Attacking Transaction Relay in MimbleWimble Blockchains

by

Seyed Ali Tabatabaee

Bachelor of Science, Sharif University of Technology, 2019

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

August 2021

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Attacking Transaction Relay in MimbleWimble Blockchains

submitted by **Seyed Ali Tabatabaee** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

Ivan Beschastnikh, Computer Science
Co-supervisor

Chen Feng, Engineering (Okanagan Campus)
Co-supervisor

Abstract

Blockchain-based networks are often concerned with privacy. Two common types of privacy in blockchain networks are (1) transaction source privacy, and (2) transaction content privacy. Research has shown that Bitcoin, the most prominent cryptocurrency, cannot easily provide these privacy types. Hence, new protocols have been proposed. For example, Dandelion++ is a solution to the source privacy vulnerability in Bitcoin. Practical systems, however, need to provide multiple privacy guarantees at the same time. To the best of our knowledge, source privacy and content privacy have not been considered simultaneously in the literature. We conjecture that cryptocurrencies that use Dandelion++ for transaction relay could be susceptible to attacks against both types of privacy and also to performance attacks. Our focus in this project is on the implementations of the MimbleWimble cryptocurrency protocol such as Beam. We have designed and implemented three different attacks against these existing privacy-focused protocols. In the first attack, the adversary uses information obtained from the content of an incoming transaction for improved detection of the transaction source. In the second attack, to increase the latency of an incoming transaction, the adversary adds an excessive delay before forwarding the transaction. In the third attack, the adversary exploits the aggregation protocol in MimbleWimble to launch a denial of service attack on an incoming transaction. We have validated our proposed attacks in a private test network of Beam nodes and a network simulator.

Lay Summary

Blockchain systems have been used for a variety of applications, such as decentralized digital currencies. Similar to other financial systems, blockchain-based digital currencies are concerned with privacy. There are different types of privacy guarantees that users of blockchain-based digital currencies care about and that blockchain networks must provide. Two common types of privacy in blockchain networks are transaction source privacy and transaction content privacy. Research has shown that the most prominent blockchain-based digital currencies such as Bitcoin cannot easily provide these privacy types. Therefore, several privacy-focused cryptocurrencies such as MimbleWimble have been proposed. In this project, we have proposed and implemented different attacks against the implementations of MimbleWimble. Our proposed attacks exploit some of the privacy-enhancing mechanisms used in these cryptocurrency protocols to compromise the source privacy of transactions and the performance of the protocols. We have evaluated the attacks in a private test network and a network simulator.

Preface

The entire work presented in this thesis is original, unpublished, independent research conducted by the author, Seyed Ali Tabatabaee, in collaboration with the partner organization, Aquanow, the Beam developer community, and Charlene Nicer under the supervision of Dr. Ivan Beschastnikh and Dr. Chen Feng. Aquanow passed on their experience regarding the maintenance of blockchain network nodes. The Beam developer community and Charlene assisted with the implementation of the private test network and extraction of information from the network nodes.

Contents

List of Tables

List of Figures

Acknowledgments

First, I would like to thank my supervisors, Dr. Ivan Beschastnikh and Dr. Chen Feng for their invaluable support and guidance throughout the project.

I would also like to thank all members of Blockchain@UBC, especially the lead of the cluster, Dr. Victoria Lemieux. Blockchain@UBC provided me with many opportunities to engage in research and education in the area of blockchains. Moreover, I want to thank our partner organization, Aquanow, the Beam developer community, and Charlene Nicer for their contributions to the project. I am also thankful to Gleb Naumenko and Fangyu Gai for sharing with me their knowledge in the area of blockchains.

My research was supported by the Department of Computer Science at the University of British Columbia, Blockchain@UBC research cluster, the Natural Sciences and Engineering Research Council of Canada (NSERC), Mitacs, and Aquanow. Many thanks to all of them.

Last but not the least, I would like to thank my family and friends. They were there for me through all the ups and downs and I am forever grateful for having them in my life.

Chapter 1

Introduction

Blockchain systems, first introduced in Bitcoin [27], have been used for a variety of applications, such as decentralized digital currencies. Bitcoin, introduced in 2008 by Satoshi Nakamoto, is the first cryptocurrency that incorporates a Proof of Work (PoW) algorithm to achieve a decentralized consensus on transactions. Bitcoin has gained huge popularity over the years, with a market capitalization of over 1 trillion USD as of April 2021 [3].

Several challenges of blockchain systems including incentive compatibility, applicability, scalability, efficiency, and privacy have been subjects of extensive research. Many research papers have analyzed incentive structures in Bitcoin [11, 13, 20, 35]. Using smart contracts, Ethereum [40] and some other blockchain protocols have provided a variety of applications beyond simple financial transactions. Aiming to achieve scalability and efficiency, some blockchain protocols incorporate alternative consensus algorithms, for instance, Proof of Stake (PoS) [21] or Byzantine Fault Tolerant (BFT) consensus [12]. Algorand [18] is an example of such protocols which in fact combines the ideas of the two aforementioned consensus algorithms. Some other distributed ledgers, instead of a chain, utilize different data structures, for example, a tree [37] or a directed acyclic graph (DAG) [7, 38].

Last but not least, similar to other financial systems, blockchain-based digital currencies are concerned with privacy [5]. Protocols such as Monero [29], Zcash [19], and MimbleWimble [32] have introduced various techniques to enhance the privacy of users and transactions. There are different types of privacy guarantees that users of blockchain-based networks, particularly public blockchains, care about and that these networks provide. One type of privacy is transaction source privacy. This privacy aims to hide the source of the transaction in the system. Another type of privacy is transaction content privacy. This privacy may be achieved with encryption techniques, as well as aggregation approaches that combine several transactions into a single transaction and make it difficult to precisely reconstruct the set of transactions. Different research papers have proposed techniques to improve the source [33] and content [26] privacy of transactions in blockchain networks.

In Bitcoin, similar to many other blockchain networks, transactions and other messages are relayed over a peer-to-peer network of TCP links. Flooding [31] which is the typical approach to broadcast transactions has been shown to potentially reveal the source IP address of the transactions to the adversarial nodes [8]. Hence, Bitcoin Core [2] which is the most popular implementation of Bitcoin incorporates a relay protocol called diffusion. Diffusion adds independent exponential delays before broadcasting the transactions from each node in order to make deanonymization attacks more difficult. However, this protocol has also been shown to offer insufficient transaction source privacy [14]. Furthermore, the Bitcoin protocol is also susceptible to attacks against content privacy [25].

In order to improve the source privacy of transactions in Bitcoin, protocols such as Dandelion [9] and Dandelion++ [15] constrain the number of neighbors that a node will send a transaction to during some stage of transaction relay. Particularly in these two protocols, transactions are first relayed through stem paths, where each node passes the transaction only to one of its neighbors. This way, only one node in the network will receive the transaction directly from the transaction source and most of the nodes will receive the transaction when it has already passed through multiple nodes, improving source privacy.

MimbleWimble [32], the privacy-focused blockchain protocol that we are going to analyze in this work, uses confidential transactions [24] to encrypt the amounts of transactions. Moreover, somewhat similar to CoinJoin [23], MimbleWimble allows for the aggregation of several transactions into one transaction to enhance the content privacy of transactions. Grin [4] and Beam [1] are the two major implementations of MimbleWimble. Both of these cryptocurrency protocols use Dandelion++ for transaction relay to improve the transaction source privacy in MimbleWimble.

In this work, we analyze the transaction relay protocol in MimbleWimble. We implement and validate three different attacks to understand their impact on MimbleWimble and its implementation in Beam. The attacks are briefly described as follows:

1. **Improved transaction source detection:** The aggregation in MimbleWimble leaks some information. If a transaction is aggregated, then it must have passed through an aggregating node after it was generated. If the transaction is not aggregated, then it has not passed through such an aggregating node. We can use this information to increase the precision of the first node detection attack [22] which outputs the first honest node to forward a transaction to an adversarial node as the source of that transaction.

2. **Delaying transaction relay:** To improve the source privacy of transactions, protocols such as the ones based on Dandelion++, at some stages of the transaction relay, constrain the number of neighbors that a node will send a transaction. Such constraints improve privacy, but they also leave the protocol vulnerable to attacks on transaction latency, specifically at the bottleneck locations along the relay paths.

3. **Transaction denial of service with aggregations:** By aggregating different incoming transactions with a newly generated transaction that has not been mined into a block, the attacker with the cost of one transaction fee can prevent all of the aggregations from ending up in a block. This denial of service attack will negatively impact the transaction throughput of the network.

This thesis is structured as follows. Chapter 2 provides background information on Bitcoin, Dandelion++, and MimbleWimble. Chapter 3 offers an in-depth look into Beam. Chapter 4 describes our threat model. Chapter 5 provides a comprehensive description of each of the three proposed attacks. Chapter 6 provides details on our simulation of the Beam protocol, the implementation of our proposed attacks, and the Beam private network that we created for testing. Chapter 7 presents the results of our simulations and the evaluation of our proposed attacks.

