Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networked Systems
ETH Zürich — seit 2015

De-anonymizing Users of Cryptocurrencies

Semester Thesis

SA-2020-07

Author: Cedric Maire

Tutor: Maria Apostolaki

Supervisor: Prof. Dr. Laurent Vanbever

February 2020 to May 2020

Acknowledgments

I wish to express my sincere gratitude and appreciation to Maria Apostolaki, my tutor, and Prof. Dr. Laurent Vanbever, my supervisor, who granted me the opportunity to work on this interesting project related to the topic of de-anonymization of cryptocurrency users as part of Marias Ph.D. thesis at the NSG laboratory at ETHZ. I would like to thank Maria for all the valuable help that she provided whenever I needed it and for the knowledge that I gained from her over the course of the semester. Finally, I wish to thank my parents and friends for helping me through the difficult task of finalizing my semester thesis within a limited time frame.

REQUIESCAT IN PACE

01001111 11101110 10011001 00001100 01000010 11000110 11110001 01101100
10111110 11000111 10110111 00111011 11101001 11011110 01011001 11001001
11001011 01000001 11011100 11100100 11010010 11101110 10001110 01010000
01100100 10110011 01000101 10110100 01001110 10010110 11111110 10111111

Abstract

In 2009, Satoshi Nakamoto introduced the first digital currency to solve the double-spending problem without a third party with the original Bitcoin whitepaper [9]. Cryptocurrencies, particularly Bitcoin, are increasingly utilized as adoption grows and sometimes wrongly mentioned to be anonymous. Bitcoin and Ethereum, the two largest cryptocurrencies, are pseudonymous, as they rely mainly on pseudonyms to dissimulate real identities. The usage of pseudonyms protects real identities as long as no link is evident between a given pseudonym and a particular identity. Pseudonymity can be quantified in terms of the probability of an attacker's success in linking specific transactions and thus public keys (pseudonyms) to source `IP` addresses (identities). This semester thesis proposes a framework for automatically conducting various types of experiments given attacker power parameters and behaviors. The framework records the state of the network as seen by the victim and its direct neighbors via logging; this data can be analyzed in future research.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Tasks and Goals . . . 2
- 1.3 Overview . . . 3

2 Background and Related Work **4**
- 2.1 Background . . . 4
- 2.2 Related Work . . . 8

3 Design **9**
- 3.1 Raw Packets and Logging . . . 9
- 3.2 Framework . . . 10
  - 3.2.1 Scripting . . . 12

4 Evaluation **14**
- 4.1 Bitcoin . . . 14
- 4.2 Ethereum . . . 16

5 Outlook **17**
- 5.1 Future Work . . . 18

6 Summary **20**

References **21**

A Appendix **I**
- A.1 Score Features . . . I
- A.2 Tutorial . . . I
  - A.2.1 Bitcoin Core . . . II
  - A.2.2 Go Ethereum . . . III

Chapter 1

Introduction

With approximately $164 billion and $23 billion market capitalization[1], respectively, Bitcoin and Ethereum are the two most commonly used cryptocurrencies to date. Bitcoin is the original first implementation of a blockchain and solved the double-spending problem without the help of a third party by combining a replicated *blockchain*, a *consensus* algorithm, and the so-called *proof-of-work*. The classical blockchain is composed of cryptographically backward-linked blocks containing transaction data. Bitcoin's main purpose is to serve as a digital currency that is equivalent to paper money in the real world. Ethereum's main objective, on the other hand, is not to function as a digital currency but rather to run decentralized applications[2]. These applications are enabled through so-called *smart contracts*: protocols or small pieces of software running on top of the Ethereum network and performing backtrackable and irreversible transactions without the help of a third party. Executions of applications are also transactions and are no different from transactions sending Ether from one account to another; they can thus be de-anonymized in the same way.
As previously stated, both of these cryptocurrencies are pseudonymous rather than anonymous. The creation of a Bitcoin wallet or Ethereum account provides the user with a *public address*, which is a hashed version of a *public key*. Transactions are signed with *secret keys*, verified with the corresponding *public keys*, and are assigned to *public addresses*. This difference is crucial in the sense that a user is hidden behind its pseudonym only as long as no link between his pseudonym and his real identity exists. The argument can be made for Bitcoin, at least, that a new public address is created and used after each transaction. As shown in [1], such transactions originating from different public addresses can be linked together. Consequently, if one transaction in the set is de-anonymized, the whole set is de-anonymized as well.
Some other cryptocurrencies take a different approach and include certain privacy guarantees at the protocol or cryptographic level. One example is Zcash[3], a fork of the *Bitcoin Core* codebase. Zcash introduces zero-knowledge proofs, allowing transactions to be verified without revealing the sender, receiver, or transaction amount. A second example is Monero[4], which makes use of ring signatures, ring confidential addresses, and stealth addresses to hide, like Zcash, the sender, receiver, and transaction amount. Transaction clustering and linking to `IP` addresses has been demonstrated in [1] to be feasible with Zcash and Monero as well, even for low budget adversaries.
Our chosen model includes a single victim node, presumed to be the node running on the victim's computer, and a single attacker node, running on the attacker's computer. The attacker is presumed to be at the network level and *on-path*, thus sees a fraction of the victim's connections. The attacker

