exists a supermajority link from B to some other checkpoint block B' in W ($B \xrightarrow{J} B'$) (B is justified ($B \in J(W)$), and B' is the consecutive checkpoint ($height(B') = height(B) + 1$)).

Property 3.2.1 (Justification is a Pre-condition of Finalization) *The finalization rule states that a block can only be finalized in* W*, if it has been justified in* W*. This implies* $F(W) \subset J(W)$*, note that this is a proper subset relation as the set* $J(W)$ *is always strictly larger than* $F(W)$ *(we omit the cold start).*

3.2.3 Slashing Conditions

Casper FFG introduces certain slashing conditions, which are conditions honest validators never violate. When a validator V' proves that another validator V violated a slashing condition, V' slashes V by burning $stake(V)$ and getting some portion of it as the slashing reward. The following are Casper's slashing conditions:

- **SC1: Fork Attempt:** No validator V can make two attestations $\alpha_1 : A_1 \to B_1$ and $\alpha_2 : A_2 \to B_2$ with $height(B_1) = height(B_2)$. In words, V cannot attempt a fork at a checkpoint, he should explicitly vote for a chain.

- **SC2: Override Attempt:** No validator V can make two attestations $\alpha_1 : A_1 \rightarrow B_1$ and $\alpha_2 : A_2 \rightarrow B_2$ with $height(A_1) < height(A_2) < height(B_2) < height(B_1)$.

3.2.4 Guarantees

Casper FFG provides the following guarantees:

- **Accountable Safety:** Two checkpoint blocks on different branches cannot both be finalized. If this happens, it means that a set of validators provably violated the slashing conditions hence they can be slashed.

- **Plausible Liveness:** Casper FFG always makes progress in finalizing new checkpoints, no deadlock occurs.

Recall that Casper FFG is a finality tool, not a complete protocol, so its functioning and its guarantees depend on the underlying blockchain creating new blocks without any problems.

3.3 LMD GHOST

The *Greediest Heavies Observed Sub-Tree* (GHOST) is a fork-choice rule that selects the branches with the most activity [20]. Zamfir [21] adapts GHOST to the PoS setup and calls the variant *Latest Message Driven Greediest Heavies Observed Sub-Tree* (LMD GHOST).

We re-parametrize the $weight()$ function as follows. Given the view W, block B, and a set of latest attestations M in W; $weight(W, B, M)$ is the sum of validator stakes whose last attestations in M are to B or B's descendants.

Data: View W
Result: A block B uniquely defining a chain
$B \leftarrow B_g$;
$M \leftarrow$ *set of latest attestations in* W;
while B *has a descendant in* W **do**
  $B \leftarrow argmax_{(B' \text{ child of } B)} weight(W, B', M)$;
  (compare block hashes in case of ties)
end
return B

Algorithm 1: LMD GHOST Fork Choice Rule

LMD GHOST uses the weights of the subtrees at the forks as the heuristic and assumes the "heaviest" is the correct one. It is a simple greedy algorithm in the sense that it always selects the most supported branch. LMD GHOST is complete, it always returns a leaf block B uniquely defining the canonical chain $chain(B)$.

3.4 Gasper

Having defined the necessary bits and pieces, we now define the main protocol Gasper. We describe parts of the protocol below where variations to existing definitions and additional definitions are given when needed.

3.4.1 Epoch Boundaries

We define the following:

Definition 3.4.1 (Epoch Boundary Pair) *An epoch boundary pair is a variation of Casper's checkpoint block to disambiguate the checkpoints occurring on the same chain multiple times. We use an ordered pair* (B, j) *where* B *is the checkpoint block and* j *is the epoch. The resulting tuple is called the (epoch boundary) pair.*

Definition 3.4.2 (Epoch Boundary Block) $ebb(B, j)$ *denotes the epoch boundary block of B, it is the block with the largest slot number less than or equal to* jC *on* $chain(B)$*, i.e.,* $argmax_{(B:slot(B) \leq jC)} slot(B)$.

Definition 3.4.3 (Last Epoch Boundary Block) $lebb(B)$ *denotes the last epoch boundary block of B, it is the epoch boundary block of B with the last seen epoch number, i.e.,* $argmax_{(B:slot(B) \leq j_{max}C)} slot(B)$.

Some observations are listed below:

Property 3.4.1 (Genesis Epoch) *$ebb(B, 0) = B_g$ for all B.*

Property 3.4.2 (Exact Boundaries) *If $slot(B) = jC$, then $ebb(B', j) = B$ for all B' such that $B \in chain(B')$ (B' is a descendant of B)*

Property 3.4.3 *B can be an epoch boundary block in some chains but not in all.*

Attestations are now to pairs, not blocks. An attestation α to the pair $P = (B, j)$ is said to have *attestation epoch* j, $ep(\alpha) = aep(P) = j$, which can be different than $ep(B)$.

3.4.2 Committees

Gasper introduces the concept of *committees* to distribute responsibilities among the participants. Validators are partitioned into committees in each slot, and one committee member is assigned as the block proposer. Other members of the committee, say V, then attest to the head of $fork(view(V))$, which will hopefully be the last block proposed by the block proposer, i.e., $fork(view(V)) = fork(view(NW))$.

We assume we have access to randomized length-N permutation $p_j : \{1, ..., N\} \to \{1, ..., N\}$ for epoch j. We use p_j during epoch j to split the set of validators Λ into C equal-size (N/C) committees $S_0, ..., S_{C-1}$. More clearly, for a $k \in \{0, ..., C-1\}$ we define $S_k = \Lambda_{p_j[s]}$ where $s \equiv k \pmod{C}$, here p_j shuffles Λ and s indexes (selects from) p_j.

Definition 3.4.4 (Committee) *A committee is a pseudo-randomly selected, N/C-size, set of validators responsible for appending a block to the blockchain at a given slot.*

Definition 3.4.5 ((Block) Proposer) *A pseudo-randomly selected member of the committee that is responsible for proposing a new block at a given slot.*

3.4.3 Attestations

Each slot is a round of the protocol, and in each round, the proposer proposes a new block and others on the committee attests to their head of the chain. The proposal and attestations are both digitally-signed and broadcasted messages.

The block proposer is selected at the beginning of the slot $i = jC + k$ as $V_k = \Lambda_{p_j[k]} = S_k[0]$, in words, the first member of the committee S_k. When V_k proposes a new block B, he broadcasts a message M_k containing the following information:

- $slot(B) = i = jC + k$
- $fork(view(V_k, i)) \leftarrow B$, in words, the canonical head in V_k's view at slot i is the parent of B. We also say $P(B) = fork(view(V_k, i))$.
- $newattests(B)$: a set of attestation pointers V_k has accepted, that supports B as the head of the chain
- Data specific to the blockchain application

The proposed block B, therefore, depends on $P(B)$ (and $newattests(B)$) and cannot be processed until $P(B)$ is in the processing node's view.

Digital signatures ensure the block proposed at slot i is proposed by the proposer designated for slot i, mathematically, if $slot(B) = jC + k$, then $sign(B) = V_k$. As a result, dishonest behavior such as proposing a multiple blocks in a single slot number can be slashed.

Now each of the other members $V \in S_k \setminus \{V_k\}$ broadcasts an attestation α at the middle of the slot i $(i + 1/2)$ containing the following information:

- $slot(\alpha) = i = jC + k$
- $block(\alpha) = B'$, which is the block α votes for. If $B' = fork(view(V_k, i))$ (the block V_k has just proposed at the beginning of slot i), then $slot(\alpha) = slot(B')$. Otherwise V is attesting to an earlier block and $slot(\alpha) > slot(B')$.

- A *checkpoint edge*: $LJ(\alpha) \xrightarrow{V} LE(\alpha)$. $LJ(\alpha)$ and $LE(\alpha)$ are epoch boundary pairs in $view(V, i + 1/2)$, we define them in the next section.

The attestation α, therefore, depends on $block(\alpha)$ and cannot be processed until $block(\alpha)$ is in the processing node's view.

Gasper attestations can be consideres as both "GHOST votes" for the head block and "Casper FFG votes" for checkpoint blocks.

3.4.4 Justification

Recall that $lebb(B)$ is the last epoch boundary block, the block with the greatest slot number such that $slot(B) \leq jC$ where j is the largest epoch so far.

We define the following for a given block B:

Definition 3.4.6 (View of a Block) *The view of a block B, $view(B)$ (note that this is different than $view(V)$, the argument to the function is now a block not a validator), is a collection of B and all its ancestors derived from the dependency graph. If $B \in view(V)$ for some V and $B \in view(V')$ for some V', then $view(B)$ is identical for V and V' as $view(B)$ is essentially $chain(B)$ with additional dependencies (attestations, etc.).*

Definition 3.4.7 (FFG View of a Block) *The FFG view of a block B, $ffgview(B) = view(lebb(B))$ is the snapshot of $view(B)$ at the latest checkpoint. $ffgview(B)$ extracts the information in $view(B)$ that is relevant to Casper FFG.*

We extend the Casper FFG definition and say there exists a *supermajority link* between the epoch boundary pairs (B', j') and (B, j); $(B', j') \xrightarrow{J} (B, j)$, if the attestations with the checkpoint edge $(B', j') \xrightarrow{V} (B, j)$ have a combined weight of 2/3 of the total stake.

Gasper justifies blocks in a view using the following rules:

- **Genesis Rule:** $(B_g, 0) \in J(W)$

- **Justification Rule:** A pair (B, j) is justified in view W, if there exists a supermajority link to (B, j) in W.

$(B, j) \in J(W)$ reads as "B is justified in W during epoch j". j is the attestation epoch of B.

We, finally, define $LJ(\alpha)$ and $LE(\alpha)$ for a given attestation α:

Definition 3.4.8 (Last Justified Pair (LJ)) *$LJ(\alpha)$ is the last justified pair of α, the pair in $fggview(block(\alpha))$ with the highest attestation epoch.*

Definition 3.4.9 (Last Epoch Boundary Pair (LE)) *$LE(\alpha)$ is the last epoch boundary pair of α, that is $(lebb(block(\alpha)), ep(slot(\alpha)))$*

3.4.5 Finalization

Note that finalization is stronger than justification. If a block B of a pair is finalized in some view W at some slot j, no other block conflicting with B (a block that is in another branch) can be finalized in any view, unless the blockchain is (1/3)-slashable (the strongest guarantee we have).

Gasper finalizes a pair (B_0, j) in a view using the following rules:

- **Genesis Rule:** If $(B_0, j) = (B_g, 0)$, then $(B_0, j) \in F(W)$
- **Finalization Rule:** A pair (B_0, j) is **k-finalized** in view W, if for some $k \geq 1$ and a sequence of blocks $B_1, ..., B_k \in view(W)$ we have:
  1. $(B_0, j), (B_1, j+1), ..., (B_k, j+k)$ are consecutive epoch boundary pairs in $chain(B_k)$
  2. $(B_0, j), (B_1, j+1), ..., (B_{k-1}, j+k-1)$ are all $\in view(W)$
  3. $(B_0, j) \xrightarrow{J} (B_k, j+k)$

Most of the time a block is just finalized (1-finalized), i.e., $(B_0, j) \in J(W)$ and the set of latest attestations in W has a supermajority link $(B_0, j) \xrightarrow{J} (B_1, j+1)$. $k > 1$ cases occur rarely due to latency and delay issues, such cases are included in the definition for completeness.

