Chapter 4

Making Blockchains Secure

> Love all, trust a few.
>
> *William Shakespeare*

4.1 Introduction

In this chapter, we tackle the problem of security. We mentioned in Chapter 1 that a blockchain should prevent an attacker from double spending and we saw in Chapter 2 that violating consensus could lead to double spending but typically requires the attacker to own a large mining power, an attack which is very hard in a sufficiently large system.

We now explain, how one can double spend without owning a large portion of the mining power. In particular, we explain how a malicious user could violate the assumptions on which the previous consensus algorithms build and we describe the dramatic consequences such attacks could have. More specifically, we question whether the synchrony assumption is realistic and what an attacker needs to do to violate it. We then present attacks against mainstream blockchain systems that introduce and exploit network delays to double spend.

4.2 Beyond synchrony

The consensus algorithms we presented so far require synchrony, i.e., all messages must be transmitted in a known bounded time, say Δ. This synchrony assumption is convenient. For simplicity consider that the time it takes to do any local computation is negligible when compared to the time needed to exchange a message. It allows to know that a failure occurs if the message we expected in response to our message has not been received in 2Δ times.

In the context of blockchain, this synchrony assumption allows Bitcoin and Ethereum to know whether a transaction is committed. Consider that a block for index m of the chain is created at time at most t—say for example that p_i learns about the existence of this block at time t. Synchrony guarantees us that any reliable participant becomes aware of this block at the latest at time $t + \Delta$. As a result, all miners start mining a new block for index $\ell > m$ at time $t + \Delta$ and no new competing blocks will be proposed by correct miners for index m. At time $t + 2\Delta$, p_i is guaranteed to know the blocks proposed by the correct processes at index m. Provided that Byzantine miners do not have a sufficient mining power one can use this information to know which block at index m is part of the longest branch and hence identify, as depicted in Figure 2.1, the transactions that are committed.

Unfortunately, synchrony is unrealistic in practice as the time to transmit a message depends on the size of the message, the route that the message takes, the congestion of the network and other environmental factors that cannot be predicted. In 2004, the Computer Emergency Response Team Coordination Center at Carnegie Mellon University that collected computer security incidents in the US stopped doing so by arguing that attacks became commonplace. In 2017, typhoons damaged an undersea cable connecting Sydney to Hong Kong, which translated into a drop in quality of Internet connections in Australia. Human misconfigurations of the routing tables used by the Border Gateway Protocol to route traffic on the Internet often impacts messages delays

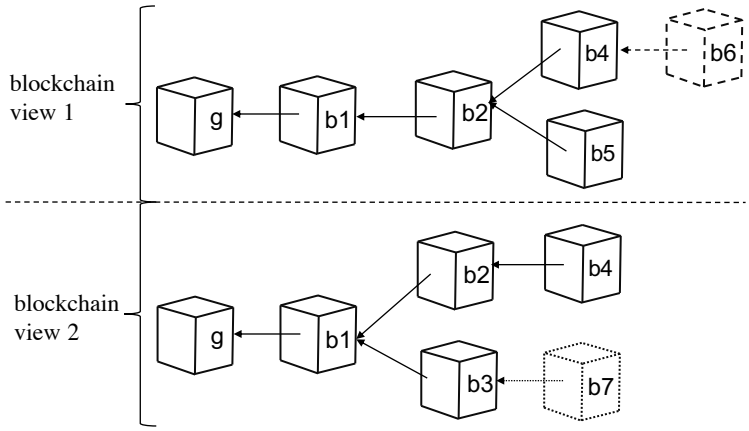

Figure 4.1: The Balance Attack consists of an attacker isolating two subgroups of nodes, one to which it sends a transaction, another to which it contributes blocks

over the Internet. So when malicious behaviors do not occur, either disasters or mistakes provoke unexpected message delays.

This problem is well-known and can be found in the eight fallacies of network and distributed systems. In particular, the network is reliable, the latency is zero, the bandwidth is infinite, the topology of the network does not change, the transport cost is zero and the network is homogeneous are well-known fallacies that are commonly assumed.