Chapter 8 provides a further discussion on the results of our proposed attacks and suggests mitigations to those attacks. Finally, Chapter 9 concludes the results presented in this thesis.

Chapter 2

Background

Before we proceed to the description and analysis of our proposed attacks on MimbleWimble and its implementations, here we provide important background information on Bitcoin (Section 2.1), Dandelion++ (Section 2.2), and MimbleWimble (Section 2.3).

2.1 Bitcoin

Bitcoin [27] is the most prominent decentralized digital currency. In Bitcoin, transactions are verified and maintained by the Bitcoin network nodes in a distributed fashion. The Bitcoin blockchain grows as new blocks, containing transactions already sent to the peer-to-peer network of Bitcoin, are added every ten minutes on average by the Bitcoin miners. Bitcoin uses a Proof of Work (PoW) algorithm to achieve a decentralized consensus on transactions. Since the security of the protocol depends on the total mining power and mining is expensive, Bitcoin miners are compensated through block rewards and transaction fees.

A Bitcoin transaction contains a list of inputs and a list of outputs. Each input has a reference to an output from some preceding transaction. It also includes the public key and signature of the owner of the input. A normal output contains a value and a Bitcoin address. Therefore, the amounts and addresses of inputs and outputs in transactions are publicly visible. This would suggest the susceptibility of Bitcoin transactions to attacks against content privacy. In fact, previous research has shown that a lot of information can be extracted from the transaction graph of Bitcoin, and different transactions of the same user can be linked together [25, 30, 34]. Consequently, several blockchain protocols that offer enhanced content privacy, such as Monero [29], Zcash [19], and MimbleWimble [32], have been proposed.

In Bitcoin, transactions are relayed over a peer-to-peer network of TCP links. Each node in the network is identified by its IP address and port, and each honest node has a specified upper bound on the number of connections it can have in the network. Bitcoin Core [2] uses a relay protocol called diffusion. With diffusion, each node adds an independent exponential delay before advertising an incoming transaction to all of its peers except the one that previously sent the transaction or the ones that sent an advertisement for the transaction. Several alternatives to this relay protocol have been proposed, such as Erlay [28] which reduces the bandwidth consumption, thereby improving the security, and Dandelion [9] and Dandelion++ [15] which enhance the transaction source privacy.

2.2 Dandelion++

Dandelion++ is a transaction relay protocol based on a preceding proposal called Dandelion. The two protocols have similar goals but subtle differences in implementation choices. To improve the source privacy of transactions, Dandelion++ constrains the number of neighbors that a node will send a transaction to at the beginning of the transaction relay. With Dandelion++, transactions are relayed in two phases. First, in the stem phase, each node when receives a transaction, passes the transaction only to one other node. Then, in the fluff phase, each node sends an incoming transaction to all of its neighbors except the one that initially sent the transaction. Figure 2.1 illustrates the two phases of transaction relay in Dandelion++.

Compared to diffusion, with Dandelion++, adversarial nodes have less chance of receiving a transaction directly from its source and it is more difficult for the adversary to localize the source of the transaction. The probability of transitioning to the fluff phase in each step of the stem phase is a parameter of the protocol. The lower this probability is, the longer the average length of stem paths will be. Long

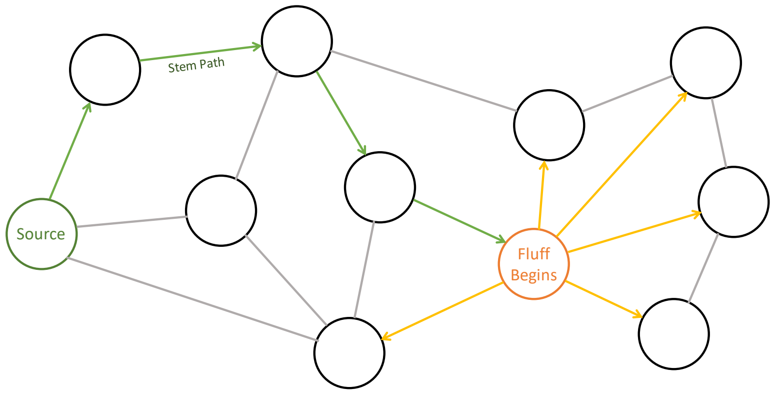


Figure 2.1: The two phases of transaction relay in Dandelion++. A transaction originated in node *Source* is first relayed through a stem path. Then, the fluff phase begins and each node that receives the transaction sends it to all of its neighbors except the one that initially sent the transaction.

stem paths lead to high protection of the transaction source privacy. To mitigate black-hole attacks where the adversarial nodes decide not to forward the incoming stem transactions, Dandelion++ incorporates a fail-safe mechanism. Each node in the stem path of a transaction would fluff the transaction itself if it does not receive the fluff version by the expiration of the independent random timer it sets for the stem transaction.

2.3 MimbleWimble

MimbleWimble is a cryptocurrency protocol that uses encryption and aggregation to enhance the content privacy of transactions. Compared to other cryptocurrency protocols such as Bitcoin, MimbleWimble has the following important advantages:

- Amounts of inputs and outputs in transactions are encrypted.
- Aggregation of transactions makes it difficult to link the inputs and outputs.
- The size of the blockchain is substantially reduced through the deletion of spent outputs.

MimbleWimble uses confidential transactions to encrypt the amounts. Instead of obvious amounts, the commitments of inputs and outputs are put into transactions and kept on the blockchain. Each commitment is in the form of

$$C = r \cdot G + v \cdot H$$

where C is a Pedersen commitment, v is the amount, r is a secret random blinding key which should be known only to the owner, and G and H are fixed Elliptic Curve Cryptography (ECC) group generators known to all. A range-proof is attached to each output commitment which proves that its amount is valid. The r value in commitments with explicit amounts, such as transaction fees and block rewards, is zero. The owner of a set of outputs is who knows the sum of their r values. Knowing the sum of r values for a set of outputs, one can create a valid transaction that spends those outputs. For a transaction to be valid, the commitments in that transaction should sum to zero and the range-proofs for the output commitments should be valid.

To prevent the sender of inputs in a transaction from spending the outputs, the sum of r values for the outputs should differ from the sum of r values for the inputs. Therefore, the commitments of inputs and outputs in each transaction should sum to a non-zero value $k \cdot G$ (kernel) where k is chosen by the recipient. A kernel is a non-spendable commitment with zero amount. A transaction is allowed to have more than one kernel. Hence, the sum of commitments in a transaction is

$$\sum_{C_i \in inputs} C_i + \sum_{C_o \in outputs} C_o + \sum_{C_k \in kernels} C_k = 0.$$

Aggregation of transactions makes it difficult to link the inputs and outputs. Since the sum of commitments in each valid transaction is zero, the total sum of commitments for multiple transactions is still zero. Therefore, the aggregation of multiple valid transactions is a valid transaction (Figure 2.2). Because each block consists of some valid transactions, a block can be interpreted as a single aggregated transaction.

The sum of all commitments in each block is zero. Hence, the sum of all commitments in the blockchain is also zero. An output of a transaction can be spent in the succeeding blocks and appear as an input of another transaction. The

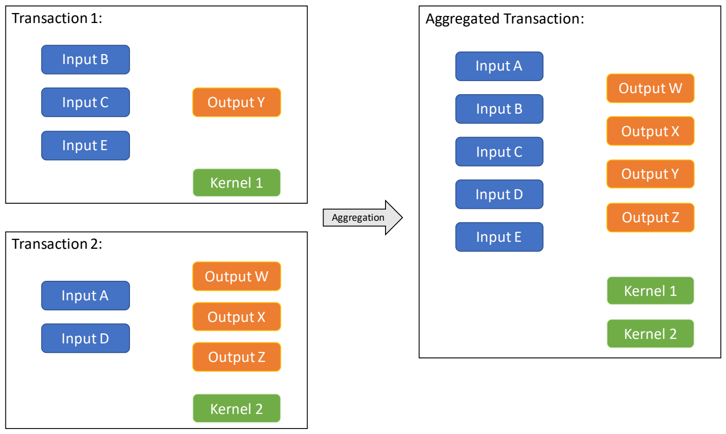


Figure 2.2: Aggregation of two valid transactions.

sum of commitments for this pair of input and output is zero. Consequently, if both commitments are removed from the blockchain, the sum of all commitments in the blockchain remains zero. Therefore, it is possible to safely remove a spent output and its corresponding input from the blockchain (Figure 2.3). Using this technique, the size of the blockchain can be substantially reduced. The only elements that remain in the blockchain are the explicit amounts for block rewards, kernels for all transactions, and unspent outputs along with their range-proofs and Merkle proofs.