3.4.6 Hybrid LMD GHOST

We begin with a prototype algorithm that simply extends the LMG GHOST algorithm to epoch boundary pairs:

Data: View W
Result: A block B uniquely defining a chain
$(B_j, j) \leftarrow$ the justified pair with the highest attestation epoch in W;
$B \leftarrow B_j$;
$M \leftarrow$ *set of latest attestations in* W;
while *B has a descendant in W* **do**
  $B \leftarrow argmax_{(B' child\ of\ B)} weight(W, B', M)$;
  (compare block hashes in case of ties)
end
return B

Algorithm 2: Prototype HLMD GHOST Fork Choice Rule

The prototype implementation suffers from the following problems:

- The finalization part, i.e., the checkpoint edge contained in an attestation includes $LJ(\alpha)$ which is the last justified pair in the "frozen snapshot" $ffgview(block(\alpha))$. However, (B_j, j) is not frozen and may change during an epoch, which may lead honest validators to violate certain slashing conditions.

- In the case of forks, the resulting blocks can have radically different last justified pairs due to the branches growing at substantially different speeds. As a result, an honest validator that had attested to a higher last justification epoch but forked to a chain whose last justification epoch is older may find himself violating a slashing condition.

Below is the actual HLMD GHOST algorithm. The main idea is to reduce the view W to the view W' so that the problems described above are prevented. The initial lines of the algorithm are pre-processing steps to compute W'. We first consider the leaves of W, as they are the most recent messages observed by the validator. Then, we backtrack from these leaves and find the highest justification epoch checkpoint pair. So instead of using heights as in Casper FFG, we compare the frozen snapshots of each chain defined by the leaves and trust the most recently

supported (attested) one, this helps us in choosing a leaf node while agreeing on checkpoints. We then eliminate the leaves that conflict with the computed (B_j, j), and reduce W to only include the chains defined by leaves that are descendants of (B_j, j). W' allows us to execute LMD GHOST while using the FFG justification and finalization information in a correct way.

Data: View W
Result: A block B uniquely defining a chain
$L \leftarrow$ set of leaf blocks B_l in W;
$(B_j, j) \leftarrow$
the justified pair with the highest attestation epoch in $J(ffgview(B_l))$ over $B_l \in L$;
$L' \leftarrow$ set of leaf blocks B_l in W such that $(B_j, j) \in J(ffgview(B_l))$;
$W' \leftarrow$ union of $chain(B_l)$ over $B_l \in L'$;
$B \leftarrow B_j$;
$M \leftarrow$ *set of latest attestations in* W;
while *B has a descendant in W'* **do**
  $B \leftarrow argmax_{(B' \text{child of } B)} weight(W, B', M)$;
  (compare block hashes in case of ties)
end
return B

Algorithm 3: HLMD GHOST Fork Choice Rule

3.4.7 Slashing Conditions

Gasper adapts Casper FFG slashing conditions as:

- **SC1: Fork Attempt:** No validator V can make two attestations α_1 and α_2 with $ep(\alpha_1) = ep(\alpha_2)$. In words, V cannot attempt a fork at an epoch, he should explicitly vote for a chain.

- **SC2: Override Attempt:** No validator V can make two attestations α_1 and α_2 with $aep(LJ(\alpha_1)) < aep(LJ(\alpha_2)) < aep(LE(\alpha_2)) < aep(LE(\alpha_1)$.

3.4.8 Guarantees

The ultimate guarantees of Gasper are "Safety", "Plausible Liveness", and "Probabilistic Liveness". We will provide proof of Safety as we believe it builds a stronger insight into the mechanics of the protocol.

Sub-guarantee 3.4.1 (Uniqueness of Attestation Epochs) *All justified pairs in a view, all $P \in J(W)$, have unique attestation epochs, otherwise, the blockchain is $(1/3) - slashable$.*

Proof. *Suppose for a contradiction that we have 2 distinct pairs with the same attestation epoch, (B, j) and (B', j) in $J(W)$. Then, $\sum_{(\alpha_i : A \rightarrow B \wedge \ aep(\alpha_i)=j)} weight(\alpha_i) \geq 2N/3$, and $\sum_{(\alpha_i : A \rightarrow B' \wedge \ aep(\alpha_i)=j)} weight(\alpha_i) \geq 2N/3$. As the total stakes in a round equal N, the sets $\Lambda_1 = \{(\alpha_i : A \rightarrow B \wedge \ aep(\alpha_i) = j)\}$ and $\Lambda_2 = \{(\alpha_i : A \rightarrow B' \wedge \ aep(\alpha_i) = j)\}$ must intersect. The intersection means we have duplicate attestations which violate **SC1** and can be provably slashed.*

Sub-guarantee 3.4.2 (Honest Validators) *Honest validators never accidentally violate the slashing conditions.*

Proof. *An honest validator is asked to attest exactly once per epoch as it is assigned to a specific committee, so it cannot violate **SC1**.*

*Suppose for a contradiction that the honest validator V is going to violate **SC2** in epoch t_4. V should have written an attestation $\alpha_i : (B_2, t_2) \xrightarrow{V} (B_3, t_3)$ and is now about to write $\alpha_j : (B_1, t_1) \xrightarrow{V} (B_4, t_4)$ such that $t_1 < t_2 < t_3 < t_4$.*

Now if we run HLMD GHOST on V's view at t_4, we get a leaf block B that is a descendent of B_4, i.e., $LE(B) = (B_4, t_4)$.

At t_3, V wrote α_i, so $(B_2, t_2) \in J(ffgview(B_l))$ for some leaf block B_l at t_3. As pairs that are already in $J(W_V)$ do not change as V's chain grows, we know that at t_4 (B_j, j), the justified pair with the highest attestation epoch in $J(ffgview(B_l))$ as defined in HLMD GHOST, must have $j \geq t_3$, and $t_3 > t_1$, so $j > t_1$.

*At t_4, however, the output block B must satisfy $(B_j, j) \in J(ffgview(B))$ (L' in HLMD GHOST). We know $LJ(B) \in J(ffgview(B))$ and $LJ(B) = B_1$ as in α_j. This implies $j \leq t_1$ which is a contradiction. Hence, following Gasper cannot force a violation of **SC2**.*

Note that rewards and penalties should be adjusted numerically such that the game theory of the protocol ensures that validators are incentivized to behave honestly, follow the slashing conditions, and catch misbehaving participants.

Sub-guarantee 3.4.3 (Justified Blocks have Finalized Parents) *Given a view W, if $(B_j, j) \in J(W)$ and $(B_f, f) \in F(W)$ where $j > f$, then B_f is an ancestor of B_j. Otherwise, the blockchain is (1/3)-slashable.*

Proof. *Suppose for a contradiction that there is a pair (B_j, j) with $j > f$ and B_f is not an ancestor of B_j. Because B_f is finalized, we have $(B_f, f) \xrightarrow{J} (B_f, f+k)$ where $(B_f, f), (B_1, f+1), ..., (B_k, f+k)$ are adjacent epoch boundary pairs.*

Knowing that B_j is not a descendant of B_f, we can say that B_j is not included in the sequence up to $(B_k, f+k)$. So there exists another pair $(B_l, l) \xrightarrow{J} (B_j, j)$, where $l < f$ and $j > f$, we omit the equality cases as there is a guarantee for the uniqueness of the attestation epochs. Addedly, $j > f + k$ as B_j is not part of the sequence.

So there must be a subset Λ_1 of Λ in view W, such that the combined stakes of Λ_1 members that have attested to $(B_l, l) \rightarrow (B_j, j)$ are more than $2N/3$. For any such attestation α_j, we have $aep(LJ(\alpha_j)) = l$ and $ep(\alpha_j) = j$.

Similarly, there must be a subset Λ_2 of Λ in view W, such that the combined stakes of Λ_2 members that have attested to $(B_f, f) \rightarrow (B_k, f+k)$ are more than $2N/3$. For any such attestation α_f, we have $aep(LJ(\alpha_f)) = f$ and $ep(\alpha_f) = f + k$.

*Now, an honest validator $V \in \Lambda_1 \cap \Lambda_2$ will see two distinct attestations α_j and an earlier α_f, where $l < f < f + k < j$, but then $aep(LJ(\alpha_j)) < aep(LJ(\alpha_f)) < ep(\alpha_f) < ep(\alpha_j)$. Thus V sees a violation of **SC2** and slashes the attester.*

Now comes the main guarantees of Gasper:

Guarantee 3.4.1 (Safety) *A view W is $(1/3) - slashable$ or we are certain of the properties below:*

1. *Any pair included in $F(G)$ remains in the set as W is updated through time.*
2. *If $(B, j) \in F(G)$, then B is in the canonical chain of W. Consequently, $chain(B)$ is a subsequence of the canonical chain of W.*

Proof. *The first property follows from the definitions of justification and finalization.*

HLMD GHOST always selects the block with a canonical chain that goes through the justified pair with the highest attestation epoch, with the last sub-guarantee this means that it always goes through the highest finalized pair in $F(W)$. If we show that no finalized blocks can conflict, then it follows that all finalized blocks must be in the same chain, which is a subchain of the consensus history.

Suppose for a contradiction that (B_1, f_1) and (B_2, f_2) are pairs in $F(W)$ and they conflict. Then W must be (1/3)-slashable because the last sub-guarantee tells us that a justified pair is necessarily the descendant of a finalized pair and we know that a pair gets justified before it can be finalized. Thus, either B_1 is the descendent of B_2 or vice versa, the only way they can conflict is, again, via a (1/3)-slashable blockchain.

Guarantee 3.4.2 (Plausible Liveness) *The blockchain is $(1/3) - slashable$ or we are certain that the honest validators will follow the protocol and new blocks will be finalized.*

The proof for plausible liveness is based on assuming plausible conditions such as good synchrony and honest validators.

The Probabilistic Liveness guarantee and the corresponding proof delve into probability theory and game theory. We omit that proof in this article and advise the curious reader to see it in the original paper as our purpose is to analyze consensus.

This in-depth overview of Gasper hopefully provides a good foundation to form an attack tree.

4
Attack Tree

Contents

4.1 Preliminary

Bellchambers provides an attack tree for generic blockchain applications where he makes the elementary assumption that the fork choice rule compares the chains by length alone [22]. His tree presents a clear top-level categorization of attacks as well as comprehensive top-down attack vectors. We, therefore, acknowledge his

work as our origin. We prune his tree when an attack vector is not applicable to the Gasper consensus protocol. We also branch his tree to highlight new paths of attacks that are not applicable for the PoW and longest chain settings. Note that our tree is a Gasper-specific tree, but many elements of it apply to general PoS protocols as Gasper is an instance of them. We discuss which paths of attack apply to general PoS protocols in the "Reflection" section.

Our approach aligns with the future work proposed by Bellchambers as an extension to his dissertation. He specifically states that the longest chain assumption restricts the potential blockchain applications and reasonably modern cryptocurrencies such as Ethereum have different notions of fork choice. He notes that it would be interesting to relax certain assumptions and consider different applications. His work inspired our extension and we thereby appreciate his efforts.