4.3 The Balance Attack

In the *Balance Attack* [NG16a], an attacker transiently disrupts communications between subgroups of Ethereum miners of similar mining power. During this time, the attacker issues transactions in one subgroup, say the *transaction subgroup*, and mines blocks in another subgroup, say the *block subgroup*, up to the point where the tree of the block subgroup outweighs, with high probability, the tree of the transaction subgroup.

Figure 4.1 depicts two blockchain views observed by two subgroups of the participating nodes. To double spend the attacker A sends a transaction in the bottom subgroup; consider that this transaction is included in block b_7. We refer to this bottom subgroup as the transaction T_A subgroup whereas we refer to the other one, the top subgroup, as the block subgroup. As the two subgroups have similar mining power, their blockchain views are expected to have similar weights and depths. It results that the attacker simply has to append few blocks to one of the two blockchain views to influence the decision regarding the view the system will eventually adopt. This is achieved by mining block b_6

in the block subgroup. When the subgroups start exchanging messages again, they both discard the bottom view as it does not contain the branch of largest weight or depth. The attacker can then re-spend the assets spent in T_A to complete the double spending.

Another option to achieve double spending in a similar situation would be to isolate a subgroup of smaller mining power than another subgroup, however, it would make the attack only possible if the recipients of the transactions are located in the subgroup of smaller mining power. Although possible this would limit the generality of the attack, because the attacker would be constrained on the transactions it can override.

Note that the Balance Attack inherently violates the persistence of the main branch prefix and is enough for the attacker to double spend. The attacker has simply to identify the subgroup that contains merchants and create transactions to buy goods from these merchants. After that, it can issue the transactions to this subgroup while propagating its mined blocks to at least one of the other subgroups. Once the merchant has shipped goods, the attacker stops delaying messages. Based on the high probability that the tree seen by the merchant is outweighed by another subtree, the attacker could reissue another transaction transferring the exact same coin again.

4.4 Double spending in Ethereum

In order to gain some insights regarding the public Ethereum blockchain we combined the name and block contributions of the top-10 mining pools as observed on `http://etherscan.io` during one week on August 3^{rd}, 2017 with their network connectivity information in Figure 4.2. To this end, to each named mining pool we registered a miner that could gather IP and *Autonomous System (AS)* information. ASes are groups of networks under the control of a single technical administration [HB96]. In particular, we estimated locations of the servers by querying five databases of IP geographic locations [DB, IP2, IP3, IP, IP4]. To reduce the inaccuracies of geographic locations, we extracted the location indicated in the majority of these databases. To retrieve the number and owner of each AS we relied on sources [CAI, Mer] based on the `whois` service. ASes have their own routing policy for internal traffic but use *Border Gateway Protocol (BGP)* [HRL06] for dynamic inter-AS routing. Unfortunately, BGP does not incorporate a mechanism to check whether an origin AS owns the IP prefixes that it announces. This makes the protocol vulnerable to route hijacking.

4.4.1 Double spending is easy in case of route hijacking

To quantify the risk of a partitioning attack, we emulated the aforementioned Ethereum 10 most powerful mining pools by deploying 10 virtual machines (VMs) linked through five BGP routers controlled in our private cloud infrastructure via OpenStack. To obtain the mining power distribution

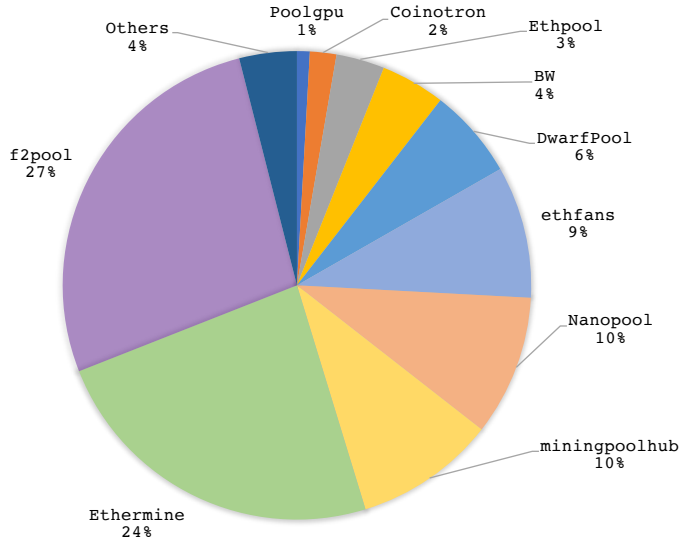

Figure 4.2: The portions of mining power of the most powerful mining pools of Ethereum