[1] `https://coinmarketcap.com/`, (May 2020)
[2] Nevertheless, it can still be used as a digital currency.
[3] `https://z.cash/`
[4] `https://web.getmonero.org/`

could be, for example, its *ISP* or a powerful entity, such as a government, running routers used by the victim. Such an attacker has power over the seen connections and may drop, inject, corrupt, or delay packets at its leisure. In this model, the attacker node follows as closely as possible the victim's connections to other peers in the network. With Ethereum, since communications with peers are encrypted, the network view of the victim's direct neighbors at any given moment is closest to the victim's real view. Since communication between Bitcoin peers is not encrypted, a network attacker seeing the victim's connections can eavesdrop, read the raw packets, and reconstruct the victim's network view.

1.1 Motivation

Bitcoin mainly and Ethereum secondarily are used as digital money; hence, it makes perfect sense to discuss financial privacy. Financial privacy is a necessity that is not limited to such activities as buying drugs on illegal marketplaces. If cryptocurrencies are to be used for everyday purchases, basic privacy guarantees are required. Transactions must withhold the transaction amounts for third parties and the total amount behind the origin and destination public addresses from everyone but the owner of these addresses for obvious reasons. In addition, the coins themselves must be fungible, meaning that coins are indistinguishable from one another. Such basic privacy guarantees prohibit transaction censorship or pseudonym profiling and analysis. With de-anonymization, in case these guarantees do not hold, a user's financial information is released to the public and can thus be analyzed and exploited. Preventing de-anonymization is an initial step to enhanced financial privacy. In order to prevent such attacks, exploits must first be found. This framework allows data to be easily collected and various types of exploits to be tested.
The primary motivation behind this project is to build a practical framework that is able to automatically simulate different kinds of attacks performed against a selected victim node. The framework enables the transition from raw wire packet dumping to simple logging for Bitcoin. As discussed below, the reconstruction of communication from raw packets is possible, and thus equivalent to logging, but slow. For both Bitcoin and Ethereum, the framework provides a victim node whose protocol behavior is unchanged compared with the original implementation, and non-critical local behaviors such as logging are introduced to record the view of the network. It also provides an attacker node whose behavior is adjusted in such a way to allow automatic following of the victim's peers, local logging of the network view, and altered critical behaviors, such as no advertising and broadcasting of new transactions. By removing the transaction advertisements and broadcasting on the attacker node, a single experiment recording can be used and the fraction of seen connections can be modified without having to record a new experiment by stripping off peers. The third node provided by the framework is a node that simulates a random node in the Bitcoin or Ethereum network and can be used as reference. In addition, the framework includes scripts to apply connection deterioration (packet dropping or delay) for a defined fraction of the victim's connections seen by the attacker, recording of the nodes logging to save the network views at any given moment, and the execution of transactions originating from the victim node. After defining the attacker's total power (fraction of connections seen), the framework automatically executes the tasks previously mentioned.

1.2 Tasks and Goals

The final objective of transaction de-anonymization in the case of pseudonymous cryptocurrencies is to link pseudonyms to real identities. In our model, we consider origin public addresses of trans-

actions as pseudonyms and `IP` addresses of running nodes as identities. The objective of this thesis is to deliver a framework as defined in 1.1. This framework enables data collection, the first critical step of data analysis. Data collection is crucial in the sense that data analysis techniques are only as good as the underlying data.
This thesis is divided into two distinct parts. First, raw Bitcoin communication packets are reconstructed in order to demonstrate that the network view and communication with neighbor peers of the victim node can be extracted and is equivalent to simple logging at the victim node. Understanding whether or not raw packet reconstruction to Bitcoin message streams is equivalent to simple logging at the victim node is useful, as a long and cumbersome process could be replaced by a far simpler and more elegant solution. In a second time, the actual framework, which consists of the two sub-projects Bitcoin and Ethereum, is developed. For each of these two cryptocurrencies, the following tasks have been accomplished:

- Analysis of the main implementations of the protocols (*Bitcoin Core* and *Go Ethereum*) and understanding the relevant sections of the source code, including peer selections, transactions broadcasting, and internal logging
- Modification of these implementations to introduce extra logging and modified behaviors, such as peer sharing and no new transactions announcements and broadcasting
- Scripting behaviors such that the attacker follows the victim's neighbors and both nodes keep a tight set of common peers
- Assembling all tasks into a single automated framework
- Conducting various kinds of experiments, including the creation of transactions on the victim node
- Simple proof-of-concept analysis of the logging to suggest the recorded data's usefulness

1.3 Overview

The rest of the report is structured as follows. In chapter 2, fundamental concepts are defined, and important tools used throughout the semester thesis and background knowledge required to understand the design choices are introduced. In chapter 3, the design choices for the framework and how the framework works are presented. Chapter 4 analyzes data recorded through the framework and briefly discusses its usefulness. Chapter 5 concludes the report by presenting possible extensions and future work that may serve as the foundation of a following thesis. Finally, chapter 6 summarizes the work, report, and framework. Additionally, the appendix A offers further explanation of experiments as well as a hands-on tutorial for the framework, from installation to recording.