Although the original proposal did not specify a transaction relay protocol for MimbleWimble, the two major implementations of this protocol, Grin [4] and Beam [1], have incorporated Dandelion++, where transactions are aggregated in the stem phase. Using this approach, not only do these cryptocurrencies attempt to improve the transaction source privacy, but also they try to make it difficult to link the inputs and outputs of transactions by first relaying them through stem paths and reducing the number of network nodes that observe them before aggregation. Besides, Bulletproofs [10] which are short proofs for confidential transactions have been proposed to improve on the original range-proofs. Bulletproofs have been adopted by both Grin and Beam. Other research projects have provided a provable-security analysis for MimbleWimble [16] and stepped toward a formalization of the MimbleWimble cryptocurrency protocol and the verification of its implementations [6, 36]. In this project, we focus on the vulnerabilities of transaction relay

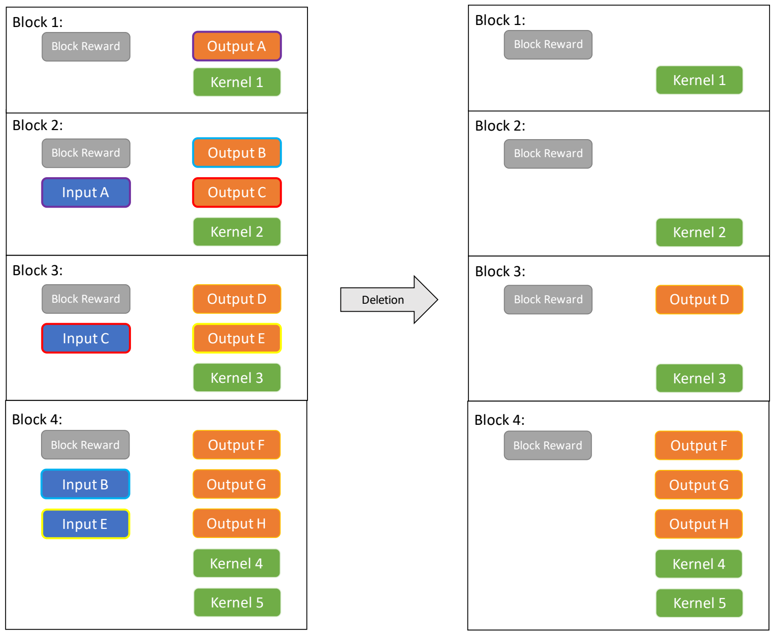


Figure 2.3: Deletion of spent outputs and their corresponding inputs from the blockchain.

in the implementations of MimbleWimble.

Chapter 3

Beam Analysis

Since we use the implementation of MimbleWimble in Beam for the purpose of validating our proposed attacks, it is important to have an in-depth understanding of Beam. Hence, in this chapter, we describe the implementation of transaction relay in Beam (Section 3.1) and provide details on the Beam main network (Section 3.2).

3.1 Transaction Relay Protocol

In this section, we describe Beam's transaction relay protocol. We provide an overview of the life cycle of a transaction from when it is received by a node to when it is forwarded to the peers of the node. For that purpose, we use pseudocode that we have written based on the source code of Beam. All the pseudocode presented in this section have been obtained from the node/node.cpp file in the "mainnet" branch of Beam's GitHub repository as of February 2021 [1].

There are six important functions that every Beam node uses to manage and forward incoming transactions. We describe all of these functions in this section. These are also the functions that we need to modify in the source code to implement our proposed attacks. Table 3.1 presents the values for the Beam network parameters that are used in the functions that we describe.

[1] https://github.com/BeamMW/beam/commit/77c3aa5a60fad3a6affdee3953dbb41208ffdc41

Name	Value
FluffProbability	0.1
TimeoutMin	20s
TimeoutMax	50s
AggregationTime	10s
OutputsMin	5
OutputsMax	40

Table 3.1: The Beam network parameters.

When a new transaction is received, the function *OnTransaction* will be called. The pseudocode for this function is presented in Algorithm 1. This function calls either *OnTransactionStem* (Algorithm 2) or *OnTransactionFluff* (Algorithm 5) based on the type of the incoming transaction.

Algorithm 1 OnTransaction

```
1: function OnTransaction(Transaction $tx$)
2:     if $tx$ is stem then
3:         OnTransactionStem($tx$)
4:     else
5:         OnTransactionFluff($tx$)
```

If the incoming transaction is a stem transaction, then the function *OnTransactionStem* will be called. The pseudocode for this function is available in Algorithm 2. This function compares the new stem transaction to transactions in the node's stempool (data structure containing valid stem transactions that have not been fluffed) and checks the validity of the new transaction. If the new stem transaction is accepted, then the stempool will be updated and the new transaction will also be added to it. Eventually, if the number of outputs in the transaction is greater than or equal to *OutputsMax*, then the transaction does not need to be aggregated any further; hence, the function *OnTransactionAggregated* (Algorithm 3) will be called. Otherwise, *PerformAggregation* (Algorithm 4) will be called.

Given a stem transaction, *OnTransactionAggregated* (Algorithm 3) sends the stem transaction to a randomly chosen peer with a probability of 0.9 or fluffs the transaction by calling *OnTransactionFluff* (Algorithm 5) with a probability of 0.1.

Algorithm 2 OnTransactionStem

```
1: function OnTransactionStem(Transaction tx)
2:     for each Kernel k in tx do                          ▷ at most one Tx in stempool has k
3:         Find Transaction q in stempool that contains k                    ▷ if it exists
4:         // continue to the next iteration if such q does not exist
5:         if tx does not cover q then                ▷ tx covers q if it has all Kernels of q
6:             Drop tx
7:             return                                ▷ error code will be returned to sender
8:         if q covers tx then                              ▷ it means tx and q are the same
9:             if q is still aggregating then                    ▷ should not normally happen
10:                Drop tx
11:                return                                         ▷ with 'accept' error code
12:            else
13:                break
14:        // if tx covers q but q does not cover tx
15:        Validate(tx)                                                 ▷ if not done before
16:        Drop q from stempool
17:    Validate(tx)                                                     ▷ if not done before
18:    // by this point, the given stem-tx is accepted
19:    Add tx to stempool                      ▷ also add dummy inputs to tx if necessary
20:    if NoNeedForAggregation(tx) then        ▷ tx has at least OutputsMax outputs
21:        OnTransactionAggregated(tx)
22:    else
23:        PerformAggregation(tx)
```

Algorithm 3 OnTransactionAggregated

```
1: function OnTransactionAggregated(Transaction tx)
2:     if $RandInt(1,10) \neq 10$ then
3:         Select a random Peer p
4:         Send (stem) tx to p
5:         Set timer (uniformly selected between TimeoutMin and TimeoutMax)
   on tx to later check if it is fluffed or not
6:     else                                                ▷ FluffProbability = 0.1
7:         OnTransactionFluff(tx)
```

PerformAggregation, presented in Algorithm 4, tries to merge a given stem transaction with other transactions in the stempool. In the end, if the number of outputs in the transaction is at least *OutputsMin* (the transaction does not necessarily need more aggregation), *OnTransactionAggregated* (Algorithm 3) will be called to forward the transaction. If the transaction still needs to be aggregated, the function will set a timer (10s) on the transaction to bound the time that it remains in the stempool without being forwarded.

Algorithm 4 PerformAggregation

```
1: function PERFORMAGGREGATION(Transaction $tx$)
2:     for each Transaction $q$ in stempool that needs to be aggregated, starting
       from the one with the closest profitability to $tx$, until !NeedsAggregation($tx$)
       do              ▷ in Beam, Transaction profitability is defined as $\frac{Transaction\ fee}{Transaction\ size}$
3:         TryMerge($tx, q$)                ▷ merges $q$ into $tx$ if the result is valid
4:
5:     if $tx$ has at least $OutputsMin$ outputs then
6:         OnTransactionAggregated($tx$)
7:     else
8:         Set timer ($AggregationTime$) on $tx$    ▷ to later add dummy outputs and
       stem if not aggregated enough by then
```

The function *OnTransactionFluff* (Algorithm 5), after making sure that a given transaction is valid, updates the stempool. Subsequently, the function updates the fluffpool (data structure containing valid fluff transactions) and sends the given transaction to all of its peers except the one that initially sent the fluff transaction.

Finally, we explain *OnTimedOut*, presented in Algorithm 6. If a stem transaction is still waiting for aggregation by the expiration of the timer that was set for it in the *PerformAggregation* function (Algorithm 4), then dummy outputs will be added to the transaction (to ensure that the transaction has at least *OutputsMin* outputs and therefore it is sufficiently difficult to link its inputs and outputs) and *OnTransactionAggregated* (Algorithm 3) will be called to forward the transaction. Moreover, if the fluff version of a forwarded stem transaction is not received by the expiration of its independent random timer, then *OnTransactionFluff* (Algorithm 5) will be called to fluff the transaction.