of mining pools retrieved in Figure 4.2 among our own VMs, we fixed the quantum of CPU time allocated to each machine using Linux control goups `cgroups` [Heo15]. The control groups allow us to specify the CPU quota Q that a VM can consume within a period of time T. Given the same value of T, we vary Q on all virtual machines based on their corresponding mining power percentage. As a result, we obtained the proportion we listed in Figure 4.2 close to one decimal.

We then combined a BGP-hijacking attack with the balance attack [NG17] to evaluate the risks of double spending in Ethereum v1.5. First, the BGP hijacking is used to delay communication, then the balance attack is used to turn these delays into double spending. To this end, we assign the role of the adversary to one of the mining pool in each of our attack instances. The adversary takes control over one BGP router to prevent two ASes from communicating with two other ASes during 7 minutes. During that time, the adversary issues a transaction to one group and contributes to the block creation of the other group in order to discard its previously issued transaction. Since it is commonly recommended to wait for 12 confirmations to be confident about the immutability of a transaction since the version Homestead of Ethereum [But16b], we consider the double spending successful when the transactions contained in a block followed by 11 consecutive blocks gets discarded.

After the 7 minutes communication delay, we observed whether the adversary transaction is discarded due to the choice of the canonical chain in 30

Table 4.1: Success of double spending with mining pools of similar power to the 10 most powerful Ethereum mining pools

Adversary mining power	Attacker subgroup mining power	Victim subgroup mining power	Double spending success
27%	34%	35%	77%
10%	43%	36%	57%

consecutive runs and concludes upon the average success of the attack. As indicated in Table 4.1, we observe that only 10% of the mining power is sufficient for the double spending to be successful most of the time. With only 27% of the mining power, the success of the attack reaches 76%. This result confirms the claim from the literature that it is supposedly feasible to attack public blockchains [AZV17], however, this is without taking into account the nature of the network topology. We explain in Section 4.4.2 why the topology described in Figure 4.2 makes the attack of the public Ethereum blockchain almost impossible.

4.4.2 Partitioning Ethereum mining pools turns out to be hard

While we showed that double spending could be easily achieved by partitioning public blockchains, it turns out that partitioning the mining power of Ethereum mining pools is actually very difficult.

Figure 4.2 lists the stratum servers of the 10 most powerful Ethereum mining pools we retrieved. We noticed experimentally that if one of the stratum servers becomes unresponsive, then the corresponding miners would connect to the next stratum server they operate in order to remain connected to the pool. Hence, attempts to partition miners may lead miners to reconnect to a different AS.

In addition, it is more difficult to determine the precise proportion of mining power connected to each stratum server, again due to the numerous stratum servers each miner operates. Indeed, a mining pool identifier is nothing more than the wallet address to receive reward when a pool successfully mines a block. While it is possible to determine a block miner by examining header information, there is no way to pin down to the stratum server, as long as these servers put their reward into the same wallet address.

Second, the stratum servers typically hide the location of the mining pool participants, which makes it hard to isolate a group of pools of a specific mining power. Without information about the miners for a stratum server, one cannot guarantee the partition success of a network attack. It may *(i)* isolate a stratum server along with its miners completely, *(ii)* partition some miners, which reduces only a fraction of computational power from the pool, or *(iii)* cut off the connectivity between a stratum server and pool participants, such that those participants decide to reconnect to different stratum servers.