4.2 Top-Level Categorization

We stick to Bellchambers' top-level categorization with slight adaptations. Bellchambers' categorization and attack tree are included in the Appendix.

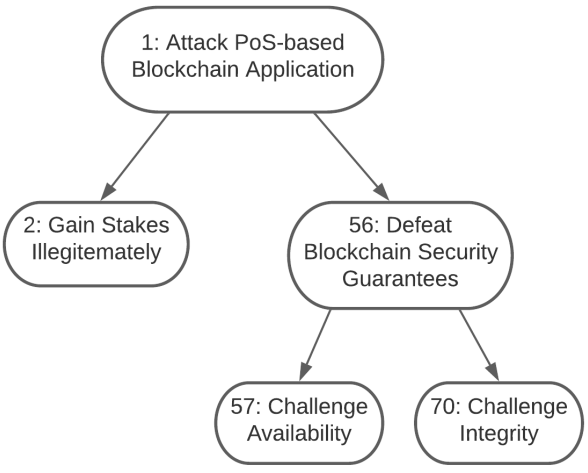


Figure 4.1: Top-Level Categorization of PoS Attacks

Semantically, we categorize three ways of attacking a PoS-based network. The latter two are categorized under the same node as they are challenging the security guarantees of a blockchain, in the general sense.

1. **Gain Stakes Illegitimately**: A secure network depends on the fairness of the flow of stakes, a validator must not be favored over others, or must not steal others' stakes.

2. **Challenge Availability**: Recall that the previous chapter stated the plausible and probabilistic liveness properties of the Gasper protocol, which mean that the blockchain keeps growing, i.e., new pairs are being finalized, no matter what. A desirable network must be able to process transactions at any time.

3. **Challenge Integrity:** Consensus is all about integrity, a blockchain must maintain the accuracy and consistency of the stored data.

4.3 Gain Stakes Illegitemately

An attacker can gain stakes illegitimately in two ways: it can either steal others' stakes or increase his relative stakes by downplaying others.

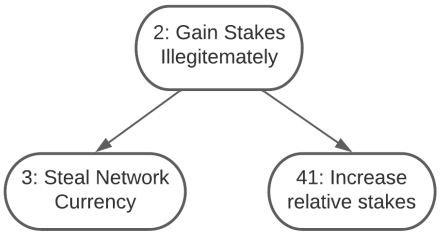


Figure 4.2: Ways of Gaining Stakes Illegitemately

4.3.1 Steal Network Currency

Stealing currency is essentially stealing a private key. The private key of a user is the key to his safe, or his wallet. An attacker can attempt digitally stealing the wallet and its key using adversarial software, or it can try physically stealing it and hacking the hardware. Common countermeasures against such thievery are specified in the tree. For example, Node 18 states a good practice to defend against stolen wallets, it suggests the user use his cold wallet only to withdraw money to his hot wallet from which he temporarily holds money that is just enough for the

awaiting transactions. Node 21 suggests memorizing the wallet seed. An attacker's job would be very hard in the hypothetical case where the user memorizes the seed and only signs from his cold wallet to send money to his hot wallet address, i.e., both countermeasures are taken. The attacker may also try to trick or coerce the user into false payments. He can also attempt to defeat certain cryptographic primitives to deduce the private key or interfere with the key exchange algorithms.

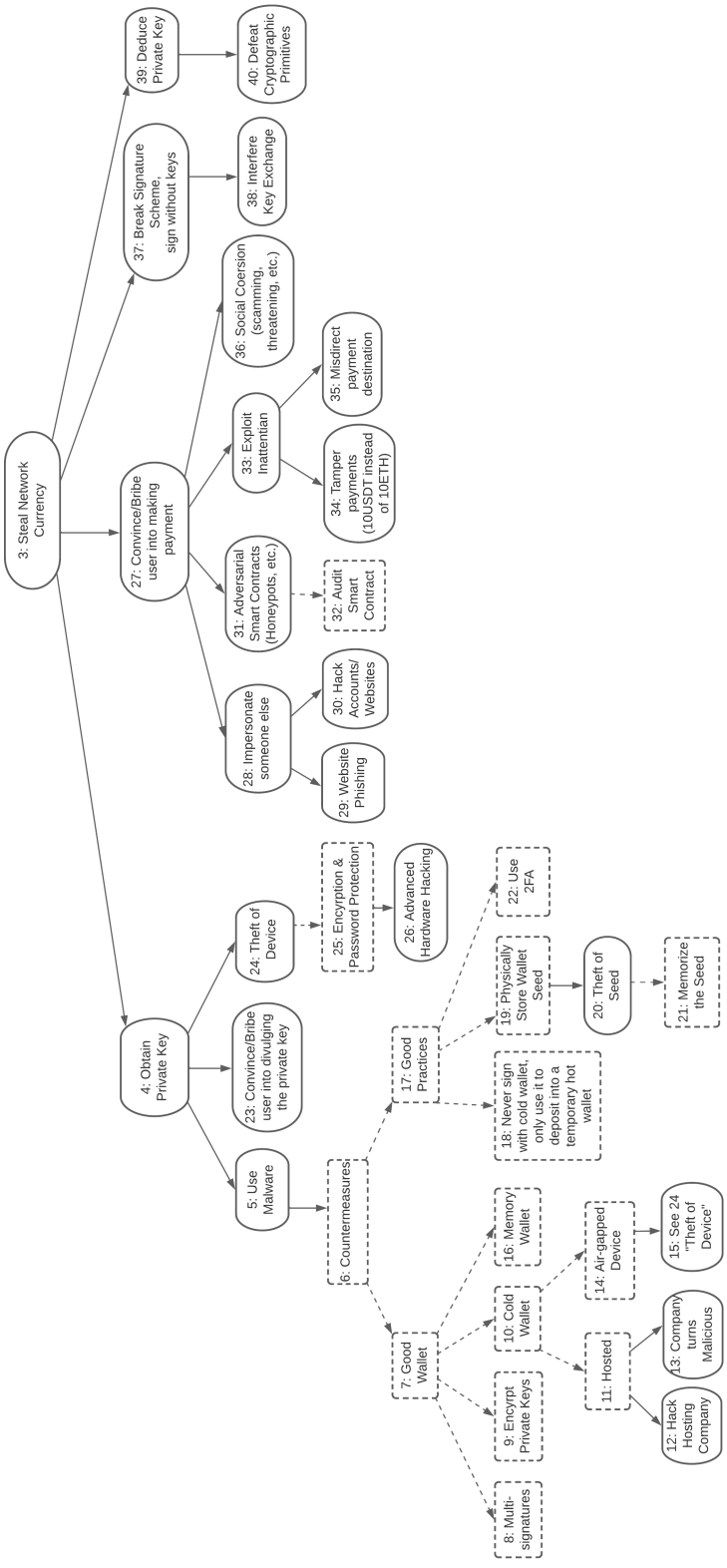


Figure 4.3: Steal Network Currency

4.3.2 Increase Relative Stakes

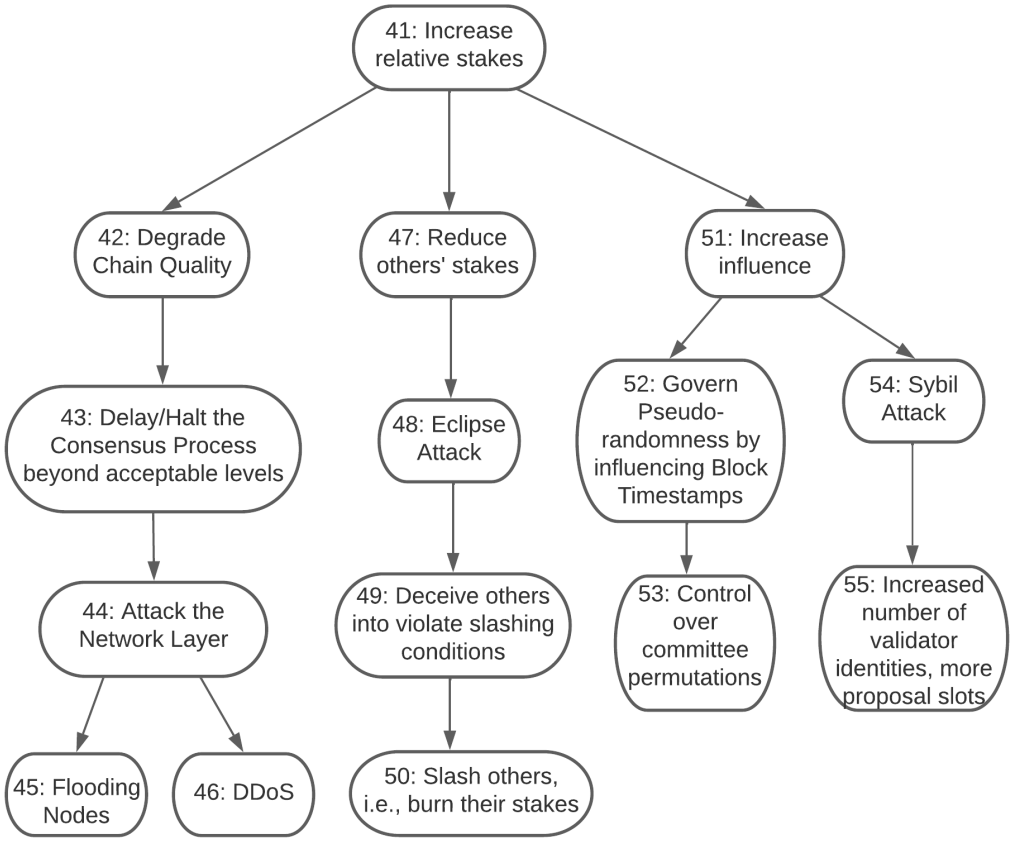


Figure 4.4: Increase Relative Stakes

From the attacker's perspective, increasing his stakes relative to others corresponds to having more control on the consensus process. One way of doing this is to introduce unacceptable delays to the network, which in turn disrupts the synchrony of others, if the attacker maintains good synchrony then he would secure control over the consensus. Another sophisticated way of increasing relative stakes is given by the attack vector defined by the rightmost leaf. An attacker may create Sybil identities and have more influence on the algorithm. Instead of validating with C stakes as a single validator, he can validate with C/k stakes as k validators. This way, he will have k times higher probability to propose blocks as he will be occupying k positions in the committee permutations. In other words, the attacker will be using the same amount of stake more effectively and thus have more control over the consensus algorithm. Other ways of increasing relative stakes are provided in the tree above.