Algorithm 5 OnTransactionFluff

```
 1: function OnTransactionFluff(Transaction $tx$)
 2:     if $tx$ is in stempool then
 3:         Drop $tx$ from stempool
 4:     else
 5:         for each Kernel $k$ in $tx$ do
 6:             Find Transaction $q$ in stempool that contains $k$        ▷ if it exists
 7:             // continue to the next iteration if such $q$ does not exist
 8:             Drop q from stempool
 9:     if $tx$ is already in fluffpool then        ▷ we already received the fluff tx
10:         Drop $tx$
11:         return                                  ▷ with ’accept’ error code
12:     Validate($tx$)              ▷ BUG - validation should be done earlier!
13:     while fluffpool does not have enough capacity for $tx$ do
14:         Find q, the least profitable Transaction in fluffpool
15:         if $q$ is less profitable than $tx$ then
16:             Drop $q$ from fluffpool
17:         else
18:             Drop $tx$
19:             return                              ▷ with ’accept’ error code
20:     Add $tx$ to fluffpool
21:     Send $tx$ to all Peers of the node          ▷ except the Peer that sent $tx$
```

Algorithm 6 OnTimedOut

```
1: function OnTimedOut(Transaction $tx$)
2:     if $tx$ is still aggregating then
3:         Add dummy outputs to $tx$ so that it has at least $OutputsMin$ outputs
4:         OnTransactionAggregated($tx$)
5:     else                                ▷ fluff timed-out, emergency fluff
6:         OnTransactionFluff($tx$)
```

During the process of studying the source code of Beam, we noticed that in the function *OnTransactionFluff* (Algorithm 5), transactions were validated too late. This would allow an attacker to change the state of the stempool in an honest node by sending invalid transactions. To be more specific, the attacker could send an invalid fluff transaction containing kernels that were already sent to the honest node within valid stem transactions and cause those valid transactions to be removed from the stempool (Line 8 in Algorithm 5). The function *OnTransactionFluff* would check the validity of the malicious transaction only after the accomplishment of the attack (Line 12 in Algorithm 5). We communicated with the Beam developer community about this issue and they modified the source code so that the function *OnTransactionFluff* would validate transactions earlier [2]. As has been previously demonstrated, we believe that independent code review is therefore critical to the security and robustness of blockchain networks.

[2]https://github.com/BeamMW/beam/commit/ade19e1f8b1a702ad81d81092ba6a8f6561513ed

3.2 Main Network Statistics

As of April 2021, with more than 1 million mined blocks and over 86 million coins in circulation [3], Beam has a market capitalization of over 133 million USD [3]. The Beam main network includes multiple bootstrapping nodes in different geographical locations [4]. When a new node joins the Beam network, it usually connects to bootstrapping nodes at the beginning. Hence, it is expected that bootstrapping nodes have more connections than normal nodes.

We have deployed a Beam main network node and a Beam test network node on UBC servers. We have recorded the number of connections for these two nodes every six hours from April 21, 2021, to April 23, 2021. Figure 3.1 shows these numbers. We observe that all the recorded numbers for the main network node are between 11 and 14. For the test network node, all the numbers are between 9 and 11.

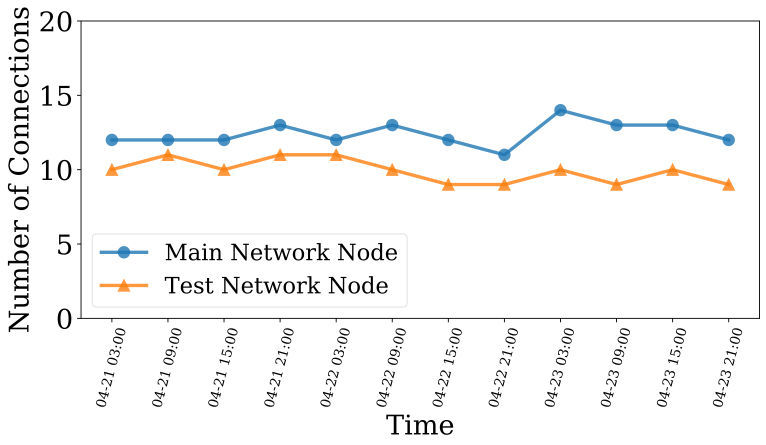


Figure 3.1: The number of connections at different times for a Beam main network node and a Beam test network node deployed on UBC servers.

In addition, we have obtained the number of incoming transactions for bootstrapping nodes located in Europe-Frankfurt (eu-nodes.mainnet.beam.mw), USA-California (us-nodes.mainnet.beam.mw), Hong Kong (ap-hk-nodes.mainnet.beam.mw),

[3]https://explorer.beam.mw/

[4]https://beam.mw/downloads/mainnet-mac

Node	**# of Fluff-Txs per Hour**	**# of 1-Kernel Fluff-Txs per Hour**	**# of Stem-Txs per Hour**	**# of 1-Kernel Stem-Txs per Hour**
eu-node01	139.0	99.0	8.6	6.0
eu-node02	139.2	105.3	9.3	6.0
eu-node03	139.3	107.1	9.1	6.7
eu-node04	139.6	105.5	8.8	6.5
us-node01	139.7	106.3	15.8	13.9
us-node02	139.7	106.9	15.1	13.2
us-node03	140.2	105.5	17.4	15.4
us-node04	139.4	104.6	15.7	13.6
ap-hk-node01	139.3	104.0	9.1	7.1
ap-hk-node02	139.3	105.9	9.3	7.0
ap-hk-node03	139.4	113.4	10.2	7.5
ap-hk-node04	139.8	105.9	9.4	7.0
ap-node01	139.3	107.4	8.1	6.0
ap-node02	139.4	105.8	9.5	6.5
ap-node03	139.3	107.1	9.1	6.7
ap-node04	139.2	106.7	7.8	6.1
ubc-node	113.0	93.1	1.0	0.8

Table 3.2: The number of incoming transactions per hour for different nodes.

and Singapore (ap-nodes.mainnet.beam.mw) from the Beam developer community. We have estimated the number of incoming transactions per hour for each bootstrapping node and also for the UBC main network node using the data of the log files with a start time between April 21, 2021, and April 23, 2021. For each aforementioned node, Table 3.2 shows the number of incoming fluff transactions per hour, the number of incoming single-kernel fluff transactions per hour, the number of incoming stem transactions per hour, and the number of incoming single-kernel stem transactions per hour. We observe that the bootstrapping nodes receive more transactions compared to the normal node deployed on UBC servers. Furthermore, the majority of incoming transactions have only one kernel which means that they are not aggregated.

Chapter 4

Threat Model

The adversary in our model can create nodes and connect to other nodes in the peer-to-peer network. The adversarial nodes can connect to as many nodes as they want. The adversary needs to know the addresses of other nodes before connecting to them. Nevertheless, the adversary cannot impose a connection on any other node if the other node does not want to connect. By increasing the number of adversarial nodes in the network or the number of connections from adversarial nodes to honest nodes, the adversary will be incident on more relay paths and therefore can attack the transaction relay network more effectively.

We assume that all pairs of adversarial nodes are directly connected by instant links through which they can send each other transactions and other messages of any size. Adversarial nodes can store all information that they receive about the transactions and the network. They can analyze the stored information and adjust their decisions. Generally, the adversary is unaware of the exact topology of the network and the connections between pairs of honest nodes. However, the adversarial nodes can obtain partial information about the network through the analysis of their stored data. We assume that the adversary cannot decrypt commitments in transactions to learn their amounts or secret blinding keys. Furthermore, the adversary cannot spend the outputs that are owned by others or trick honest nodes into accepting invalid transactions.

We assume that instead of targeting specific nodes or users, the adversary is interested in mass attacks on the honest portion of the network. Nonetheless, selective attacking could help to hide the position of the adversary in the network. The adversary can recognize the address of a node that forwards a transaction to an adversarial node and acquire partial information about the content of the transaction, such as the number of kernels it has. Also, the adversary can deviate from the relay policy of the network and disregard the relay phase of transactions. Moreover, the adversarial nodes can generate new valid transactions and aggregate them with other valid transactions that they previously received.

Chapter 5

Approach

In this chapter, we describe our approach for each of the three proposed attacks on the implementation of MimbleWimble in Beam. In Section 5.1, we explain the improved transaction source detection. Then, in Section 5.2, we present our approach for delaying transaction relay. Finally, in Section 5.3, we describe transaction denial of service with aggregations. Figure 5.1 illustrates the proposed attacks.

5.1 Improved Transaction Source Detection

In this attack, we use the information leaked by the aggregation in MimbleWimble to improve the first node detection attack. In the original first node detection attack [22], for each incoming transaction, the adversary outputs the first honest node to forward that transaction to an adversarial node as the source of the transaction.

Each time a transaction arrives at a new node in the stem phase, it might get aggregated with other transactions. As a transaction passes more nodes in the stem phase, the probability of it being aggregated with other transactions increases. We surmise that the first node detection attack would lead to an increased precision for transaction source detection by announcing the sender of a transaction as the source if the transaction is not aggregated. Nodes can find whether a transaction is aggregated by counting the number of kernels in it. If a transaction has one kernel, it is not aggregated; otherwise, it is aggregated.