Third, BGP-hijacking cannot affect the direct interconnection between ASes, because ASes are aware of static network prefixes that belong to their peer ASes. Apart from exchanging routes at the *Internet Exchange Points (IXPs)*, any pair of ASes may decide to establish either layer 2 or layer 3 links to connect their networks directly. This prevents dynamic routing attacks like the BGP hijacking we discussed above in Section 4.4.1. To better understand the applicability of the attack to the Ethereum public blockchain, we retrieved the direct peering information of the eight ASes we identified using available information [CAI17] and their interconnections. Among the 10 most powerful public Ethereum mining pools, 7 of them solely rely on this group of ASes; together, they account for more than 87% of the total mining power of the network. As the majority of ASes in this group are linked by direct peering, it appears extremely difficult to partition Ethereum's overlay. For example, `f2pool` may send and update to `ethfans` via a peering connection, which in turn forwards the update to `BW` via another peering connection. Without an adversary gaining access to configuration on the border routers of these ASes, it will remain difficult to partition a pool from the rest of the group.

4.5 Proof-of-Authority and Permissioned Sealers of Ethereum

Proof-of-Authority (PoA) was recently proposed as a Byzantine fault tolerant consensus mechanism that integrates with the Ethereum protocol. The Ethereum geth software offers two different consensus algorithms, called Clique and Istanbul BFT whereas the Ethereum parity software offers the consensus algorithm, called Aura. The concept is similar to traditional Byzantine fault tolerant consensus in that only n sealers are permissioned to create new blocks but requires authentication and strictly less than $\frac{n}{2}$ Byzantine participants. If Aura could be safe despite partial synchrony, then it would be an ideal replacement to the Ethereum's default consensus algorithm we presented earlier.

4.5.1 The Aura algorithm

Algorithm 7 depicts the way Aura guarantees that participating nodes reach consensus on the uniqueness of the block at a given index of the blockchain in `parity`. Every participating node maintains a state comprising a set of *sealers*, its current view of the blockchain c_i as a directed acyclic graph $\langle B_i, P_i \rangle$, a block b with fields *parent* that links to the parent block, a *sealer* and a *step* indicating the time at which the block is added to the blockchain, as explained below. Initially, they are $\perp$ meaning "undefined".

The function propose() is invoked in order to propose a block for a particular index of the blockchain. The consensus is reached once the block is decided, which can happen much later as we will explain in the function is-decided()

Algorithm 7 The parity Aura algorithm at process p_i

```
 1: State:
 2:    sealers ⊆ Ids, the set of sealers
 3:    c_i = ⟨B_i, P_i⟩, the local blockchain at node p_i is a directed
 4:       acyclic graph of blocks B_i and pointers P_i
 5:    b, a block record with fields:
 6:       parent, the block preceding b in the chain, initially ⊥
 7:       sealer, the sealer that signed block b, initially ⊥
 8:       step, the blockchain step when the block gets added, initially ⊥
 9:    step-duration, the duration of each step as configured
10:
11: propose()_i:
12:    while true do
13:       step ← clock-time()/step-duration
14:       if i ∈ sealers ∧ step mod |sealers| = i) then
15:          b.parent ← last-block(c_i)
16:          b.sealer ← p_i
17:          c_i ← ⟨B_i ∪ {b}, P_i ∪ {b.parent}⟩
18:          broadcast(c_i)
19:       sleep(step-duration)
20:
21: score(⟨B_j, P_j⟩):
22:    return UINT128_MAX × height(⟨B_j, P_j⟩) − step-num(⟨B_j, P_j⟩)
23:
24: deliver(⟨B_j, P_j⟩)_i:
25:    if score(⟨B_j, P_j⟩) > score(⟨B_i, P_i⟩) then
26:       ⟨B_i, P_i⟩ ← ⟨B_j, P_j⟩
27:
28: is-decided(b)_i:
29:    V ← {b_k.sealer | b_k ∈ B_i; k ≥ i}
30:    return (|V| × 2 > |sealers|)
```

(line 28) below. The algorithm discretises time into steps that corresponds to consecutive periods of step-duration time, as specified in a configuration file. Each sealer executes an infinite loop that periodically checks whether the clock-time() indicates that this is its turn to propose a block (line 13). When it is its turn (line 14), a sealer sets the parent of the block to the last block of its view and signs it (line 16).