Chapter 2

Background and Related Work

2.1 Background

Let us begin by defining some important terms, including victim, attacker, and neighbor:

Victim
: This is the entity from whom the attacker attempts to de-anonymize the transactions, i.e., predict which transactions originated from this node. The victim is running his or her own Bitcoin or Ethereum node; the node's parameters or protocol behavior cannot be directly changed or influenced by the attacker. The victim is assumed to execute a transaction at some point in time and the task is to determine this transaction.

Attacker
: One or multiple entities running Bitcoin or Ethereum nodes and having access to `IP` addresses as needed. We have full control of these nodes during the complete process of an attack. We can modify the protocols' behavior at leisure and the attacker has the ability to act at the network level, thus it is capable of seeing a fraction of the victim's connections. The attacker can act on these connections at leisure and drop, duplicate, corrupt, or delay packets.

Neighbor
: Neighbor peers include all of the other nodes taking part in the Bitcoin or Ethereum networks to which the victim or attacker connects directly. Similarly to those of the victim, these node's parameters or protocol behavior cannot directly be changed or influenced by the attacker.

We will now present and describe important tools used throughout the thesis and key observations that help understand our attack model and the choice of design. The first task was to investigate the possibility of reconstructing a Bitcoin protocol stream of messages from a recording of raw wire packets. The following tools and observation have extensively been used:

TShark
: Command line tool able to dump and analyze network traffic from an interface. It is used to dump the raw wire packets coming from a victim Bitcoin node executing a transaction halfway through a recording session. The output format is a `PCAP` file containing all the captured packets, including headers and payloads. The tool can be found at: `https://www.wireshark.org/docs/man-pages/tshark.html`.

gopacket
: `Go` library maintained by Google providing packet processing capabilities. This library has

proved useful and efficient in reordering and dropping duplicate packets as well as extracting a continuous stream of Bitcoin messages payloads. This library is hosted on GitHub at: `https://github.com/google/gopacket`.

Messages

The extracted stream of payloads is a continuous stream of raw Bitcoin messages following a pre-defined structure. Each message is composed of a header and a payload, which depends on the type of message. All raw message structures can be found at: `https://en.bitcoin.it/wiki/Protocol_documentation`. The header always contains the following fields and structure:

magic
Value of type `uint32_t` defining the origin network of the message, such as the main or test network. This value can be used to look for the next message if the stream state is unknown.

command
Value of type `char[12]` indicating the type of message contained in the payload that follows the header. This value is `NULL` padded to be constant in length.

length
Value of type `uint32_t` indicating the total byte length of the payload.

checksum
Value of type `uint32_t` composed of the first 4 bytes of the double `SHA-256` hash of the payload, i.e., `sha256(sha256(payload))`. This value verifies that the payload contains no errors before decoding it into a message.

Before discussing the message types of interest, let us define two important formats:

Variable Length Integer
Defines the type `var_int`. The encoding is variable in length to save space. Values less than `0xFD` are of type `uint8_t`. Values less than or equal to `0xFFFF` are represented by the value `0xFD` followed by an `uint16_t`. Values less than or equal to `0xFFFF FFFF` are represented by the value `0xFE` followed by an `uint32_t`. Finally, greater values are represented by the value `0xFF` followed by an `uint64_t`. The actual integer is encoded in little endian.

Inventory Vectors
Defines the type `inv_vect`. Composed of two fields: **type** of type `uint32_t` and **hash** of type `char[32]`. The **type** field informs us of the object type referenced by **hash**.

Three message types are of interest in terms of reconstructing the network view of a node[1]:

inv Advertisement message for new transactions and blocks. Composed of two fields: **count** of type `var_int` defining the number of objects advertised and **inventory** of type `inv_vect[]`, an array of `inv_vect` of length **count**.

getdata
Message requesting the objects referenced in the inventory vector. Same composition as the **inv** message.

[1] We are only interested in the view of known transactions.

tx Message containing the actual transaction data. The raw structure is more complex and not detailed out, since a single field is of interest: **hash** of type `char[32]` - the hash of the broadcast transaction.

Based on the protocol and implementation details, it is apparent that each time a node learns about a new transaction it sends out an **inv** message to each of its neighbors to advertise this new transaction. In the event that the destination node does not know about this particular transaction, it sends back a **getdata** message and the first node sends out a **tx** message to broadcast this transaction to the requesting node. This succession of messages leaks information about when a node learns about a new transaction. For example, when node A sends out an **inv** message to node B we know that A recently learned[2] about the referenced transaction and that A thinks[3] that node B does not yet know about it. If node B sends a **getdata** message to node A, we can infer that A sent an **inv** message to B and that B did not know about this transaction, at that time. If node A sends a **tx** message to B, we can infer that B sent a **getdata** message to A and just learned about this transaction.

The second task that had to be accomplished was to build the simulation framework. In this regard, let us first discuss the two main Bitcoin and Ethereum protocol implementations we focused on:

Bitcoin Core
Main `C++` implementation of the Bitcoin protocol. The framework builds on top of version `0.19.1`, the most recent stable version at the time of this thesis. This implementation has been chosen because it is considered to be the reference implementation and is used by the majority of nodes. A modification has been made in the configuration to ease up the attack simulation: the victim, attacker, and reference nodes do not listen for incoming connections nor do they discover new nodes, but instead only connect to a pre-defined set of peers.