In our attack, a rogue node predicts the origins of single-kernel transactions

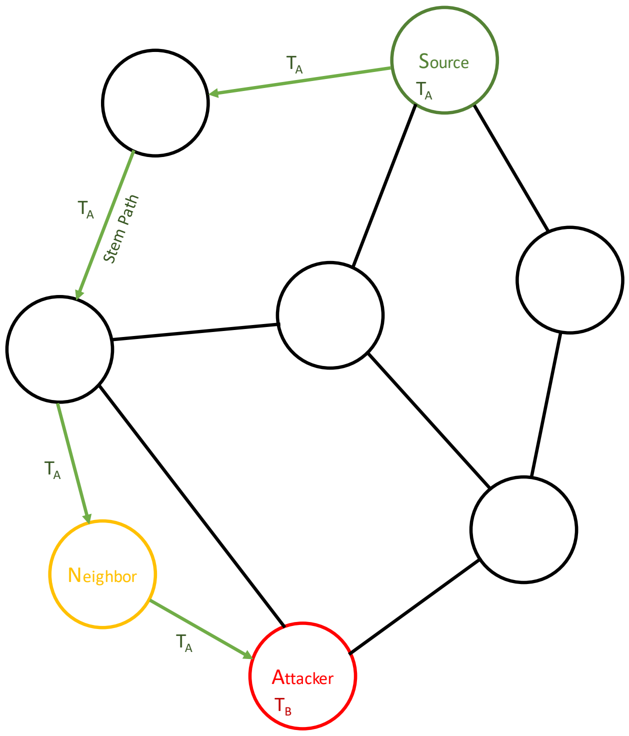


Figure 5.1: The proposed attacks on the transaction relay protocol in Beam. Node *Attacker* is an adversarial node that receives a stem transaction T_A that was originated in node *Source*. The adversary can (1) use the information about the number of kernels in T_A and the address of node *Neighbor*, the sender of the transaction, for improved detection of the transaction source, (2) increase the latency of T_A by adding an excessive delay before forwarding it, or (3) generate a new transaction T_B and fluff both $T_A + T_B$ and T_B to perform a denial of service attack on T_A.

and announces the senders of those transactions as their sources (Figure 5.1). We will then evaluate the precision of this attack to show that our proposed attack has higher precision compared to the normal first node detection attack.

5.2 Delaying Transaction Relay

Privacy-focused transaction relay protocols such as Dandelion++, at some stages of the relay, constrain the number of neighbors that a node will send a transaction.

While such approaches improve the source privacy of transactions, they also make the performance of the protocol dependent on the behavior of the nodes at the bottleneck locations along the relay paths.

Particularly in Dandelion++, each node passes an incoming stem transaction only to one of its neighbors. Although Dandelion++ incorporates a fail-safe mechanism to mitigate black-hole attacks, it is still vulnerable to attacks on transaction latency. An adversarial node in the stem path of a transaction can increase the latency of the transaction by adding an excessive delay before forwarding the transaction or not forwarding the transaction at all. This attack will increase the latency even if another node in the stem path fluffs the transaction itself due to the expiration of its timer.

In our attack, a rogue node will follow the protocol for the relay of a portion of incoming transactions and add a major delay in the relay of some others (Figure 5.1). We will calculate the average time that it takes for the transactions of each group to be broadcasted through the network. We will then compare the results to verify that the attack can considerably delay the transactions.

5.3 Transaction Denial of Service with Aggregations

To improve content privacy, MimbleWimble allows for the aggregation of transactions. However, the adversary can exploit this feature to launch a denial of service attack. Among different aggregations that have a transaction in common, at most one can end up in the blockchain. Therefore, by aggregating different incoming transactions with a newly generated transaction, the adversary can perform a denial of service attack on the incoming transactions.

Let T_A be a new stem transaction received by an adversarial node. To execute the attack, instead of normally aggregating and relaying the stem transaction, the adversarial node generates a new transaction T_B. Then, the adversarial node aggregates the two transactions into $T_A + T_B$ and fluffs both $T_A + T_B$ and T_B. Since T_A is fluffed as a part of an aggregated transaction, the other nodes in the stem path of T_A will not separately fluff T_A. Nevertheless, only one transaction between $T_A + T_B$ and T_B can end up in the blockchain. Hence, if the adversarial node creates T_B in a way that miners prioritize it over $T_A + T_B$ (for this to happen, the profitability for

T_B should be higher than the profitability for $T_A + T_B$), then T_A will not end up in the blockchain. In this case, the wallet that initially created T_A needs to resend T_A to the network. The cost of this denial of service attack for the adversary is the transaction fee of T_B.

In our attack, a rogue node will aggregate the incoming stem transactions with new transactions that have not been mined into any block and fluff the resulting aggregations and the newly generated transactions (Figure 5.1). We will then compare the block mining time for the transactions that were aggregated in this way by the adversary and for the normally relayed transactions. Subsequently, we can validate the feasibility of this attack.

Chapter 6

Implementation

In this chapter, we explain our implementation of the Beam network simulator, the three proposed attacks, and the Beam private test network.

We have provided a simulation of the Beam network to estimate the percentage of stem paths that the adversary will be incident on, the expected number of hops between the source of a stem transaction and the first adversarial node that receives the transaction, and the precision of the first node detection attack on stem transactions. The inputs of this simulation are parameters such as the number of nodes, the percentage of malicious nodes, the expected degree of each node, and the probability of transitioning to the fluff phase in each step of the stem phase. Based on these parameters, the program creates a pseudorandom graph representing the network. The connections of each node are uniformly selected among all other nodes without replacement. The program then tests 1 million pseudorandom stem paths for the estimation. Each tested stem path starts from a source uniformly selected from the set of all nodes and each node in the stem path selects the next node uniformly from the set of its neighbors to forward the stem transaction. We implemented this network simulator in approximately 200 lines of code using the C++ programming language.

To implement each of our proposed attacks, we modified the source code of Beam in the ”testnet” branch [1]. Most of the changes were applied to the files in the ”node” directory and especially the functions described in Section 3.1. We

[1] https://github.com/BeamMW/beam/tree/testnet

modified approximately 600 lines of C++ code in total for the implementation of our three proposed attacks.

To validate our proposed attacks, we have implemented a private test network. The network consists of some normal Beam nodes and some malicious nodes that are the modified versions of normal nodes. The properties of the private network are described as follows:

1. **Number of nodes:** After consulting with the Beam developer community, considering the requirements of our evaluations, and also taking into account the resources available, we decided to have 100 nodes in our private network.

2. **Number of bootstrapping nodes:** There are 10 bootstrapping nodes in our private network, out of a total of 100 nodes. When normal nodes join the network, they first connect to the bootstrapping nodes. Therefore, it is expected that bootstrapping nodes have more connections than normal nodes.

3. **Number of adversarial nodes:** This is a configurable parameter of our private test network. For different attacks, we might need different numbers of adversarial nodes.

4. **Versions of nodes:** For the honest nodes, we use the latest stable version of the "testnet" branch [2]. For the adversarial nodes, we modify this source code to implement each of our proposed attacks.

5. **Number of connections for each node:** We do not change the algorithm for finding new connections and we maintain the policy of nodes in the Beam main network and test network.

6. **Probability of transitioning to the fluff phase:** Similar to the policy of the Beam main network and test network, we set the probability of transitioning to the fluff phase in each step of the stem phase in our private network to 0.1.

7. **Mining:** In our private test network, similar to the Beam main network and test network, a new block is added to the blockchain every minute, on average. The Beam main network uses a Proof of Work (PoW) scheme to grow

[2]https://github.com/BeamMW/beam/commit/cfe091468fbcfcd2092352c22a18099bf9d017f0

the blockchain. Instead, in our private test network, we use a fake mining scheme to avoid wasting our resources on the expensive process of PoW mining. In the fake mining scheme, nodes do not compete with each other over mining new blocks. Fake mining is adequate for the private test network because our proposed attacks focus on transaction relay and do not include byzantine behavior in miners.

8. **Transaction generation rate:** This is a configurable parameter of our private network. For most of our experiments, we want to set the transaction generation rate in a way that the frequency of stem transactions received in the private network nodes would reflect this frequency in the Beam main network nodes. Nevertheless, we also want to be able to vary the transaction generation rate to observe its effect on aggregations.

9. **Number of wallets assigned to each node:** We assign one wallet to each node of the network because we want newly generated transactions to be relayed from each node.

To deploy our private test network, we have used Azure virtual machines (Ubuntu Server 18.04 LTS - Gen 1). We have launched 100 virtual nodes in two geographically separated servers located in the Eastern United States and South East Asia with each server containing 50 virtual nodes. The latency between the virtual nodes in our setup is simulated by adding a pseudorandom delay from a normal distribution with a mean of 100ms and a standard deviation of 20ms to each message using a network emulator called NetEm. The topology of the network is controlled by the peer parameter in the command line interface of the Beam node. The transaction emulation is done using the Beam wallet application programming interface (API) along with a Node.js script.

Chapter 7

Evaluation

In this chapter, we evaluate our proposed attacks on the implementation of MimbleWimble in Beam. In particular, we focus on the following questions:

1. How does using the information about the number of kernels in a transaction improve the localization of the source in the first node detection attack?

2. How does delaying the relay of a transaction increase the total latency of that transaction?

3. How does maliciously aggregating a transaction with other transactions increase the transaction processing time from the user's perspective?