4.4 Challenge Availability

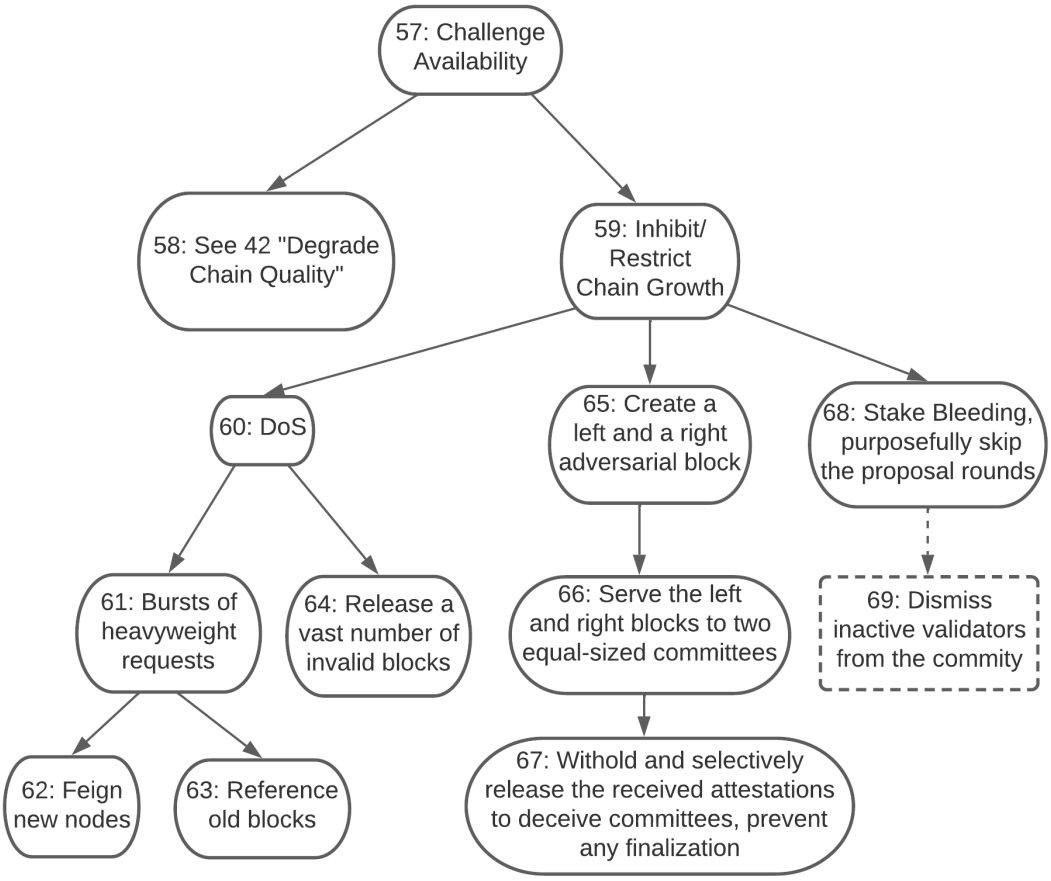


Figure 4.5: Challenge Availability

Challenging availability comes in two forms. An attacker can either sabotage a node or a set of nodes, or he can exploit certain vulnerabilities of the underlying protocol to avoid block finalization. The attacker can degrade the chain's quality to dis-function certain sets of nodes, ways of doing this are enumerated in the sub-tree rooted at Node 42. If the attacker chooses to target the protocol layer, then his options are DoS, *the Balancing Attack*, and *Stake Bleeding* (these attacks are described in the following sections).

4.5 Challenge Integrity

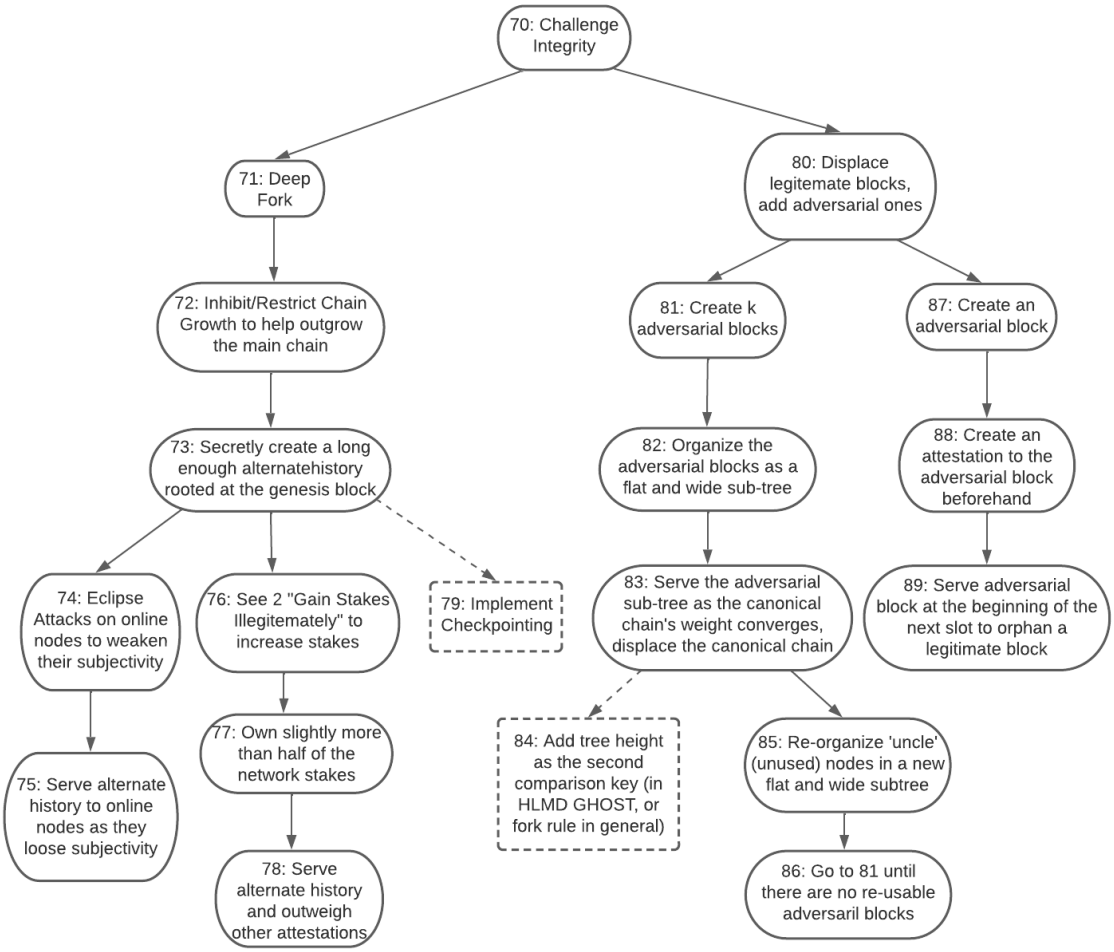


Figure 4.6: Challenge Availability

Ways of challenging the integrity of the chain are vectorized above. The path defined by Node 75 is the *Long Range Attack*, Node 78 defines the *51% Attack*, these attacks are ways of deep forking. The rightmost two paths are ways of replacing honest blocks with adversarial ones, these attacks are called the *Avalanche Attack*, and the *Reorg Attack* respectively. All the named attacks are described in the next section.

4.6 Recognizable Attacks in the Tree

4.6.1 The Balancing Attack

(Node 65)

The Balancing Attack is similar to the Blockwitholding Attack we described in the introduction where we delineated what sort of things an attacker can do. The attacks are similar in the sense that, in both, the adversary purposefully withholds messages and releases them intentionally to harm the network participants. One can think of the Balancing Attack as the Gasper counterpart of the Blockwithholding Attack. Neu et. al. formalize the attack in [23], we give a brief description.

We make the following assumptions about the network:

1. The attacker knows, up to a certain error margin, when the honest validators execute the HLMD GHOST algorithm, i.e., run *fork()* to attest to the head of their view.

2. The attacker is able to target a message for delivery to an honest validator before a certain point in time.

3. Honest validators cannot exchange information about their local views arbitrarily quickly, i.e., they must wait a certain time to update each other about the messages they have received.

These assumptions are sensible, they oftentimes hold. The first assumption is given by the definition of Gasper as the Casper FFG votes are attestations sent by the committee at the beginning of new epochs. The second and third assumptions hold when the attacker has better synchrony than the other network participants, which is not an ignorable scenario.

We define:

Definition 4.6.1 (Opportune Epoch) *An epoch is opportune if the attacker is assigned as the proposer of the first slot and there are sufficiently many (six is enough [23]) adversarial validators in each slot of the epoch. The probability of an epoch being opportune is roughly f/n for large n where f is the number of adversarial validators. In expectation, an attacker controlling* 1% *of the validators waits for only* 100 *epochs for an opportune epoch.*

The steps of the attack are described below:

1. The attacker waits for an opportune epoch.
2. The attacker produces two conflicting blocks: B_{left} and B_{right}
3. The attacker reveals B_{left} and B_{right} to two equal-sized subsets of the committee: C_{left} and C_{right} respectively. Naturally, C_{left} attests to B_{left} and C_{right} attests to B_{right}. The attacker withholds the attestations he received.
4. The attacker selectively releases the withheld attestations from the previous slot to manipulate the validators of the next slot into two equal-sized groups where one sees B_{left} and the other sees B_{right} as parts of their canonical chains.
5. The attacker continues this strategy in the upcoming slots and epochs. He releases withheld attestations selectively to reaffirm honest validators in the illusion that their previous votes are in accordance with what is still happening.

The attacker can continue this indefinitely to break the liveness of the protocol and avoid it from finalizing new blocks as the attestations from "left" and "right" groups are, in expectation, of equal weight. Recall that finality requires a combined weight of $2N/3$ supporting a specific pair.

A careful reader may point out slashing and say that the attacker would get slashed when it broadcasts two conflicting blocks. This is true, the proposer in the first slot of the opportune block would lose his stakes, however, B_{left} and B_{right} are not slashed and remain part of the history in the consensus protocol. Hence, slashing is not a proper defense against the Balancing Attack.

4.6.2 The Reorg Attack

(Node 87)

A reorg attack is a deliberate attempt to rewrite the consensus history as an alternate legitimate chain of transactions. Such an attack violates the integrity of the blockchain, which is arguably the most important promise of the technology. Schwarz-Schilling et. al. describes a strategy for [24] a low-cost reorg attack against the Gasper protocol:

1. The attacker secretly creates B_{n+1} as the child of B_n and attests to it, at the beginning of slot $n+1$. Others attest to B_n as their view do not include B_{n+1}.

2. An honest proposer publishes B_{n+2}, at the beginning of slot $n+2$. Now, assuming good synchrony, the attacker publishes the secret B_{n+1} and its attestation. The committee now sees B_{n+1} and B_{n+2} simultaneously, these blocks conflict as they share the same parent B_n.

3. B_{n+1} and B_{n+2} both inherit the attestation weight of B_n, due to the heaviest sub-tree strategy. However, the committee attests to the adversarial block B_{n+1} as a result of HLMD GHOST as it has one more attestation which the attacker had secretly prepared at slot $n+1$.

4. At the beginning of slot $n+3$, an honest proposer will create B_{n+3} as the child of the adversarial block B_{n+1} as it outweighed the legitimate block B_{n+2}. This effectively displaces the honest block out of the chain, i.e., reorgs out the block, or orphans it.

Schwarz-Schilling et. al. call the above strategy the 1-reorg attack, and generalize the strategy to k-reorg where k is used to parameterize the portion of honest validators the attacker must control [24].

One may point out slashing, but again, slashing makes the attacker lose its stakes, not the adversarial blocks it had created.