Go Ethereum
Main and official `Go` implementation (*Geth*) of the Ethereum protocol (alongside Aleth a `C++` and Trinity a `Python` implementation). The framework builds on top of version `1.9.12`, the most recent stable version at the time of this thesis. This implementation has been chosen because it is considered to be one of three reference implementations and is used by the majority of nodes. A modification has been made in the configuration to ease up the attack simulation: the victim, attacker and reference nodes do not listen for incoming connections. The victim and reference nodes build outgoing connections to newly discovered nodes and the attacker receives commands from the victim to determine which peers it should connect to.

There is a fundamental difference in how communications occur within the Bitcoin and Ethereum networks. All Bitcoin communications are done in plaintext and thus can be read by an on-path eavesdropper. As we will see, this implies that the eavesdropper can, by reading directly on wire, reconstruct the view of the victim by analyzing the exchanged messages. In contrast to Bitcoin, all Ethereum communications are encrypted[4]. An on-path eavesdropper cannot read the exchanged messages. Therefore, there is no way to directly infer the victim's view of the network. To obtain

[2] We will see that the *Bitcoin Core* implementation introduced what they call *diffusion* of transaction advertising to increase the privacy of its users.

[3] A transaction is not advertised to nodes if it is known that they have already learned about it.

[4] `https://github.com/ethereum/devp2p/blob/master/rlpx.md`

a good approximation of the victim's network view, the attacker can connect to all of the victim's neighbor peers as well as directly to the victim. By doing this, the network view of the attacker approximates the view of the victim, rushed or delayed by the network's latency.
Both protocols also differ in the way transactions are broadcast. As seen previously, Bitcoin uses an advertisement system and broadcasts a transaction only if the destination node specifically requests it. Additionally, *Bitcoin Core* introduced transaction *diffusion*, which adds random delays to the advertisement of new transactions to each neighbor peer. This prevents most neighbor nodes from immediately seeing a transaction originating from a node and makes obtaining a transaction from a third-party node that is not the originating node more probable. This advertisement *diffusion* falsifies the attacker's network view when only relying on the messages obtained from the victim and his or her neighbor peers instead of reading all visible connections from the wire. Ethereum broadcasts newly learned transactions immediately and also makes use of an advertisement system, but only for a subset of its neighbor peers. If an Ethereum node has n neighbor peers, each time a new transaction is learned, this transaction is first broadcast to $\lfloor\sqrt{n}\rfloor$ neighbors and then advertised to the remaining $n - \lfloor\sqrt{n}\rfloor$ neighbors, excluding neighbor peers known to have already learned about that particular transaction. Since transactions are broadcast and advertised without delay, the network view of the attacker is a good approximation of that of the victim.
Another important difference between Bitcoin and Ethereum nodes is the way connections to neighbor peers are handled. Bitcoin uses a **ping-pong** mechanism to test if a neighbor node is still alive. A **ping** message is sent every two minutes and a peer is dropped if it fails to reply to **ping** messages in an interval of 20 minutes. This rule enables an attacker to add a very high packet dropping percentage without disconnecting the victim from his or her neighbor peers due to the node being unreachable. If at least one of these 10 **ping** messages receives a **pong** message in return, peers are still considered to be alive and the connection is maintained. Ethereum also uses a **ping-pong** protocol to test the liveness of its neighbors. A **ping** message is sent every 15 seconds and the peer is dropped instantly if it fails to reply. Additionally, it instantly disconnects a peer if it receives an error in response to sending protocol messages or if a `TCP` connection returns an error. These differences are key. Bitcoin connections are long-lived and robust while Ethereum connections are short-lived and fragile. At best, the attack should only hardly be detectable at the application layer. The victim node should remain *in-sync* and its set of neighbor peers should remain stable. For Bitcoin, an attacker can apply a packet dropping rate to a victim's connections of approximately 25% while maintaining a healthy, somewhat stable set of connected neighbors. Such a drop rate is not possible for Ethereum. In fact, after some testing it was determined that even a very low percentage lead to a very unstable set of neighbors. It was found that a constant delay of approximately 1000 milliseconds maintains a healthy, somewhat stable set of peers. These observations led to the following decisions: Bitcoin nodes connect to a pre-defined set of selected peers while Ethereum nodes continuously discover new random peers and connect to them. For Bitcoin we selected a set of 200 peers using an online database: `https://bitnodes.io/nodes/leaderboard/`. We selected peers in the order of assigned leaderboard grade but applied some extra criteria, including the most recent Bitcoin protocol version[5], a recent *Bitcoin Core* version[6], a long uptime[7], and high best block number[8]. As for Ethereum, the framework restricts the victim node to outbound connections to simplify the simulation. This does not introduce any critical change nor does it impact how the victim node interacts with its neighbor peers. Inbound connections only know about the randomly

[5] 70015
[6] Greater or equal to version `0.17`.
[7] 7 days or more.
[8] Not less than the current best block minus 144, the average amount mined in a day, since the leaderboard is updated only once every 24 hours.