Each broadcast() invoked by the propose() function sends blocks that get delivered to all other participating nodes that are honest (in reality only the last block is broadcast unless some sealer is unaware of more blocks). The deliver() function (line 24) is thus invoked at each honest participating node, regardless of whether it is a sealer, upon reception of the broadcast message. Once a blockchain view is delivered to p_i, the node compares the score of the blockchain view it maintains to the blockchain view it receives, using the score (line 21). The highest blockchain has the greatest score, however, if two blockchains share the same height, then the one that is denser in terms of its number of non-empty slots obtains the highest score. That is, among many blockchains with the same height, a blockchain whose last block has the the

lowest index wins.

This is indicated by the two functions height and step-num that represent the height of the blockchain and the number of slots for which there exists a block in the blockchain.

4.5.2 The Attack of the Clones

Aura, just like Clique, is vulnerable to a particular attack, called the Attack of the Clone, that can lead to double spending. In particular, the clone is a malicious sealer that joins, with a duplicate identity, two groups of correct sealers in order to create two apparent majorities of sealers that progress to decide upon conflicting blocks.

By assumption, only a minority of the sealers can be malicious in Aura, this is the reason why Aura seems to need only a majority of sealed blocks to consider whether a block and its transactions appear to be committed. As we explain below, $(2 - (n \bmod 2))$ attacker(s) cloning their own instance into two clones are actually sufficient to double spend.

The first step of the attack of the clones is for some attacking sealer to duplicate its Ethereum instance into two clones. This consists for the malicious sealer of running two instances of the Ethereum protocol with the same address or public-private key pair. Note that these two instances could either share the same IP address or use distinct IP addresses. We call these two instances clones because one has the same information as the other during the whole duration of the attack. Ethereum allows these two cloned instances to both create blocks, however, as they use the same private key to seal blocks, they are considered to act as a unique sealer.

The attacker must exploit message delays between two groups of a minority of $\lceil n/2 \rceil - 1$ sealers, hence creating a transient partition. Recall that this can be achieved using the network attack presented in Section 4.4.1. At this moment, the two clones may not share exactly the same database content as they may not be aware of the exact same blocks that are present in the blockchain. To maintain the cloning at the start of the partition, the attacker copies the content of the blockchain database of one of the clones to the database of the other clone and connects each of these clones to a different partition.

In order to progress towards a double spending situation, each partition must commit transactions and thus decide blocks, this is why we need $(2 - (n \bmod 2))$ attackers that clone instances. There are now two cases to consider depending on whether the number n of sealers is odd or even. If n is odd, then the honest sealers can be split into two groups of $(n-1)/2$ sealers, each representing a minority. Adding one clone to each minority is thus sufficient to obtain two majorities of $\lfloor n/2 \rfloor + 1$ sealers. If n is even, then with a single attacker $n - 1$ honest sealers would be split into two partitions of different sizes, one that contains $n/2$ sealers and another that contains $n/2 - 1$ sealers. Adding one clone in each partition would thus be insufficient to obtain two majorities, this is why we need $(2 - (n \bmod 2)) = 2$ attackers in this case.

In order to double spend, the attacker, say Alice, simply has to issue two conflicting transactions, one in each of the partition:

- T_A consists of Alice transferring all her coins to Bob whereas
- T'_A consists of Alice transferring all her coins to Carole.

The presence of the majority of sealers guarantees the progress of the protocol in both partitions so as to obtain the commit of a transaction T_A and T'_A, even though they conflict. If these transaction buy goods, then the goods are ready to be shipped to Alice. When the communication is re-enabled, the adopted branch contains only one of the these transactions. The double spending has been successful.

4.6 Accountability

In traditional blockchain systems it is hard to track the behavior of participants. The problem of double spending relies precisely on the fact that some blocks vanish from the system during a fork resolution. This naturally prevents other participants from being able to hold attackers accountable for their double spending—as there is no longer trace of the double spending when it completes. This lack of accountability is one of the key reasons why blockchains are so often subject to attacks.