4.6.3 The Avalanche Attack

(Node 81)

The Avalanche Attack aims to displace honest blocks out of the consensus history just like the attack we have previously described. The attacker exploits a vulnerability inherent in the design of the GHOST algorithm. The attacker releases the withheld blocks in a flat but wide sub-tree when the tree's weight catches up to the weight of the legitimate chain. At first, it may seem like the attacker has to create an abounding number of adversarial blocks. However, this is not the

case as the attacker has the advantage of re-using "uncle" blocks, this is because only two blocks in the flat-wide sub-tree replacing the long-legitimate one joins the canonical chain when a fork choice is made. The rest of the adversarial blocks act as attesters which can be used over and over again.

The Avalanche Attack may not be crystal clear when described in words, so we put a pictorial illustration of what happens in the attack below:

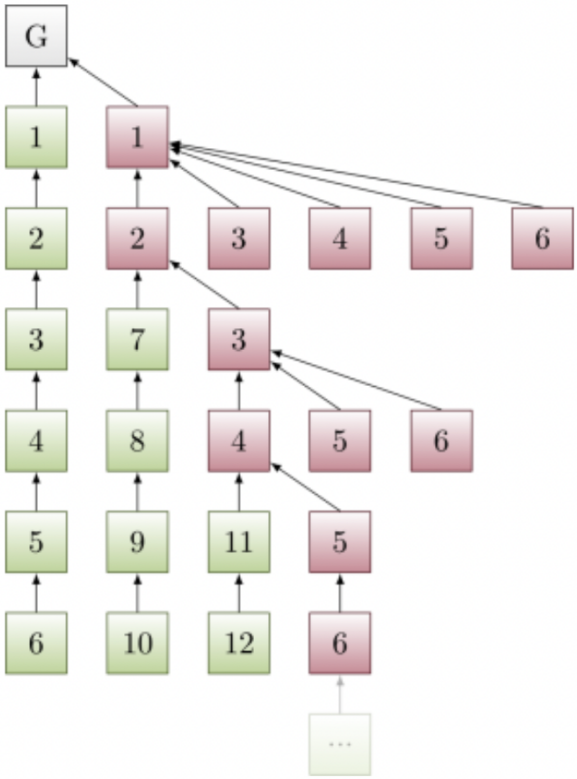


Figure 4.7: An Illustration of the Avalanche Attack

The green boxes represent the honest blocks whereas the red ones represent the adversarial ones. The flat sub-tree in the second and third levels of the tree is released by the adversary when the $6th$ honest block is added to the canonical chain, this in turn displaced the honest chain. The attacker repeats this strategy and releases another flat sub-tree, the one in the third and fourth levels, when the $10th$ honest block is added, and so on. Observe that the attacker reuses certain blocks. Analysis shows that with k adversarial withheld blocks, an attacker can displace $O(k^2)$ honest blocks [25].

Steps of the attack are listed below:

1. The attacker secretly creates k adversarial blocks.
2. The attacker waits until the weight of the honest canonical chain matches the flat and wide sub-tree it created. The attacker releases the adversarial sub-tree outweighing the honest one when the time arrives. The adversarial sub-tree displaces the legitimate one.
3. The attacker waits until the new sub-tree formed by the honest proposals matches the weight of the flat and wide sub-tree it created by re-using the "uncle" adversarial blocks. It releases the adversarial sub-tree when the time arrives and displaces the honest blocks.
4. The attacker continues in this fashion, displacing as many honest blocks as possible until it runs out of reusable blocks.

Neu et. al. show that the LMD variant of the GHOST rule interferes with the Avalanche Attack but comes with its own problems [26].

Defenses

One may add node depth as a comparison key in the HLMD GHOST algorithm. In other words, the algorithm will not only look at the sub-tree weight when choosing a branch but also look at how deep that branch goes, perhaps it will look at some weighted linear combination of the two parameters. Note that this will not prevent the Avalanche type of attacks completely but will force the attacker to organize the adversarial sub-trees in a narrower form, which decreases the number of blocks he can re-use.

4.6.4 The Long Range Attack

(Node 73)

The Long Range Attack is often attributed as the PoS version of the 51% Attack in the sense that the attacker takes over the blockchain with a false one

of its own. The attacker rewrites the history by replacing the consensus history with the so-called "alternate" history.

Weak Subjectivity is a critical property of PoS blockchains. It describes the situation where offline nodes or nodes that have been offline for a sufficiently long time are not certain of the canonical chain. They need to traverse the entire history, starting from the genesis block, and end up in a specific leaf defining the consensus history. For that, they run the $fork()$ function.

Costless Simulation is another critical property in the PoS setting. It means that anyone can create a branch, or a sub-tree, rooted at the genesis block, with a negligible cost. This is simply because the PoS consensus does not involve any computational puzzle solving.

Weak Subjectivity together with Costless Simulation implies that an attacker can create a custom-tailored branch and deceive non-active nodes. Even the active, i.e., online, nodes can be deceived via Eclipse type of attacks.

Before describing the defenses against the Long Range Attack, we point out an inherent challenge for the attacker. The attacker must outperform the honest validators in creating a real-time branch as blocks contain timestamps. If the attacker can forge timestamps, then it can simply create blocks ahead of time as it is the single validator viewing the adversarial branch. However, creating blocks becomes a competition when timestamp creation is secured via cryptographic primitives. The attacker has two ways of outperforming others in the block creation process and these ways are described below:

Posterior Corruption

Under certain conditions, validators can leave the network, i.e., they can retire. This must be the case as it would be unfair if the set of validators, Λ, had been static.

One way for the attacker to create additional blocks is to hack (or bribe and get) a retired validators, V_r private key. Once the attacker controls the retired validators private key, it can use the blocks V_r had created in the past that are not part of the consensus history, sign them as V_r and add them into the adversarial

chain. The more retired validators the attacker controls, the higher its chances to outgrow the canonical chain.

Stake Bleeding

(Node 68)

Another strategy the attacker can follow is to stall the canonical chain by intentionally decreasing its stakes. Every round the attacker gets selected as the block proposer it can purposefully skip the round, which in turn stalls the canonical chain's growth in the cost of not having any block rewards. This gradually decreases the attacker's stakes, hence the name *stake bleeding*. The attacker grows the alternate chain in parallel and publishes it when it outgrows the main chain. Here we use outgrow to mean that it is selected over the main chain when a $fork()$ call is made.

Defenses

The most effective defense against the Long Range Attack is the use of checkpoints. Checkpointing allows only a subset of the latest blocks to be reorganized and makes it impossible to construct an entire alternate history. Note that an attacker could still launch the attack if it succeeds to forge the checkpoints.

The Gasper protocol, which is the one we are focusing on, is secure against this attack as Gasper votes contain Casper FFG votes that finalizes pairs as checkpoints.

4.6.5 The 51% Attack

(Node 77)

The notorious 51% attack also comes into play in PoS-based networks. However, the attacker must control 51% of the total network stakes instead of the total compute power.

4.7 Reflection

The tree we have established is Gasper-specific, meaning all attack vectors are created with Gasper's vulnerabilities in mind. However, Gasper is an instance of a

PoS consensus protocol, so most branches of our tree are either directly applicable to PoS protocols in general or have counterparts for other PoS protocols.

The sub-tree rooted at Node 3, "Steal Network Currency", is applicable to all PoS protocols. Since stealing tokens depends on getting hold of a user's private key or deceiving the user, this sub-tree does not have a Gasper-specific exploit. In fact, this sub-tree is applicable to all paradigms of consensus, including PoW and others.

The sub-tree rooted at Node 41, "Increase Relative Stakes" is also quite general. The sub-tree defined by Node 42, "Degrade Chain Quality", is applicable to all consensus paradigms, as the attack strategy involves attacking the network layer which is common to all blockchain types. The sub-tree given by Node 47, "Reduce Others' Stakes", is applicable to all PoS protocols implementing slashing, with the only difference being the way in which you deceive the user in Node 49 as the specific slashing conditions may differ. Similarly, the sub-tree given by Node 51, "Increase Influence", is applicable to any PoS network that conducts the block proposal process by voting rounds among randomized committees.

Nodes 58 (Referencing Degrade Chain Quality), 60 ("DoS"), and 68 ("Stake Bleeding") listed as ways of challenging availability are also generally applicable to all PoS protocols. However, Node 65, in the availability sub-tree defines the Balancing Attack, which is specific to Gasper.

In the integrity sub-tree, Node 71, "Deep Fork", explains a general strategy to replace the canonical PoS chain, but Node 79, "Checkpointing" gives a defense option that is implemented in Gasper. The branches rooted at Node 80, "Displacing blocks", are Gasper specific as they describe the Avalanche and Reorg attacks respectively.

4.8 Comparison with the PoW Tree

Janse van Rensburg's doctoral thesis on analyzing Attack Graphs inspired the current section of our article [27]. The "Relative Attack-Graph Security" chapter of his work demonstrates that it is possible to objectively compare two different attack graphs in terms of security to a certain extent. However, Rensburg's proposed method for comparison depends on comparing paths that share common exploits,

otherwise, the attack graphs cannot be compared as the cost of different exploits are necessarily subjective for different attackers. He, therefore, mentions other simpler ways to draw lighter conclusions on the secureness of the attack graphs, one of which is to compare the number of paths.

A complete path, or attack vector, in our attack tree, is a path starting from the root node and extending to a leaf node. Note that the defenses are given in dashed lines and they are not parts of the main tree, they are considered as parts of the augmented tree, so we do not consider the defense leaves as leaf nodes. Intuitively, if a tree has many leaf nodes, it means that the system it is modeling has many vulnerabilities that give the attacker chances to accomplish different attacks. Based on this intuition, Rensburg proposes the *number of paths* metric to compare the likelihood of an attacker reaching his targets in different systems modeled by different trees.

The number of paths metric also has some important limitations. It weighs each path equally, irrespective of the length of the paths and the exploits contained in them. This is a problem of the metric as well as a problem of attempting to "objectively" compare systems in terms of their secureness.

The PoS tree has 24 leaf nodes, whereas Bellchamber's PoW tree has 43 leaf nodes. A direct conclusion of this difference in the number of paths is that PoW systems supply the attackers with more ways of attacking the system. This is an intuitively correct conclusion as PoS-based consensus is derived from PoW-based consensus with the vulnerabilities and limitations of PoW in mind. It is important to note that the number of paths in the two trees is associated with the level of detail their authors have decided to provide. The reader is encouraged to enhance both trees by adding new paths that they may have noticed and re-compare them, even so, we think that the level of detail we have provided matches Bellchamber's.

5
Conclusion & Future Work

This article focused on PoS networks and specifically the Gasper protocol which is a variant of stake-based consensus algorithms.

The foundation of consensus is to prove that you are willing to sacrifice "something" of importance in exchange for information. This "something" can be computational work, staked currency, storage space, or real-life concepts. Hence there are many types of networks: Proof-of-Work, Proof-of-Stake, Proof-of-Capacity, and Proof-of-Concept respectively. We call this "something" the consensual value. Exploring interesting consensual values and the associated attacks other than work and stake would be a good direction for future research.