assigned remote ports on the neighbor peer's machine, which makes it more difficult for the attacker to connect to it. Most nodes use the default port but in the event they do not, the attacker would need to port scan the machine to determine the correct port to send a connection request to. To simplify this process only outbound connections are allowed.
An important ability of the attacker is to act on a fraction of the victim's network connections. The framework can apply different packet dropping and delay rules to a given percentage of observed connections, even when nodes run on the same machine. To enable this, the framework queries the current connections of the victim and applies the network conditions to the up- and downstreams using the local port allocated by the victim for the connections to the selected neighbor peers. This permits deterioration of victim connections without impacting either the attacker or the reference client. The following tool is used:

tc A traffic control and manipulation tool used to configure the kernel packet scheduler. Packet control consists of shaping, scheduling, policing, and dropping. The framework only makes use of the dropping and delaying features. Requires `sudo` rights. A description can be found at: `https://linux.die.net/man/8/tc`.

2.2 Related Work

The privacy guarantees of both Bitcoin and Ethereum have already been studied. The first attacks used tools such as the graph analysis in [2], the timing analysis in [8, 5, 3], and the network analysis in [1]. Eclipsing attacks have also been demonstrated to alter, filter, or control a victim's view of the blockchain in [6, 4, 7]. These attacks do not act directly on the quality of the victim's connections to its neighbor peers.
The novelty in the design and setup this framework supposes is that the attacker has strong network capabilities and is located on-path for a fraction of the victim's connections. This hypothesis gives the attacker powerful capabilities, such as directly acting on the connection's quality by dropping, delaying, or corrupting packets sent and received by the victim. This new setup opens up new attack vectors, because the attacker holds power over the pace at which the victim receives new transactions from his or her neighbors and can delay or shape the victim's network view at leisure. This semester thesis introduces the first fully automated framework by which to setup and simulate network level and on-path attacks.

Chapter 3

Design

3.1 Raw Packets and Logging

As seen previously, because Bitcoin communications are conducted in a non-encrypted manner any on-path actor is a potential eavesdropper. Let us suppose that the victim is running a Bitcoin node V on his or her own machine. This node V is connected to N_V neighbor peers. Let us also suppose that there exists another actor, on-path, between V and a fraction f_V of its neighbor peers, who is actively recording and dumping all packets. Let's call this actor the attacker. Because the payloads contained in these packets are not encrypted, this attacker potentially has the ability to reconstruct all communications. Taking this possibility into account, the simulation of any attack relying on this setup can also rely on logging all communication occurring at the victim node V instead of recording the raw packets and reconstructing the communication, which is a slow and resource-intensive process. The setup explained in this paragraph is depicted in figure 3.1.

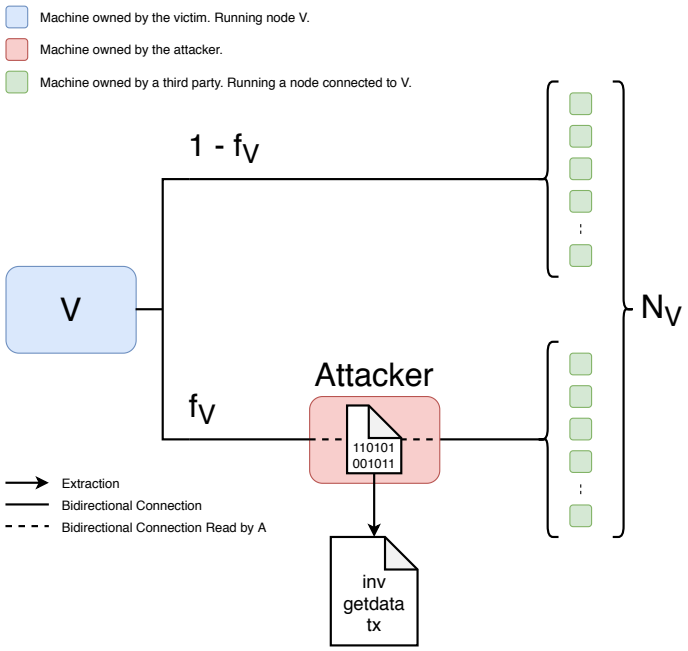

Figure 3.1: Communication Reconstruction From Raw Packets

To verify this hypothesis, I was provided with multiple `PCAP` files containing the raw packets recorded by the attacker seeing $f_V = 1$ communications. This verification process comprises multiple steps:

1. The `PCAP` file is split into 5 smaller files, each containing 20% of the communications streams. This helps simulate the fact that the attacker may not see $f_V = 1$ of all streams. Once the Bitcoin messages are reconstructed, the parts can be merged again to simulate fractions $f_V \in \{0.2, 0.4, 0.6, 0.8, 1\}$.

2. Each stream of packets between V and a neighbor peer contained in these smaller files is reconstructed through the stripping of duplicates and the reordering of out of order packets with the library *gopacket* presented earlier. The results are directional per neighbor streams of payloads sent from or to V.

3. Each of these payload streams is parsed and converted into Bitcoin messages of type **inv**, **getdata**, or **tx**. All other message types are ignored. For each of these messages the recording timestamp is applied, resulting in directional per neighbor streams of Bitcoin messages with recording timestamps. The age of each message is computed against a reference node[1]. This state is saved on disk as a `JSON` file, as this process may take multiple hours to execute.