Accountability has been proposed in the context of peer-to-peer systems as a property of the distributed system that allows correct nodes to detect the malicious activities of other nodes, hence holding nodes responsible for their actions. Unfortunately, distributed system accountability is hard to achieve as it requires nodes, especially Byzantine ones, to participate in the protocol by sharing voluntarily their logs. When synchrony cannot be assumed then malicious nodes are incentivized to not share their log to remain undetected. In particular, no correct nodes is able to distinguish a slow network delaying a log message from a malicious node refusing to share its log.

The Accountable Byzantine Agreement has thus been proposed as the more specific problem of guaranteeing that consensus is reached when possible, when $f < n/3$, but to detect Byzantine nodes when a disagreement is reached. The key idea is for correct nodes to treat received messages to progress in the consensus towards a decision only if they receive a signed justification for these messages, i.e., that they satisfy the protocol. If no justification is provided, then the associated message is simply ignored. It is then sufficient to cross-check these justifications to identify who has misbehaved. As a result, either malicious nodes do not justify themselves and no damages to the system happens, or they justify their messages and expose themselves to an eventual punishment.

The Accountable Byzantine Agreement problem is key to the security of blockchain systems, as accountability can incentivize nodes to behave correctly. Imagine a blockchain system that requests consensus participants to put assets

on a deposit, under the control of the blockchain system, prior to participating. In a rationale model, it is reasonable to expect the consensus participants to accept to deposit assets given that they typically gain a reward when they help providing the service by for example creating new blocks. Consider that the blockchain system builds upon an accountable Byzantine agreement algorithm and compensates the losses resulting from double spending attacks by withdrawing assets from the Byzantine nodes that provoked this disagreement. In a rational model, a node will certainly think twice before taking the risk of losing its deposit before acting maliciously to try and double spend. This accountable Byzantine agreement thus makes the participants of the blockchain system accountable.

4.7 Conclusion

This chapter presented a fully implemented attack against blockchain that incorporates both components of a network attack and an asset loss using double spending. This attack has been evaluated against the Ethereum blockchain system in public, consortium and private contexts. In the public context, on real-world data, we discussed the feasibility of a network partitioning attack. We found that while such an attack is provably possible, the risks of succeeding in stealing assets remains extremely low on the existing main chain. When Ethereum is deployed over a WAN in a consortium environment, however, we demonstrated that an adversary could easily double-spend through BGP hijacking, with a double-spending success rate up-to 80%.

There exist a set of counter-measures to the attacks presented above. The simplest one is perhaps to design solutions that do not require synchrony as messages cannot be all delivered in a known and fixed delay over large networks. We present such a solution in Chapter 5. We have discussed accountable solutions that incentivize blockchain participants to behave correctly. These solutions are particularly promising and would benefit from a game theoretical treatment.

4.8 Bibliographic notes

A survey of attacks against blockchain networks can be found in [NYGEV19] and network requirements have been listed before [DPS+14]. Many blockchains assume synchrony where all messages must be delivered in less than a known bounded time [Nak08, Woo15, LNZ+16, AMN+17, KKJG+17a, GHM+17, HMW18, ZMR18]. The drawback is that if some messages experience an unforeseen delay, then the blockchain guarantees are violated. Algorand [GHM+17] and Dfinity [HMW18] use randomization to restrict the task of deciding a block to a small subset. Elastico [LNZ+16] proposes a sharded consensus partitioned into sub-committees to run more but small-scale consensus instances. RapidChain [ZMR18] improves Elastico's shard-

ing by using a randomized selection and erasure coded gossip for scalability. Other blockchains [KJG+16, AMN+17, KKJG+17b, Roc18] assume synchrony for some operations, but use a Byzantine consensus that would work with partial synchrony [DLS88]. Solida [AMN+17] assumes synchrony but builds upon PBFT [CL02]. Avalanche [Roc18] is analyzed under synchrony but is conjectured to work in a partially synchronous network. OmniLedger [KKJG+17b] reorganizes nodes among shards using randomization to avoid attackers taking control of individual shards but needs an external consensus protocol. To assign participants to shard, Omniledger uses an identity protocol that requires synchronous communication channels.