To answer the questions above, we need to determine the proportion of transactions attacked by the adversarial nodes and the impact of each proposed attack on a targeted transaction. We use our network simulator to estimate the proportion of transactions that the adversary can attack, given the percentage of adversarial nodes and other network parameters. To determine the impact of each of our proposed attacks on a targeted transaction, we run a rogue node in our private test network to perform the attack.

In each of our three proposed attacks, the adversary can perform the attack on a transaction if and only if an adversarial node is on the stem path of that transaction. Therefore, to estimate the proportion of transactions that the adversary can attack, we have conducted several experiments with our network simulator to estimate the

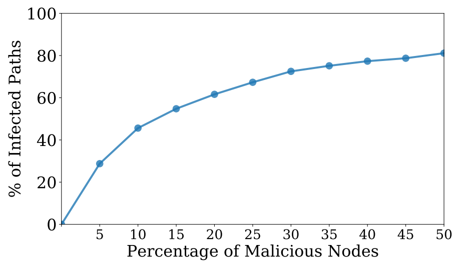
Name	**Value**
Percentage of Malicious Nodes	10%
Probability of Transitioning to the Fluff Phase	0.1
Number of Nodes	1000
Expected Degree of Each Node	10
Number of Bootstrapping Nodes	0

Table 7.1: The default values for the parameters of the network simulator.

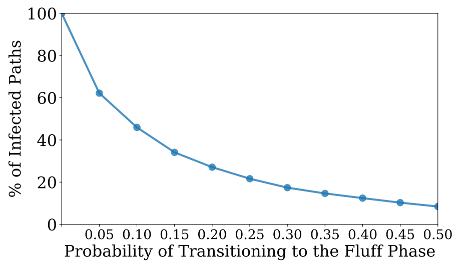
percentage of stem paths that the adversary will be incident on (also referred to as infected stem paths), given the network parameters. Table 7.1 presents the default values that we have used for the parameters of the network simulator. In each experiment, we have varied the value of one network parameter while maintaining the default values for other parameters. Hence, we have observed the effect of each network parameter on the percentage of stem paths that pass through the adversary.

Figure 7.1 shows the results of our experiments with the network simulator. We observe that by increasing the percentage of malicious nodes in the network, the adversary will be incident on more stem paths and therefore can attack more transactions (Figure 7.1a). We also observe that increasing the probability of transitioning to the fluff phase in each step of the stem phase (and consequently decreasing the average length of stem paths) decreases the percentage of stem paths that pass through the adversary (Figure 7.1b). Nevertheless, varying the values of other network parameters, such as the number of nodes (Figure 7.1c), the expected degree of each node (Figure 7.1d), and the number of bootstrapping nodes (Figure 7.1e), does not particularly affect the percentage of stem paths that pass through the adversary.

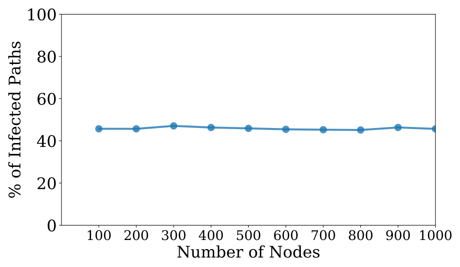
Let us consider the cases where the probability of transitioning to the fluff phase in each step of the stem phase is set to 0.1 (similar to the Beam main network). We note that if only 10% of the nodes are malicious, the adversary will be incident on the stem paths of more than 45% of all transactions in the network and hence can attack those transactions. Increasing the percentage of malicious nodes to 30% increases the percentage of transactions that the adversary can attack to more than 70%.



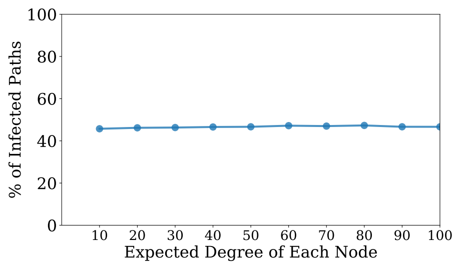
(a)



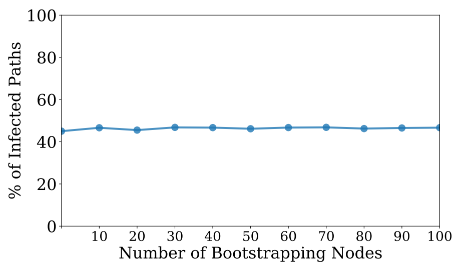
(b)



(c)



(d)



(e)

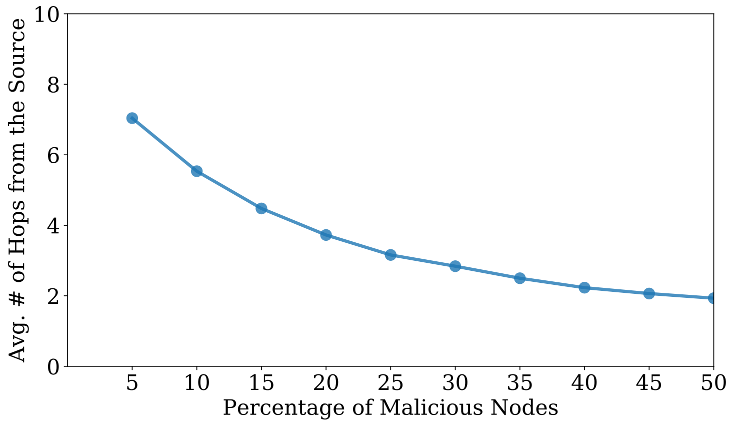
Figure 7.1: The percentage of stem paths that the adversary will be incident on, also referred to as infected paths, with (a) varying percentages of malicious nodes, (b) varying probabilities of transitioning to the fluff phase in each step of the stem phase, (c) varying numbers of nodes, (d) varying expected degrees of nodes, and (e) varying numbers of bootstrapping nodes.

To measure the performance of the first node detection attack, with varying percentages of malicious nodes, we have used our network simulator. In our simulation, the malicious nodes worked collaboratively and the adversary was aware of all transactions received by every malicious node. We tested 1 million stem transactions to measure the performance of the attack. For each stem transaction that was forwarded to an adversarial node, the adversary outputted the first honest node to forward that transaction as the source of the transaction. For the parameters of the network simulator other than the percentage of malicious nodes, we have used the default values presented in Table 7.1.

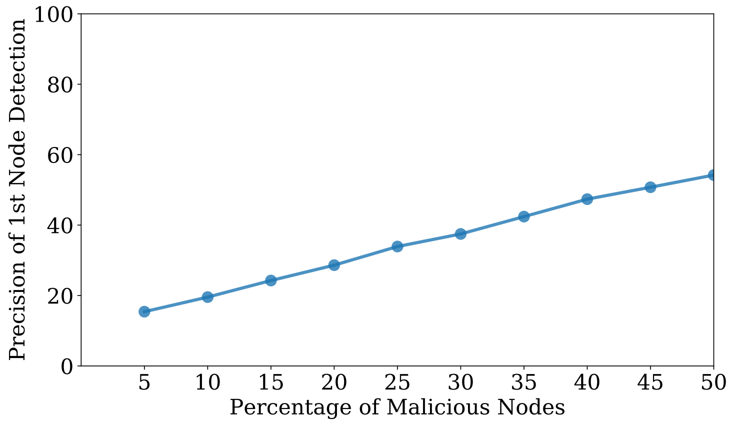
Figure 7.2 shows the results of running the first node detection attack in our network simulator. We observe that increasing the percentage of malicious nodes in the network decreases the expected number of hops between the source of a stem transaction and the first adversarial node that receives the transaction (Figure 7.2a). Moreover, increasing the percentage of malicious nodes increases the precision of the first node detection attack (Figure 7.2b). We note that if 10% of the network nodes are malicious, the precision of the first node detection attack will be less than 20%. Therefore, to make the first node detection attack more effective, we need to increase the percentage of malicious nodes.

To validate our conjecture that the adversary can increase the precision of the first node detection attack by only predicting the sources of single-kernel transactions, we have run a rogue node in our private test network. Our rogue node performed the first node detection attack on 160 single-kernel stem transactions and 40 aggregated stem transactions. For each examined stem transaction, we recorded the number of hops between the source of that transaction and our rogue node and whether the sender of the transaction was its actual source.

Figure 7.3 shows the numbers of intervening hops in stem paths separately for the two aforementioned groups of transactions. We observe that the average number of hops between the source of a single-kernel transaction and our rogue node (6.15) is less than the average number of hops between the source of an aggregated transaction and our rogue node (9.35). The precision of the first node detection attack is 32% for the single-kernel transactions and 12% for the aggregated ones. Hence, performing the first node detection attack only on single-kernel stem transactions would lead to an improved transaction source detection.



(a)



(b)

Figure 7.2: (a) The expected number of hops between the source of a stem transaction and the first adversarial node that receives the transaction with varying percentages of malicious nodes; (b) the precision of the first node detection attack on stem transactions with varying percentages of malicious nodes.