4. From this intermediary `JSON` file, some simple pre-analysis is performed, such as the removal of all messages referring to transactions seen first by the reference node or that have been requested by the victim node. None of these transactions can originate from V.

5. Statistics for each transaction are extracted about the number of peers to whom the transaction has been advertised, the number of peers that requested the transaction from V, and the number of peers to whom the transaction has been broadcast. A `CSV` file is generated storing these statistics per transaction. By combining the resulting `CSV` files, more powerful attackers can be simulated.

This verification is powerful, as it enables us to simulate attacks following this design by removing the attacker entity during the simulation and recording a log stream directly at V. The final simulation design is a victim running a Bitcoin node V on his or her own machine, logging all messages sent to neighboring peers and recording a fraction f_V of these logs. The simulation setup is depicted in figure 3.2. As described above, one can also record using $f_V = 1$ and strip off logs from particular peers to simulate different levels of power with a single recording[2].

3.2 Framework

As suggested previously, nodes participating in the Ethereum network encrypt their communications in a *peer-to-peer* manner prohibiting third-party actors from eavesdropping on messages in the same way possible with Bitcoin and the setup discussed in 3.1. To obtain a comprehensive network view approximation of the victim, an attacker can directly connect to the victim and his or her neighbor peers. Because both the victim and the attacker are connected to the same peers in the network and broadcasting of transactions is not delayed in Ethereum, their views will be

[1] This reference node is accessible at `https://blockchain.info/` via an `API`.

[2] One is only allowed to deteriorate as many peers as the lowest power to be simulated. If the lowest attacker power to be simulate using a particular recording is $f_V = 0.3$, then during this experiment the attacker is allowed to deteriorate a maximum of 30% of all connections.

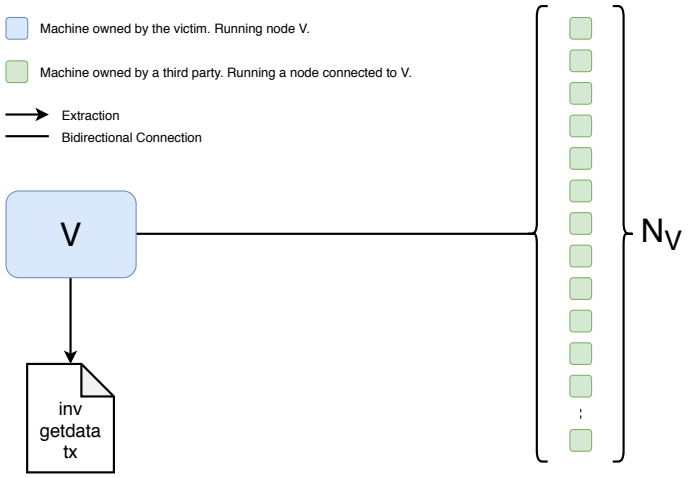

Figure 3.2: Communication Logged by Node V (Simulation)

very similar.
Let us suppose that the victim is running a node[3] V on his or her own machine. Additionally, let us suppose that another actor, on-path, exists between V and a fraction f_V of his or her neighbor peers, who is actively inspecting packets. Let us call this actor the attacker. In the case of Bitcoin, packets sent from V to neighbor peers are easily detected due to the fact that they are not encrypted. As for Ethereum, the attacker is still capable of inspecting the header of the packets and discovering the neighbor peers by port scanning the destination `IP` addresses to determine which are running a node. Additionally, let us now suppose that the attacker is running his or her own node A and connects to V. As the attacker inspects the raw packets and discovers neighboring peers of V, A connects to these peers. This process is dynamic: A connects with or disconnects from peers as V does. Because the attacker is on-path for these neighbor peers, deteriorating network conditions can be applied to these streams. For example, the attacker may choose to drop or delay a percentage of all forwarded packets from or to V. On his or her own node A, the attacker may now log any messages sent to and received from V as well as from and to all connected neighbors to obtain a view of the network very similar to that of V. Despite the fact that it does not make sense to attack a victim on the Bitcoin network using this particular design, this work has been conducted because it is very similar to the setup for Ethereum. This design and setup potentially allows for new types of de-anonymization attacks that will still work regardless of whether or not the communications are encrypted in the future and may be combined with other types of attacks to increase their effectiveness. The setup explained in this paragraph is depicted in figure 3.3.

The setup used to simulate these types of attacks is slightly different in the framework, as it is kept as simple as possible without breaking equivalence. For example, in the case of Bitcoin, both V and A run on the same machine to reduce costs and keep scripting as simple as possible. Both nodes will have the same public `IP` address, which is not an issue, as Bitcoin nodes do not have restrictions considering this parameter. In the context of Ethereum, restrictions exist per `IP` address when a new incoming connection is set up. A node will refuse a second incoming connection originating from the same `IP` address if both occur within an interval of fewer than 30 seconds. To simulate real conditions, it was decided that both V and A would run on machines having

[3]Either Bitcoin or Ethereum.