Godel et al. [GKKT16] analyzed the effect of propagation delays on Bitcoin using a Markov process. Garay et al. [GKL15] investigated Bitcoin in the synchronous communication setting, however, this setting is often considered too restrictive [Cac01]. Pass et al. extended the analysis for when the bound-on message delivery is unknown and showed in their model that the difficulty of Bitcoin's crypto-difficulty has to be adapted depending on the bound on the communication delays [PSS16].

The problem is that various attacks can be used to delay messages beyond the expected synchronous bound of all message delays: The fallacies of distributed computing were first proposed in the 90's by L. Peter Deutsch, and Bill Joy and Tom Lyon under the name of fallacies of networked computing. It has been extended later by James Gosling. Some statistics about the CERT/CC on the fact that network attacks have become commonplace can be found at https://www.cdrinfo.com/d7/content/certcc-statistics-1988-2004#incidents.

Several attacks benefited from an attacker able to attack the communication graph [PSS16, SZ15, NKMS16]. Decker and Wattenhoffer already observed that Bitcoin suffered from block propagation delays [DW13]. To hack blockchains without a significant mining power, hackers and researchers already thought of attacking the network [LS14, HKZG15, NG16b, NG17, EGJ20]. In 2014, a BGP hijacker exploited access to an ISP to steal $83000 worth of bitcoins by positioning itself between Bitcoin pools and their miners [LS14]. The Eclipse attack against Bitcoin [HKZG15] consists of isolating at the IP layer a victim miner from the rest of the network to exploit its resources by proposing new neighbors to a miner until it accepts to connect only to nodes under the control of the attacker. It showed that an attacker controlling 32 IP addresses could "eclipse" a Bitcoin node with 85% probability. While a Bitcoin node typically connects to 8 logical neighbors, an Ethereum node typically connects to 25 nodes, making the attack harder. The Blockchain Anomaly [NG16b] exploits message reordering in Ethereum to rollback committed transactions and double spend. The Balance Attack [NG17] partitions the network into groups of similar mining power to influence the selection of the canonical chain and double spend. Recently, actual man-in-the-middle attacks were run to demonstrate the feasibility of stealing assets in Ethereum without a significant mining power [EGJ18].

The Attack of the Clones and in particular the Aura algorithm (Algorithm 7) is taken from [EGJ20]. This attack was acknowledged by the security team of

Parity Technology and the security team of the Ethereum foundation and was also presented to the Ethereum development community [EGJ19]. The xDai blockchain of the POSDAO project has been working on the implementation of one of the counter-measures presented in [EGJ20].

The idea of accountability in distributed systems was pioneered by Haeberlen, Kuznetsov, and Druschel [HKD07]. In a partially synchronous system, they guarantee that faulty processes will be suspected forever, though definitive evidence may not be obtained. The problem of Accountable Byzantine Agreement was defined in [CGG19], and a solution called Polygraph for the partially synchronous system was proposed: it accurately detects at least $n/3$ Byzantine nodes when disagreement occurs. Polygraph is at the heart of the Long-Live Blockchain [RPG20] that recovers from forks by excluding Byzantine consensus nodes to tolerate more than $n/3$ Byzantine faults. Polygraph can be considered as the accountable variant of the Democratic BFT consensus algorithm [CGLR18] presented in Chapter 5.

Bibliography

[AMN+17] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solida: A blockchain protocol based on reconfigurable byzantine consensus. In *Proc. of the 21st International Conference on Principles of Distributed Systems, (OPODIS)*, pages 25:1–25:19, 2017.

[AZV17] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *IEEE S&P 2017*, pages 375–392, 2017.

[But16] Vitalik Buterin. How should i handle blockchain forks in my dapp?, 1 2016. https://ethereum.stackexchange.com/questions/183/how-should-i-handle-blockchain-forks-in-my-dapp/203/#203.

[Cac01] Christian Cachin. Distributing trust on the internet. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 183–192, 2001.

[CAI] CAIDA: Center for Applied Internet Data Analysis.

[CAI17] The CAIDA AS Relationships Dataset, August 2017.