In the case of PoS-based networks, it would be interesting to explore how the choice of the consensus algorithm opens up possibilities for new types of attacks, or how the algorithm prevents certain attacks. For example, the Avalanche Attack and its variants are specifically targeting the GHOST fork-choice rule, other fork-choice rules may be resilient to such attacks but vulnerable to others. Hence, delving into the specifics of the consensus algorithms would be a great step forward. Even parameters of Gasper (such as slots in an epoch, slashing conditions, etc.) would drastically affect the success of attacks, the effect of algorithm parameters on specific attacks would also be a great research topic.

Our attack tree serves as a model that explores the possible attacks in general terms. However, more advanced models may incorporate edge weights to specify the costs of certain exploitation or to quantify the probability of certain paths. An interesting research goal would be to try and construct a probabilistic attack tree, or an attack graph that captures cyclic exploitation.

The field of blockchain is still crawling, and the pace of innovation is great. Analyzing attacks at a matching pace is crucial to building reliable systems. In fact, it must be a prerequisite in blockchain development, as we are using blockchain networks in mission-critical fields such as finance, defense, and health.

Appendices

.1 Appendix: PoW Tree

The following figures are parts of Bellchamber's attack tree for the PoW-based networks with longest chain fork-choice.

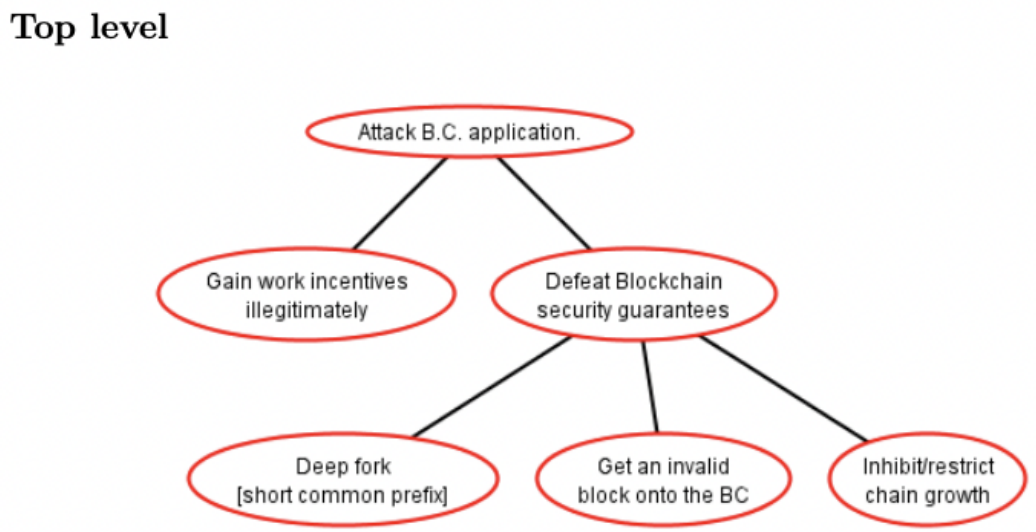


Figure 1: Top-level Categorization

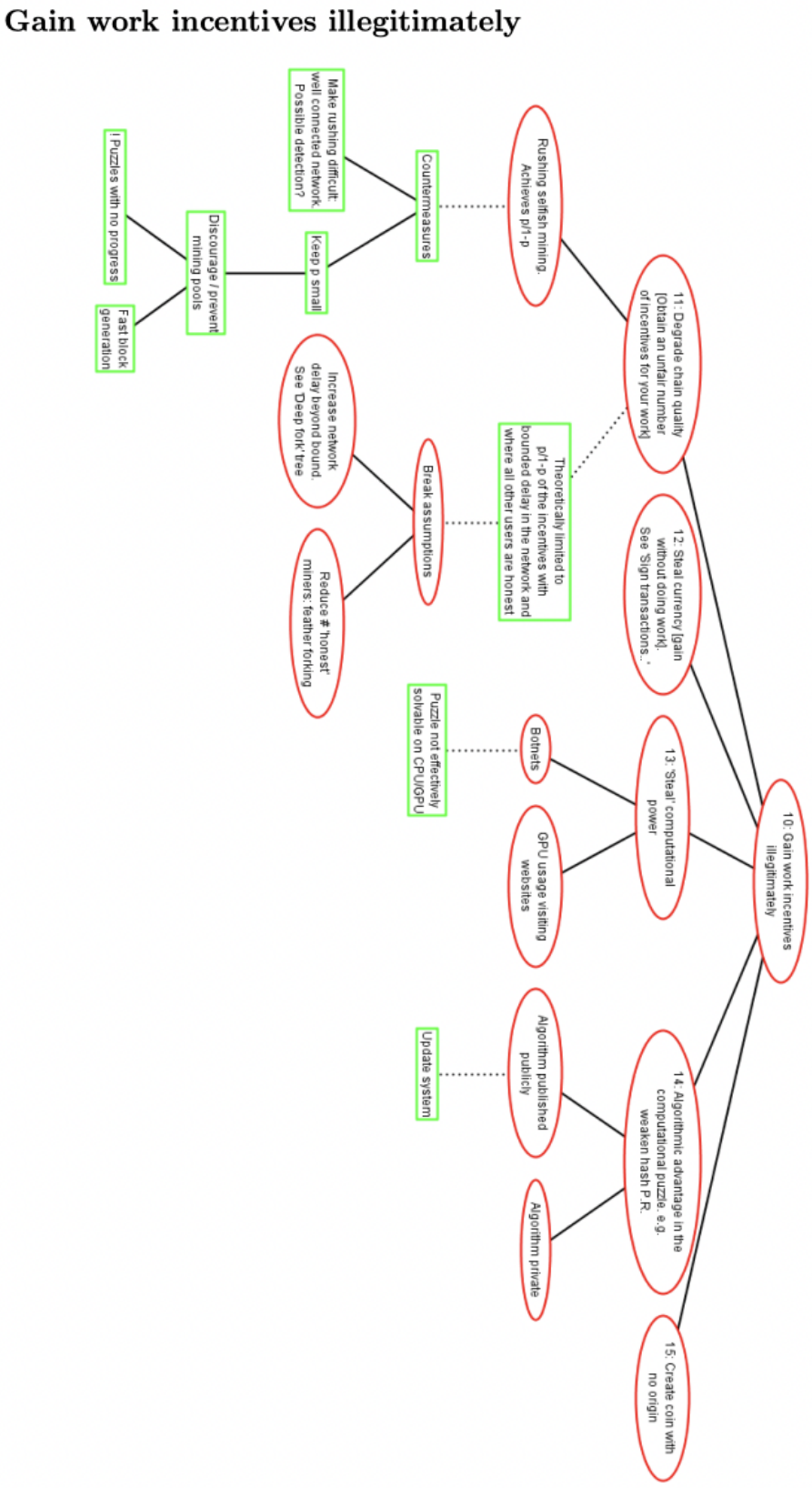


Figure 2: Gaining Work Incentives Illegitimately

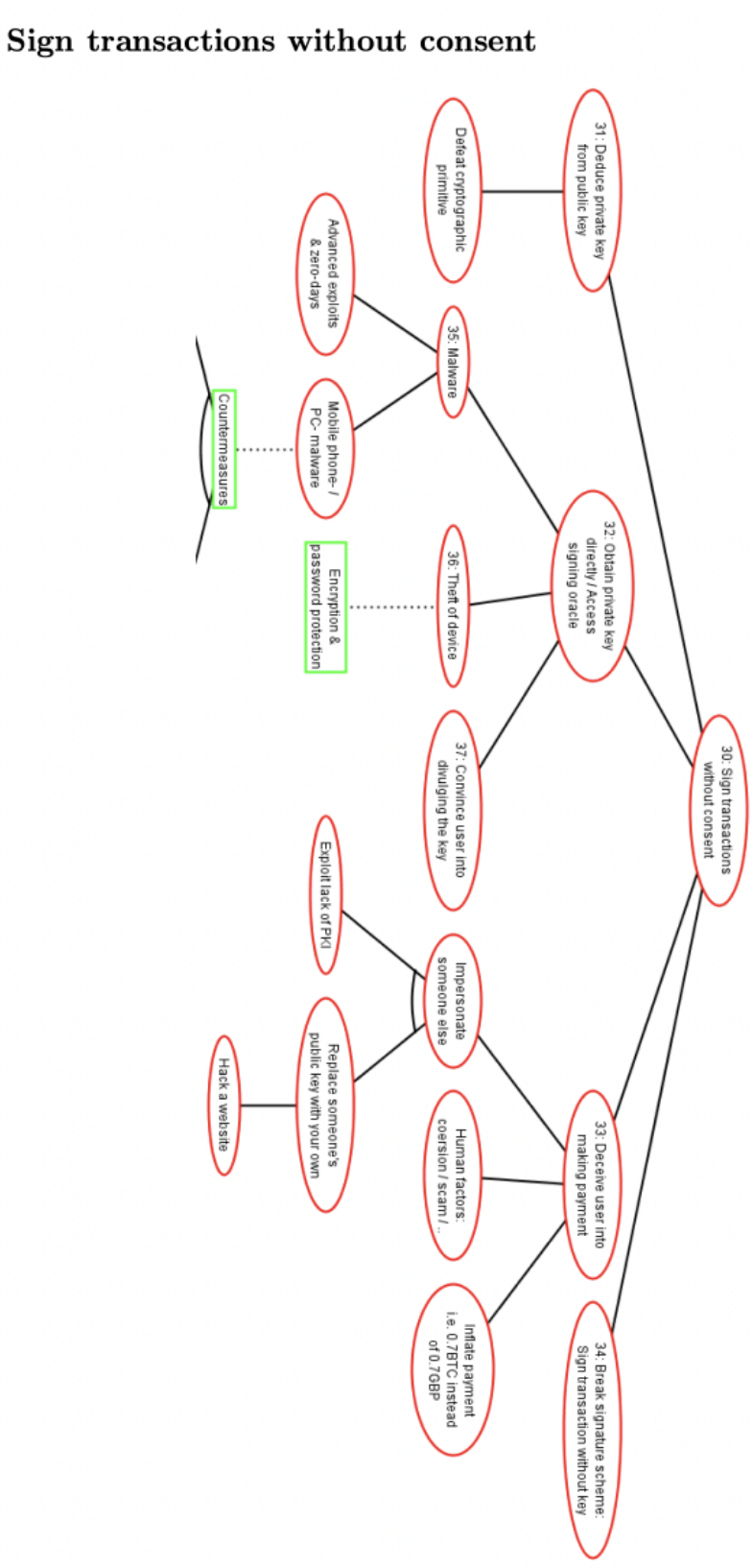


Figure 3: Signing Transactions Without Consent

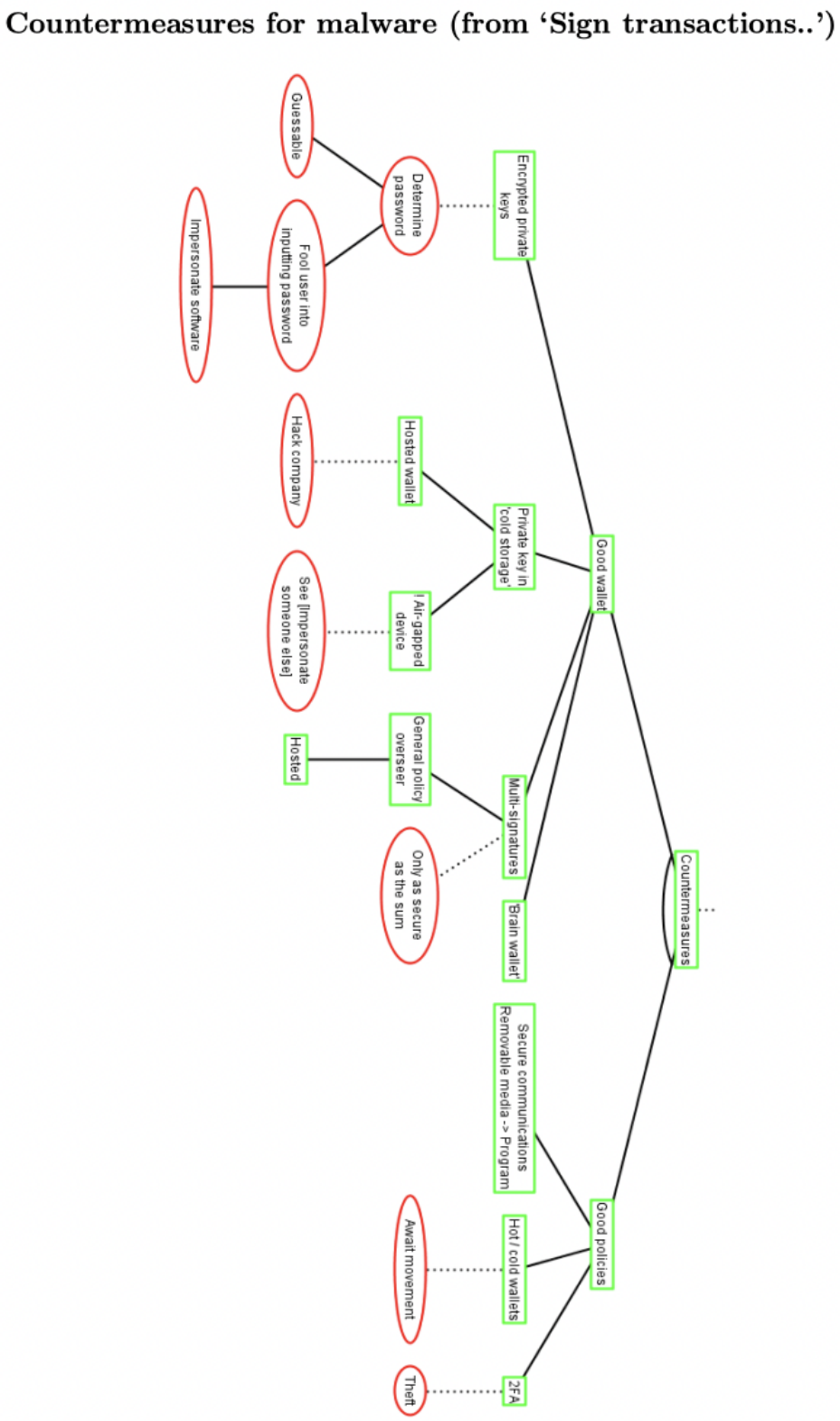


Figure 4: Countermeasures for Malware

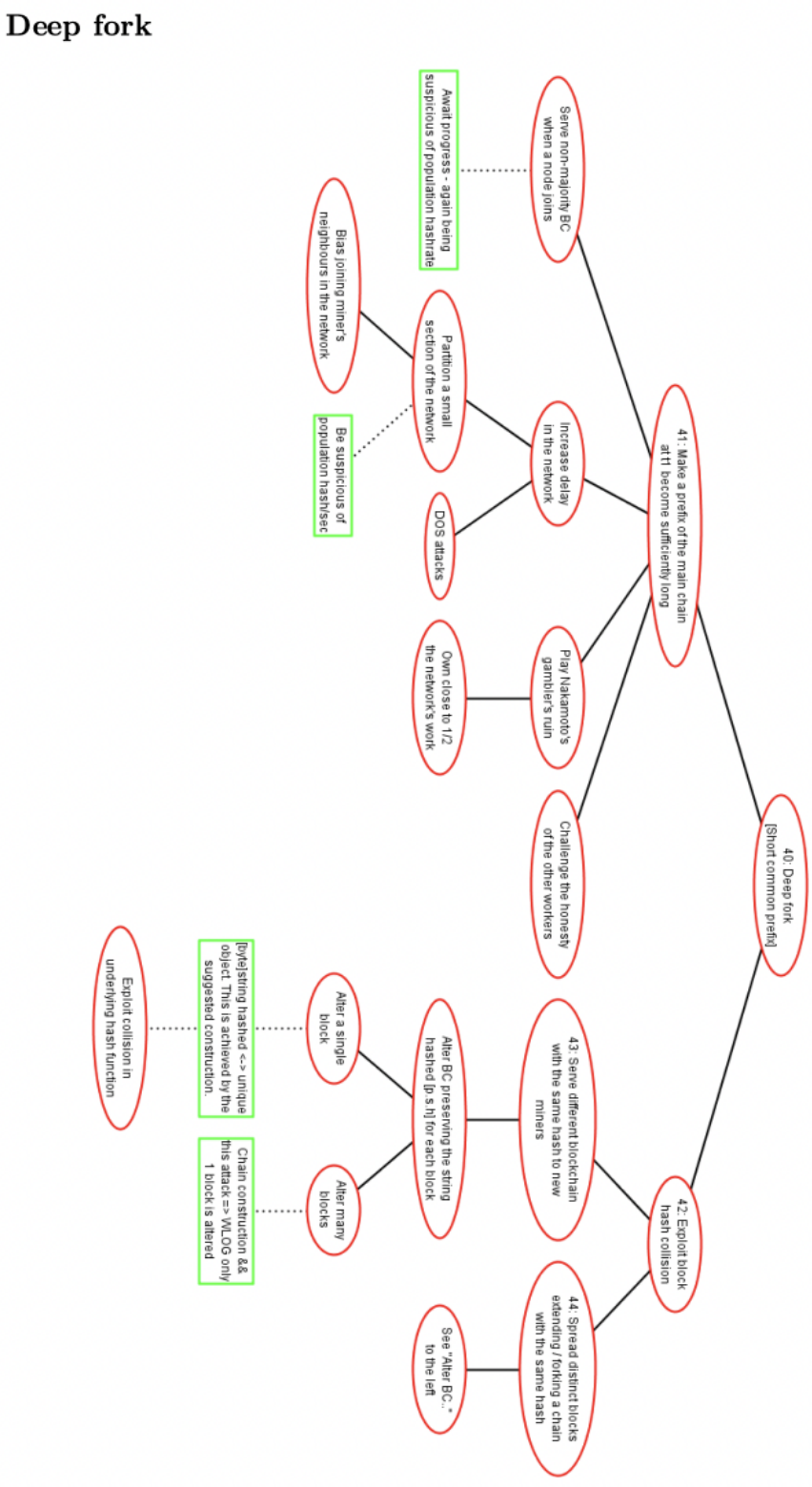


Figure 5: Deep Fork

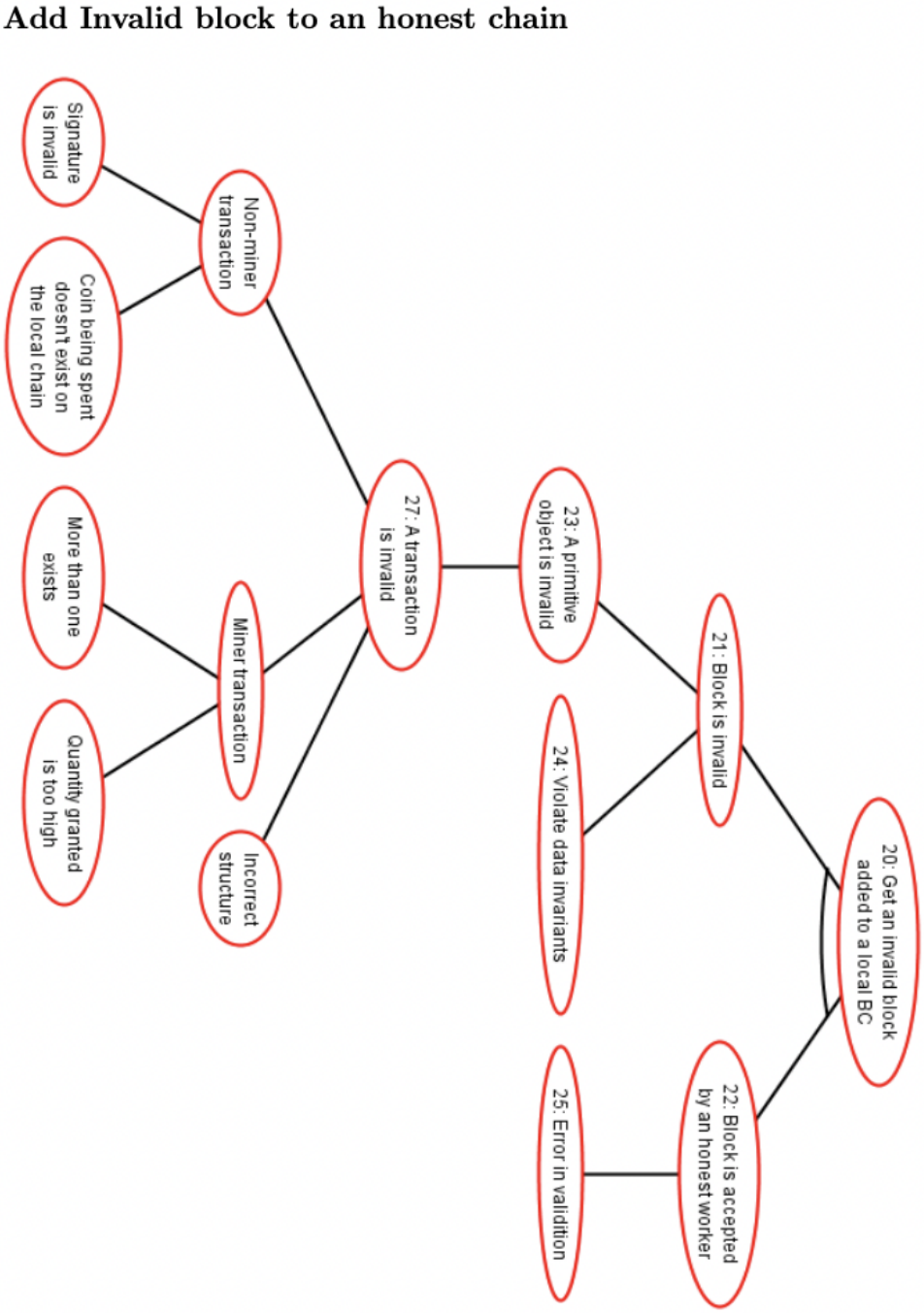


Figure 6: Adding an Invalid Block to the Chain

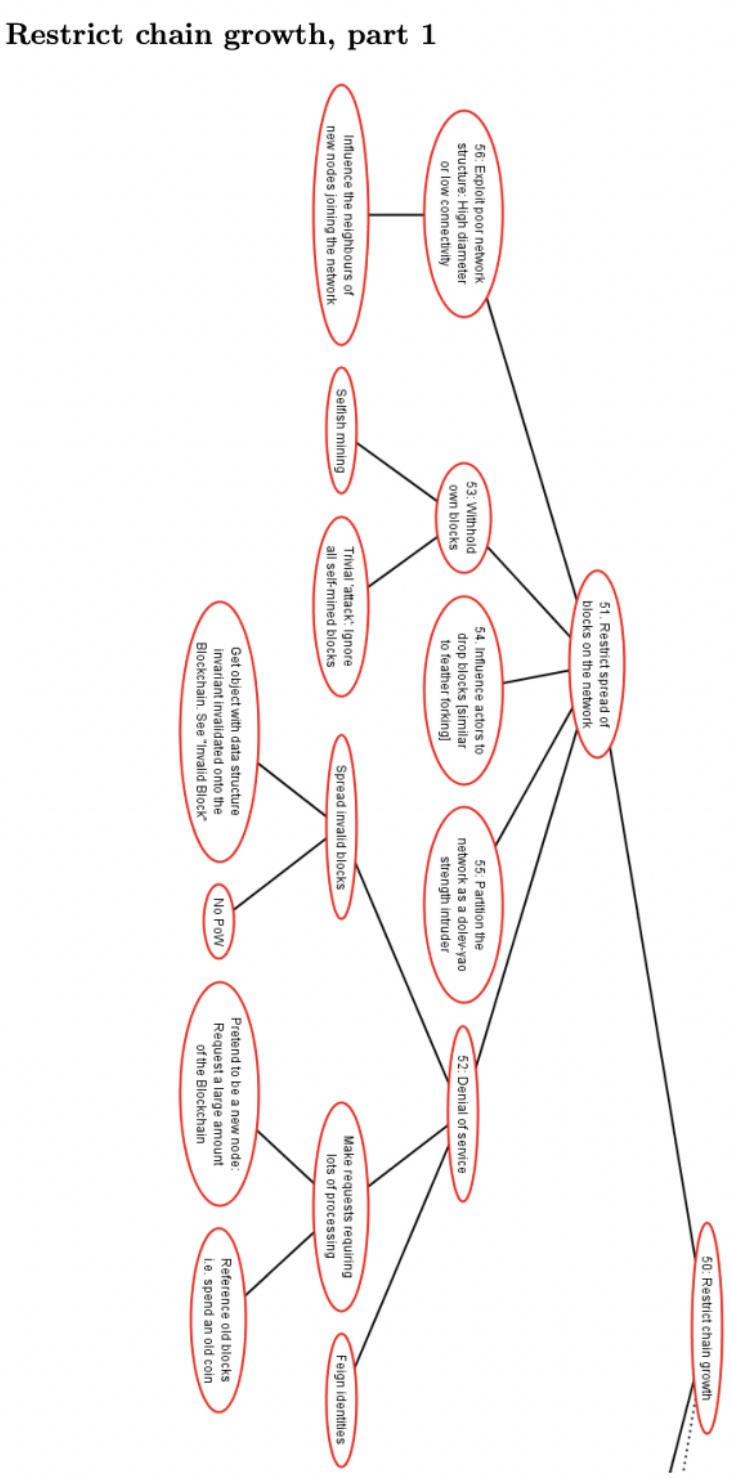


Figure 7: Restricting Chain Growth 1

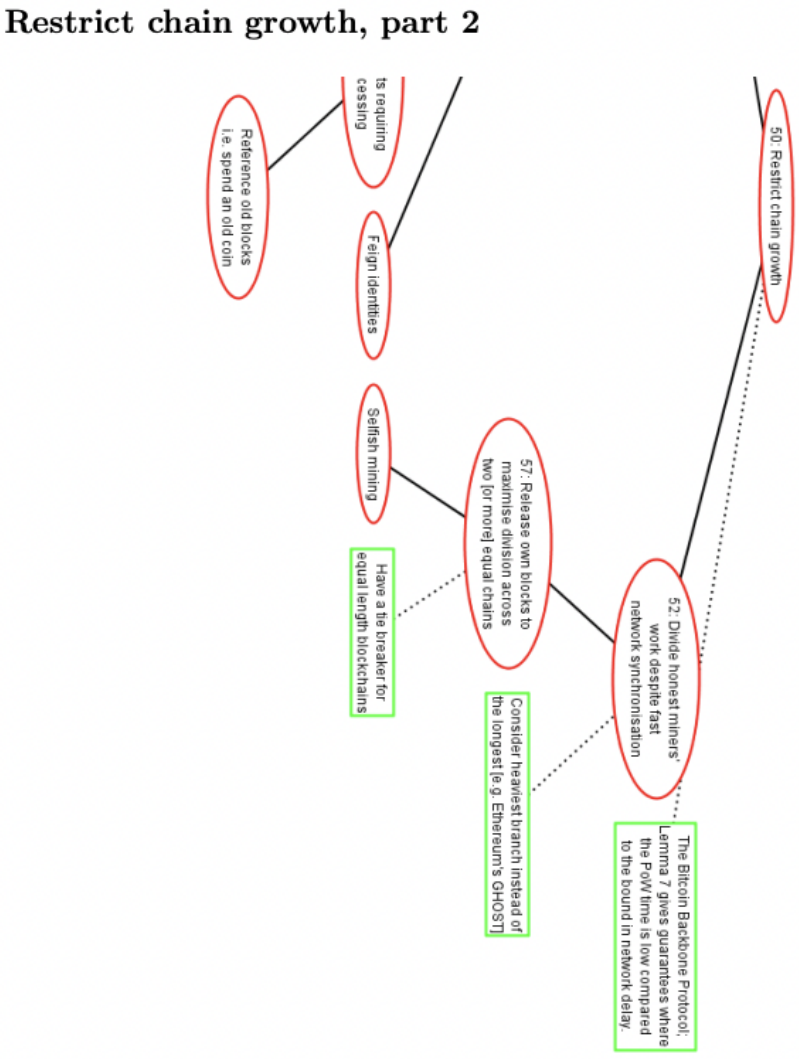


Figure 8: Restricting Chain Growth 2

References

[1] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. In: (May 2009). URL: http://www.bitcoin.org/bitcoin.pdf.