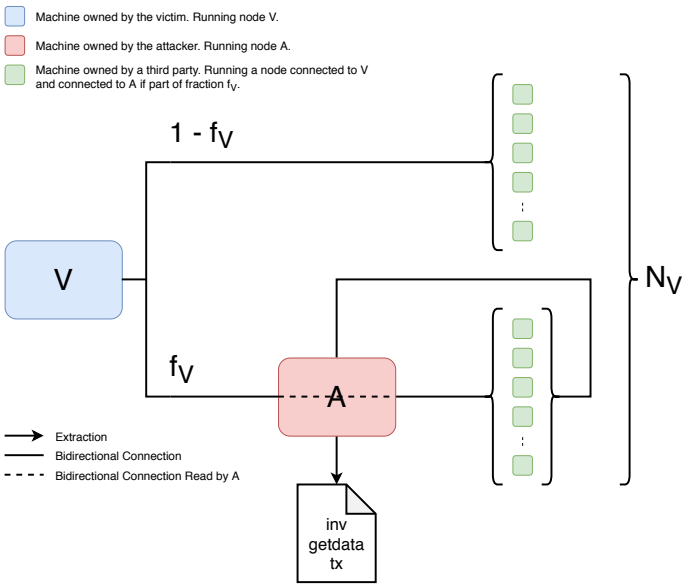

Figure 3.3: Setup Simulated by the Framework

different public `IP` addresses. Another modification in the framework to simulate this design is a direct channel between V and A to keep A connected to the same set of peers as V. A real attacker would follow the steps described in the previous paragraph. Additionally, during a simulation, the framework has access to the victim's machine; therefore, network conditions are applied directly at the source instead of at an on-path device. Finally, the framework sets up a reference node R connected to a set of pre-selected peers following the same conditions used for V and A in the case of Bitcoin and random peers for Ethereum.
The source code of both *Bitcoin Core* and *Geth* clients has been modified. Extra logging is introduced to keep track of exchanged messages from or to nodes V, A, and R. All advertising and broadcasting of transactions has been deactivated for A only. Because A is a passive[4] actor, it is possible to record experiments where the attacker sees $f_V = 1$ of the victim's neighbors and later strips off some neighbors from the resulting logs to simulate different values of f_V with a single recording. Finally, only for Ethereum, incoming connections have been disabled for V to bypass the required port scan on the neighbor's machine, permitting the attacker to discover the listening port. This does not change the protocol's behavior, as connections are not handled differently based on whether they are incoming or outgoing, but it simplifies the simulation within the framework.

3.2.1 Scripting

Let us examine the different scripting components that enable and constitute the framework. All scripts can also be used in a stand-alone manner. For Bitcoin, there are:

peers.py
: Queries the external `API` of the leaderboard hosted on `https://bitnodes.io/` and generates a set of peers complying with filters discussed in 2.1. This set of peers can be integrated in the configuration files of V, A and R.

[4]Passive only regarding transactions broadcast in the networks.

sync.py
Freezes the current set of common peers and applies network deteriorating rules to a fraction of the victim peers. This script has a delay prior to the stabilization of the set of common peers because of the 20-minute grace period before dropping a peer. This script runs until interrupted and resets all `tc` rules before quitting.

record.sh
Activates the internal logging of Bitcoin nodes and records all logging related to **inv**, **getdata** and **tx** messages. Runs until interrupted and resets Bitcoin's internal logging before quitting.

tx.sh
Executes a transaction on node V. This script contains a sleeping period before and after executing the transaction corresponding to the total recording time of an experiment.

auto.sh
Combines `record.sh` and `tx.sh` to enable automatic recording of experiments.

And for Ethereum we have:

ethereum.sh
Set up a private Ethereum network composed of a victim, an attacker, and a few mining third-party nodes. Can be used for local testing.

sync.py
Keeps the set of common peers shared by V and A in sync. Needs to run continuously in the background on the attacker's machine.

deter.py
Applies network deteriorating rules on a fraction of the victim's peers. To be run on the victim's machine.

record.sh
As for Bitcoin, the internal logging of the Ethereum nodes is activated and all advertising or broadcasting messages are recorded. Runs until interrupted and resets Ethereum's internal logging prior to quitting. To be run on the victim's machine.

tx.sh
Similarly to Bitcoin, it executes a transaction on node V. Includes a sleeping period before and after execution of the transaction corresponding to the total recording time of an experiment. To be run on the victim's machine.

auto.sh
As with Bitcoin, it combines `record.sh` and `tx.sh` to enable automatic recording of experiments. To be run on the victim's machine.

Making use of the modified *Bitcoin Core* and *Geth* clients combined with the above described scripts, it is possible to simulate attacks according to the design described in 3.2 with various levels of power.

Chapter 4

Evaluation

Results and outcomes of this thesis are not intrinsically quantifiable when discussing the fundamental framework produced to collect data as a first step against any attack relying on the design described in 3.2. Nevertheless, proof-of-concept analysis was conducted in order to demonstrate the usefulness of this thesis, which may pave the way for preventing new types of de-anonymization attacks against Bitcoin and Ethereum users.