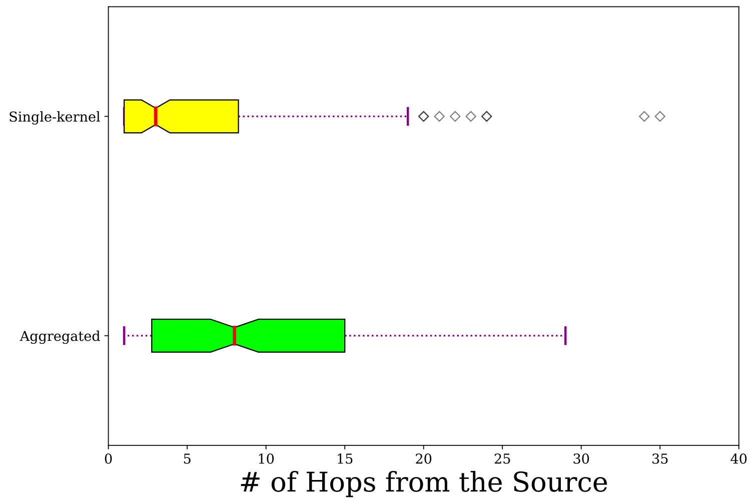


Figure 7.3: Comparison of the single-kernel transactions and aggregated transactions in the number of hops between their sources and our rogue node.

To measure the additional transaction latency caused by our second proposed attack, delaying transaction relay, we have run a rogue node in our private test network. Our rogue node followed the protocol for the relay of 300 examined stem transactions and delayed the relay of 300 other stem transactions. Some node in the stem path of each delayed transaction fluffed the transaction after a while (the fail-safe mechanism). We have measured the latency of each transaction by calculating the difference between the time that our rogue node received that transaction in the stem phase and the time that the transaction was recorded in the blockchain (obtained from the block timestamp).

Figure 7.4 shows the latency for the normally relayed transactions and excessively delayed ones. We observe that the delayed transactions are expected to have more latency compared to the normally relayed ones. Indeed, the average latency for the delayed transactions is 53s. For the normally relayed transactions, however, the average latency is 31s. That means the expected latency for an attacked transaction is 71% higher than the expected latency for a normally relayed transaction. We have previously observed through simulation that if 10% of the nodes are malicious, the adversary can attack more than 45% of all transactions in the network. Consequently, by controlling 10% of the nodes in the network, the adversary can increase the expected transaction latency in the network by more than 31%.

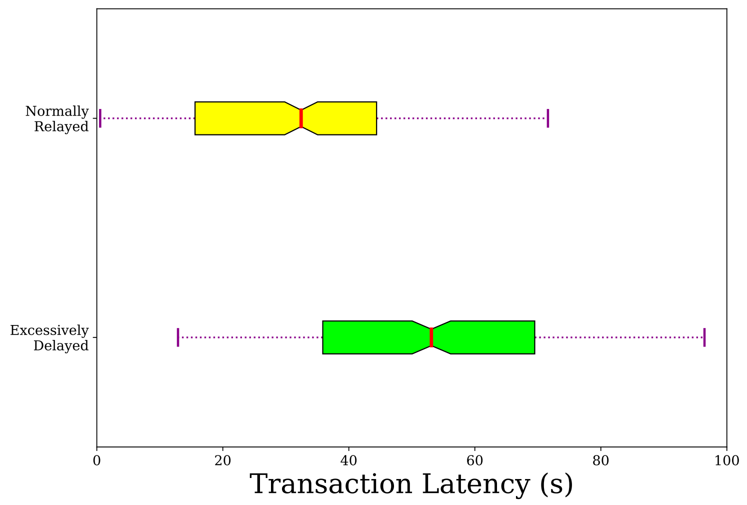


Figure 7.4: Comparison of the normally relayed transactions and excessively delayed transactions (through our second proposed attack, delaying transaction relay) in their latency.

To measure the impact of our third proposed attack, transaction denial of service with aggregations, we have run a rogue node in our private test network. Our rogue node performed the attack on 300 incoming stem transactions. For each attacked transaction T_A, our rogue node generated a new transaction T_B with higher profitability compared to T_A and fluffed $T_A + T_B$ and T_B. For each attacked transaction T_A and its corresponding adversarial transaction T_B, we observed whether $T_A + T_B$ or T_B ended up in the blockchain.

Our rogue node successfully prevented 100% of the attacked transactions from ending up in the blockchain. In fact, for each attacked transaction T_A and its corresponding adversarial transaction T_B, $T_A + T_B$ had lower profitability compared to T_B and hence T_B ended up in the blockchain. Therefore, if 10% of the nodes are malicious, the adversary can attack more than 45% of all transactions and prevent them from ending up in the blockchain.

Figure 7.5 shows the latency for the transactions that the adversary generated to perform the denial of service attack. We have measured the latency of each adversarial transaction by determining the difference between the time that our rogue node generated that transaction and the time that the transaction was recorded in the blockchain. The average latency for the adversarial transactions in this attack is 29s which is slightly lower than the average latency for normally relayed transactions (31s). That is because the adversary immediately fluffs the transactions generated to perform the denial of service attack.

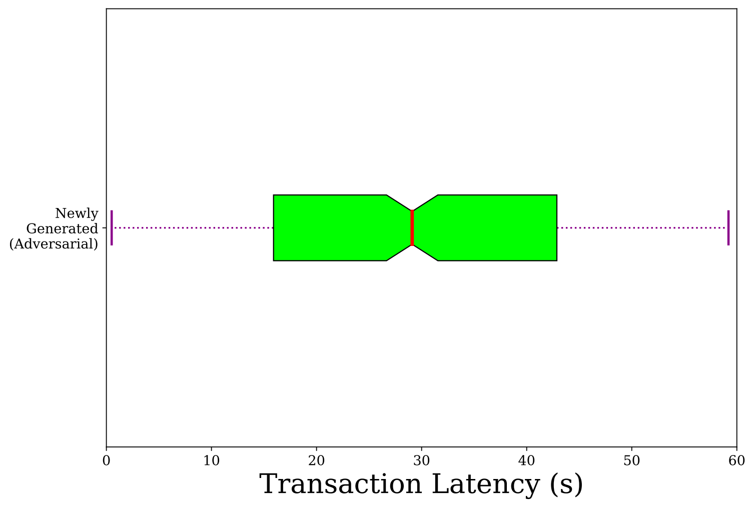


Figure 7.5: The latency for the transactions that the adversary generated to perform the denial of service attack with aggregations.

Chapter 8

Discussion

8.1 Improved Transaction Source Detection

Our evaluation has demonstrated that by increasing the number of adversarial nodes in the network, not only will the adversary be able to predict the source for more transactions, but the adversary also can perform the first node detection attack with better precision. Since the precision of the first node detection attack will be less than 20% if 10% of the network nodes are malicious, we need to increase the percentage of malicious nodes to make the attack more effective. Moreover, the adversary can increase the precision of the first node detection attack by performing the attack merely on single-kernel transactions. That is because the precision of the first node detection attack is 32% for the single-kernel transactions while only 12% for the aggregated ones. The adversarial nodes performing this attack cannot be detected by other nodes in the network because they seem identical to the normal nodes in terms of their behavior perceived by the rest of the network.

The mitigation to this attack may lie within a better understanding of the potential relationship between source privacy and content privacy. In this attack, the rogue node is in fact compromising the source privacy of the incoming transactions based on the information exploited from the means of providing content privacy, particularly aggregations. To the best of our knowledge, the formalization of source and content privacy has not been explored in the literature. Hence, an interesting direction for future research is to formalize these two types of privacy to better understand their relationship and to develop general-purpose mitigation strategies.

8.2 Delaying Transaction Relay

If 10% of the network nodes are adversarial, by delaying transaction relay, the adversary can increase the expected transaction latency by more than 31%. This attack is easy to implement and does not require the adversary to create any new transaction or pay any transaction fee. Furthermore, this attack is not specific to the implementations of MimbleWimble, and other blockchain systems that use Dandelion++ for transaction relay could also be susceptible to this attack. At the moment, Beam does not incorporate any mechanism to detect the nodes that are performing delay or black-hole attacks.

The mitigation to this attack can be investigated in approaches that have been used in some routing protocols for ad hoc networks, such as Castor [17]. In these protocols, each node keeps an estimate of reliability for each of its neighbors and makes routing decisions based on those estimates. We conjecture that there is value in adopting a similar idea to blockchain protocols concerned with source privacy. One design is for each node in the blockchain network to maintain an internal reliability score for each of its neighbors. The scores would be updated based on the feedback that nodes receive regarding the propagation of the transactions that they relayed to their neighbors. Using reliability scores, nodes can improve their relaying decisions. Scoring schemes have also been incorporated in other blockchain networks, such as Filecoin and Ethereum 2.0 [39].

Before integrating the aforementioned scheme into MimbleWimble blockchain networks, the effects of the scheme on the network topology should be analyzed. Taken to an extreme, the scoring scheme could result in further centralization of the network (nodes with higher reliability would receive more transactions), which is not a desirable feature for blockchain networks.