[2] *“Hashrate Distribution An estimation of hashrate distribution amongst the largest mining pools.* URL: https://www.blockchain.com/pools.

[3] Yourong Chen et al. “A survey on blockchain systems: Attacks, defenses, and Privacy Preservation”. In: *High-Confidence Computing* 2.2 (2022), p. 100048.

[4] Vishal; Chaudhary Kaylash; Chand and Ansgar Fehnker. “Double-Spending Analysis of Bitcoin”. In: *PACIS 2020 Proceedings. 210.* (2020). URL: https://aisel.aisnet.org/pacis2020/210.

[5] Mubashar Iqbal and Raimundas Matulevicius. “Exploring sybil and double-spending risks in Blockchain Systems”. In: *IEEE Access* 9 (2021), pp. 76153–76177.

[6] 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)* (2017).

[7] Liang Chen et al. “Phishing scams detection in Ethereum Transaction Network”. In: *ACM Transactions on Internet Technology* 21.1 (2021), pp. 1–16.

[8] AHMET FARUK AYSAN et al. “Survival of the fittest: A natural experiment from crypto exchanges”. In: *The Singapore Economic Review* (2021), pp. 1–20.

[9] Mubashar Iqbal and Raimundas Matulevičius. “Exploring Sybil and Double-Spending Risks in Blockchain Systems”. In: *IEEE Access* 9 (2021), pp. 76153–76177.

[10] Yujin Kwon et al. “Be Selfish and Avoid Dilemmas”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, Oct. 2017. URL: https://doi.org/10.1145%2F3133956.3134019.

[11] Jelena Mirkovic and Peter Reiher. “A taxonomy of ddos attack and ddos defense mechanisms”. In: *ACM SIGCOMM Computer Communication Review* 34.2 (2004), pp. 39–53.

[12] Hatem Ismail, Daniel Germanus, and Neeraj Suri. “Detecting and Mitigating P2P Eclipse Attacks”. In: Dec. 2015, pp. 224–231.

[13] Kuan Zhang et al. “Sybil Attacks and Their Defenses in the Internet of Things”. In: *Internet of Things Journal, IEEE* 1 (Oct. 2014), pp. 372–383.

[14] Dieudonné Mulamba, Indrajit Ray, and Indrakshi Ray. “Sybilradar: A graph-structure based framework for Sybil Detection in on-line social networks”. In: *ICT Systems Security and Privacy Protection* (2016), pp. 179–193.

[15] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. *BitIodine: Extracting Intelligence from the Bitcoin Network*.