[CGG19] Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable byzantine consensus. In *Workshop on Verification of Distributed Systems (VDS'19)*, Jun 2019.

[CGLR18] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: Efficient leaderless byzantine consensus and its applications to blockchains. In *Proceedings of the 17th IEEE International Symposium on Network Computing and Applications (NCA'18)*, 2018.

[CL02] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[DB] DB-IP - IP Geolocation and Network Intelligence.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[DPS+14] Maya Dotan, Yvonne-Anne Pignolet, Stefan Schmid, Saar Tochner, and Aviv Zohar. Survey on cryptocurrency networking: Context, state-of-the-art, challenges. Technical Report 1409.6606, arXiv, 2014.

[DW13] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Proc. of the IEEE International Conference on Peer-to-Peer Computing*, pages 1–10, 2013.

[EGJ18] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. Impact of man-in-the-middle attacks on ethereum. In *SRDS*, 2018.

[EGJ19] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The attack of the clones against proof-of-authority. In *Community Ethereum Development Conference (EDCON'19)*, 2019.

[EGJ20] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The Attack of the Clones against Proof-of-Authority. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'20)*. Internet Society, Feb 2020.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, 2017.

[GKKT16] J. Göbel, H.P. Keeler, A.E. Krzesinski, and P.G. Taylor. Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. *Performance Evaluation*, Juy 2016.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Technique (EUROCRYPT)*, pages 281–310, 2015.

[HB96] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS), March 1996.

[Heo15] Tejun Heo. Control Group v2, October 2015.

[HKD07] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. *SOSP*, 2007.

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium*, pages 129–144, 2015.

[HMW18] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. Technical Report 1805.04548, arXiv, May 2018.

[HRL06] Susan Hares, Yakov Rekhter, and Tony Li. A Border Gateway Protocol 4 (BGP-4), January 2006.

[IP] IP Address to Identify Geolocation Information.

[IP2] IP Address Details - ipinfo.io.

[IP3] IP Address Geolocation to trace Country, Region, City, ZIP Code, etc.

[IP4] IP Geolocation and Online Fraud Prevention | MaxMind.

[KJG+16] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, Austin, TX, 2016. USENIX Association.

[KKJG+17a] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger. Technical Report 2017/405, Cryptology ePrint, 2017.

[KKJG+17b] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. Cryptology ePrint Archive, Report 2017/406, 2017. https://eprint.iacr.org/2017/406.

[LNZ+16] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 17–30, 2016.

[LS14] Pat Litke and Joe Stewart. BGP hijacking for cryptocurrency profit, August 2014.

[Mer] Merit RADb.

[Nak08] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008. http://www.bitcoin.org.

[NG16a] Christopher Natoli and Vincent Gramoli. The balance attack against proof-of-work blockchains: The R3 testbed as an example. Technical Report 1765133, arXiv, 2016.

[NG16b] Christopher Natoli and Vincent Gramoli. The blockchain anomaly. In *Proceedings of the 15th IEEE International Symposium on Network Computing and Applications (NCA'16)*, pages 310–317, Oct 2016.

[NG17] Christopher Natoli and Vincent Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *Proceedings of the 47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'17)*, June 2017.

[NKMS16] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 305–320, 2016.

[NYGEV19] Christopher Natoli, Jiangshan Yu, Vincent Gramoli, and Paulo Esteves-Verissimo. Deconstructing blockchains: A comprehensive survey on consensus, membership and structure. Technical Report 1908.08316, arXiv, 2019.

[PSS16] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. Technical Report 454, Crytology ePrint Archive, 2016.

[Roc18] Team Rocket. Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies, 2018. Unpublished manuscript.

[RPG20] Alejandro Ranchal-Pedrosa and Vincent Gramoli. Blockchain is dead, long live blockchain! accountable state machine replication for longlasting blockchain. Technical Report 2007.10541, arXiv, 2020.

[SZ15] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, pages 507–527, 2015.

[Woo15] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2015. Yellow paper.

[ZMR18] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. Cryptology

ePrint Archive, Report 2018/460, 2018. https://eprint.iacr.org/2018/460.