8.3 Transaction Denial of Service with Aggregations

If 10% of the network nodes are adversarial, by maliciously aggregating incoming stem transactions with newly generated transactions, the adversary can prevent more than 45% of all transactions from ending up in the blockchain. The cost of this attack for the adversary is the transaction fees of newly generated transactions. The adversary can reduce the cost for a newly generated transaction by reducing its size while choosing a sufficient transaction fee so that the profitability (defined as $\frac{Transaction\ fee}{Transaction\ size}$) of the new transaction is higher than the profitability of the incoming stem transaction that it gets aggregated with. Moreover, if the adversary modifies each of its nodes to aggregate multiple incoming stem transactions that arrive within a short period of time with one new transaction, then the adversary can reduce the number of newly generated transactions and hence the total cost of the attack.

We note that this attack can be performed only on stem transactions and it does not work on fluff transactions. Let us consider the case that an adversarial node aggregates an incoming fluff transaction T_A with a newly generated transaction T_B and fluffs both $T_A + T_B$ and T_B. The fact that T_A was sent to the adversarial node in the fluff phase means that some honest node(s) had T_A as a fluff transaction. Since honest nodes follow the protocol and relay the incoming fluff transactions without aggregating them, T_A will be broadcasted through the network. Hence, T_A can still end up in the blockchain, even if miners prioritize T_B over $T_A + T_B$.

The mitigation to this attack can be achieved by modifying the wallet's source code to periodically resend the previously broadcasted transactions that have not ended up in the blockchain. Increasing the number of retries for a transaction exponentially decreases the probability that the transaction does not appear in the blockchain. Nonetheless, even if we modify the wallet's source code as mentioned earlier, the proposed attack can still significantly increase the latency of transactions. It should also be noted that resending a transaction too frequently could cause that transaction to appear in multiple aggregations and therefore prevent some other transactions from ending up in a block.

Similar to the proposed mitigation scheme for the delay attack, we can modify the network nodes to maintain internal reliability scores and make routing decisions based on those scores. Using this scheme, adversarial nodes lose some reliability scores when they prevent transactions from ending up in the blockchain by maliciously aggregating them with other transactions. However, as previously mentioned, this scheme might lead to further centralization of the network.

We can also modify the protocol and disallow aggregation of transactions during the relay phase. Therefore, we allow aggregation only when transactions are being added to the blockchain. This approach mitigates the proposed denial of service attack but compromises the content privacy of transactions as network nodes can observe transactions before they get aggregated with other transactions during the relay phase.

Chapter 9

Conclusion

Bitcoin has been shown to be susceptible to attacks against the source and content privacy of transactions. These susceptibilities have been studied in the literature and several privacy-focused protocols have been proposed. Dandelion++ is a transaction relay protocol that has been used as a solution to improve the source privacy of transactions. Also, MimbleWimble has been proposed as a cryptocurrency protocol that would offer enhanced content privacy. The prominent implementations of MimbleWimble, such as Beam, have adopted Dandelion++ for transaction relay.

In this thesis, we proposed three attacks on the implementation of MimbleWimble in Beam: (1) improved transaction source detection, (2) delaying transaction relay, and (3) transaction denial of service with aggregations. We implemented these attacks in a private test network of 100 Beam nodes. We observed that the precision of the first node detection attack is 32% for the single-kernel transactions while only 12% for the aggregated ones. Therefore, performing the first node detection attack only on single-kernel stem transactions would lead to an improved transaction source detection. We demonstrated that if 10% of the network nodes are adversarial, by delaying transaction relay, the adversary can increase the expected transaction latency by more than 31%. Also, if 10% of the network nodes are adversarial, by performing transaction denial of service with aggregations, the adversary can prevent more than 45% of all transactions from ending up in the blockchain. Furthermore, we discussed potential mitigations to the proposed attacks that could be used to improve transaction relay in MimbleWimble-based protocols.

Bibliography

[1] Beam developers, Beam. https://beam.mw/. Accessed: 2021-04-23. → pages 2, 9

[2] Bitcoin core developers, Bitcoin core. https://bitcoincore.org/. Accessed: 2021-04-23. → pages 2, 6

[3] Coinmarketcap. https://coinmarketcap.com/historical/20210418/. Accessed: 2021-04-23. → pages 1, 17

[4] Grin developers, Grin. https://grin-tech.org/. Accessed: 2021-04-23. → pages 2, 9

[5] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 34–51. Springer, 2013. → pages 1

[6] G. Betarte, M. Cristiá, C. Luna, A. Silveira, and D. Zanarini. Towards a formally verified implementation of the mimblewimble cryptocurrency protocol. In *International Conference on Applied Cryptography and Network Security*, pages 3–23. Springer, 2020. → pages 9

[7] G. Birmpas, E. Koutsoupias, P. Lazos, and F. J. Marmolejo-Cossío. Fairness and efficiency in dag-based cryptocurrencies. In *International Conference on Financial Cryptography and Data Security*, pages 79–96. Springer, 2020. → pages 1

[8] A. Biryukov, D. Khovratovich, and I. Pustogarov. Deanonymisation of clients in bitcoin p2p network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 15–29, 2014. → pages 2

[9] S. Bojja Venkatakrishnan, G. Fanti, and P. Viswanath. Dandelion: Redesigning the bitcoin network for anonymity. *Proceedings of the ACM on*

Measurement and Analysis of Computing Systems, 1(1):1–34, 2017. → pages 2, 6

[10] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018. → pages 9

[11] M. Carlsten, H. Kalodner, S. M. Weinberg, and A. Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 154–167, 2016. → pages 1

[12] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002. → pages 1

[13] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014. → pages 1

[14] G. Fanti and P. Viswanath. Anonymity properties of the bitcoin p2p network. *arXiv preprint arXiv:1703.08761*, 2017. → pages 2

[15] G. Fanti, S. B. Venkatakrishnan, S. Bakshi, B. Denby, S. Bhargava, A. Miller, and P. Viswanath. Dandelion++ lightweight cryptocurrency networking with formal anonymity guarantees. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):1–35, 2018. → pages 2, 6

[16] G. Fuchsbauer, M. Orrù, and Y. Seurin. Aggregate cash systems: A cryptographic investigation of mimblewimble. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 657–689. Springer, 2019. → pages 9

[17] W. Galuba, P. Papadimitratos, M. Poturalski, K. Aberer, Z. Despotovic, and W. Kellerer. Castor: Scalable secure routing for ad hoc networks. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010. → pages 37

[18] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017. → pages 1

[19] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 2016. → pages 1, 6

[20] A. Kiayias, E. Koutsoupias, M. Kyropoulou, and Y. Tselekounis. Blockchain mining games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 365–382, 2016. → pages 1

[21] S. King and S. Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 2012. → pages 1

[22] P. Koshy, D. Koshy, and P. McDaniel. An analysis of anonymity in bitcoin using p2p network traffic. In *International Conference on Financial Cryptography and Data Security*, pages 469–485. Springer, 2014. → pages 3, 21

[23] G. Maxwell. Coinjoin: Bitcoin privacy for the real world. In *Post on Bitcoin forum*, 2013. → pages 2

[24] G. Maxwell. Confidential transactions. 2016. → pages 2

[25] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140, 2013. → pages 2, 6

[26] T. Mitani and A. Otsuka. Anonymous probabilistic payment in payment hub. 2020. → pages 2

[27] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. → pages 1, 5

[28] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh. Erlay: Efficient transaction relay for bitcoin. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 817–831, 2019. → pages 6

[29] S. Noether. Ring signature confidential transactions for Monero. Cryptology ePrint Archive, Report 2015/1098, 2015. https://eprint.iacr.org/2015/1098. → pages 1, 6

[30] M. Ober, S. Katzenbeisser, and K. Hamacher. Structure and anonymity of the bitcoin transaction graph. *Future internet*, 5(2):237–250, 2013. → pages 6

[31] L. L. Peterson and B. S. Davie. *Computer networks: a systems approach*. Elsevier, 2007. → pages 2

[32] A. Poelstra. Mimblewimble. 2016. → pages 1, 2, 6

[33] E. Rohrer and F. Tschorsch. Counting down thunder: Timing attacks on privacy in payment channel networks. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 214–227, 2020. → pages 2

[34] D. Ron and A. Shamir. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*, pages 6–24. Springer, 2013. → pages 6

[35] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 515–532. Springer, 2016. → pages 1

[36] A. Silveira, G. Betarte, M. Cristiá, and C. Luna. A formal analysis of the mimblewimble cryptocurrency protocol. *arXiv preprint arXiv:2104.00822*, 2021. → pages 9

[37] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015. → pages 1

[38] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. Spectre: A fast and scalable cryptocurrency protocol. 2016. → pages 1

[39] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras. Gossipsub: Attack-resilient message propagation in the filecoin and eth2. 0 networks. *arXiv preprint arXiv:2007.02754*, 2020. → pages 37

[40] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. 2014. → pages 1