[16] Philip Koshy, Diana Koshy, and Patrick McDaniel. "An analysis of anonymity in bitcoin using P2P network traffic". English (US). In: *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Revised Selected Papers*. Ed. by Reihaneh Safavi-Naini and Nicolas Christin. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Funding Information: This material is based upon work supported by the National Science Foundation Grants No. CNS-1228700 and CNS-0905447. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Publisher Copyright: © International Financial Cryptography Association 2014.; 18th International Conference on Financial Cryptography and Data Security, FC 2014 ; Conference date: 03-03-2014 Through 07-03-2014. Germany: Springer Verlag, 2014, pp. 469–485.

[17] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM J. Comput.* 18.1 (1989), pp. 186–208.

[18] Vitalik Buterin et al. *Combining GHOST and Casper*. 2020. eprint: arXiv:2003.03052.

[19] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *3rd Symposium on Operating Systems Design and Implementation (OSDI 99)*. New Orleans, LA: USENIX Association, Feb. 1999. URL: https://www.usenix.org/conference/osdi-99/practical-byzantine-fault-tolerance.

[20] Yonatan Sompolinsky and Aviv Zohar. "Secure high-rate transaction processing in bitcoin". In: *Financial Cryptography and Data Security* (2015), pp. 507–527.

[21] V. Zamfir. "Casper the friendly ghost: a "correct-by-construction" blockchain consensus protocol". 2017. URL: https://github.com/vladzamfir/research/blob/master/papers/%20CasperTFG/CasperTFG.pdf.

[22] Christopher Bellchambers. *A Block Structure Framework Attack Tree for Generic Blockchain Applications*. 2018.

[23] Joachim Neu, Ertem Nusret Tas, and David Tse. "Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma". In: *CoRR* abs/2009.04987 (2020). arXiv: 2009.04987. URL: https://arxiv.org/abs/2009.04987.

[24] Caspar Schwarz-Schilling et al. "Three Attacks on Proof-of-Stake Ethereum". In: *CoRR* abs/2110.10086 (2021). arXiv: 2110.10086. URL: https://arxiv.org/abs/2110.10086.

[25] Joachim Neu, Ertem Nusret Tas, and David Tse. *Avalanche Attack on Proof-of-Stake GHOST*. Jan. 2022. URL: https://ethresear.ch/t/avalanche-attack-on-proof-of-stake-ghost/11854.

[26] Joachim Neu, Ertem Nusret Tas, and David Tse. *Balancing Attack: LMD Edition*. Jan. 2022. URL: https://ethresear.ch/t/balancing-attack-lmd-edition/11853.

[27] Alastair Janse van Rensburg. "Generation and Analysis of Attack Graphs on Computer Networks". In: (Sept. 2018).