4.1 Bitcoin

A first experiment was conducted to measure the delay introduced to the victim's view. The setup was as follows: both the victim and attacker were connected to the same set of neighbor peers, but V and A weren't connected to one another. During this experiment, the attacker saw $f_V = 1$ connections. A packet drop percentage of $25 - 30\%$ and a delay of 1000 milliseconds was applied to 100% of the victim's peers. The recording time ranged between 2 and 5 hours. The extracted statistics are listed in table 4.1, which shows that the victim's view was delayed up to several seconds in comparison with the view of the attacker. It is important to remember that *Bitcoin Core* nodes apply diffusion of transactions with a mean delay of 5 seconds.

Duration	Drop (%)	Common TXs (%)	Only Victim (%)	First Attacker (%)	Mean Adv. Attacker (seconds)
5h	30	94	5	85	1.47
3h	30	94	5	84	7.88
5h	30	93	5	76	6.85
3h30min	30	96	3	72	23.12
2h	30	93	4	97	4.52
3h	25	95	3	99	41.39
3h	25	84	12	99	15.50

Table 4.1: Experiment - Delayed View

An interesting metric to examine for a particular recording is the percentage of transactions seen only by the victim. This metric is unusually high when compared with the percentage seen only by the attacker. In order to seek an explanation for this difference, another experiment was conducted as follows. A first recording of logs was begun on both node V and A. After a delay of

20 minutes, a second recording was started in parallel with the first, again on both nodes. After an additional 20 minutes, both recordings were stopped. The resulting statistics of the second recording are presented in table 4.2, which reveals that 25% of all transactions are seen only by the victim node V. All transactions seen by the attacker during the first recording were extracted and compared with the transactions seen only by the victim during the second delayed recording. Common transactions were removed from the second set, the set of transactions seen only by the victim during the second recording. The results indicate that only approximately 3% of the original 25% of all transactions were actually not seen by the attacker during the first or second recording. This finding demonstrates that most of the attacker's unseen transactions during a recording were delayed transactions that were seen by the attacker before the recording is started. The remaining unseen 3% of the original 25% of all transactions may be explained by the fact that transaction broadcasting in the Bitcoin protocol is best effort, and the only guarantee that exists is that a node eventually receives a transaction.

Common TXs (%)	Only Victim (%)	First Attacker (%)	Mean Adv. Attacker (seconds)
67	25	85	3.57

Table 4.2: Experiment - Transactions Seen Only by Victim

A second experiment was conducted in which the victim and attacker were connected to the same set of peers; this time, V and A were connected to each other. The fraction of connections seen by the attacker was again $f_V = 1$, but no packet dropping percentage or delay was introduced, and all network conditions were left untouched. The recording time amounted to 10 minutes. Node A used the original implementation of transaction advertisement and broadcasting. The extracted statistics are given in table 4.3, which shows that, in comparison with the first experiment, the percentage of transactions only seen by either V or A was very low and similar in value. Another important observation is that, since the network conditions of the victim remain untouched, the percentage of transactions seen first by either V or A was approximately 50% each.

Common TXs (%)	Only Victim (%)	First Attacker (%)
95	3	47
94	3	47
97	1	47
97	2	46
97	1	47
98	1	46
97	2	47
94	3	48
91	4	44
91	4	46

Table 4.3: Experiment - No Deterioration

From these two experiments and the basic analysis conducted, we conclude that the view of V may be delayed by the attacker by an amount of time in units of seconds. As a result, the attacker can expect to see most transactions before the victim does. This observation is interesting, as the attacker may take action in light of the powerful knowledge that he or she is able to obtain: a network view similar to that of the victim but with an advantage in time.

4.2 Ethereum

As for Ethereum, two experiments were conducted that included proof-of-concept analysis to demonstrate the usefulness of the data collected through the framework. A more deep and complete analysis of the data must be conducted to extract more accurate and meaningful results.
The setup for both experiments was the same. Both the victim and attacker were connected to the same set of peers, and nodes V and A were connected to one another. The fraction of the connection seen by the attacker was $f_V = 1$. The applied network conditions for the victim's connection was a packet dropping percentage of 0% and a delay of 1000 milliseconds. The recording time amounted to 20 minutes. For these experiments, node A used the modified advertisement and broadcasting functions, i.e., no transaction advertisement or broadcasting was done by A. The first experiment deteriorated 50% of the victim's connections, while the second experiment deteriorated 100% of the connections. A total of 10 recordings were conducted for each experiment.
In order to predict which transactions originated from the victim node V, a few simple features were extracted from each attacker's recordings and then combined into a single dataset of labeled samples. The features generated per transaction were as follows: the percentage of reporting peers, whether the victim reported the transaction, whether the victim reported the transaction prior to all other peers, the victim's advantage[1] or disadvantage, the mean interval between reporting peers, the median interval between reporting peers, the skipping score, and the deteriorating score[2]. This dataset was fed to a very basic isolation forest trained on 50% of the non-outliers[3] and outliers. The other 50% of non-outliers and outliers were predicted[4] as follows by the isolation forest:

Conn. Deteriorated (%)	Outliers (Accuracy)	Normal (Accuracy)
50	1.0	0.96
100	1.0	0.97

Table 4.4: Predictions - Isolation Forest

This analysis is only a toy example and proof-of-concept that indicates that the recorded data may contain useful information to de-anonymize transactions from a given victim. It is assumed and expected that, in this analysis, the most prevalent feature detected by the isolation forest is that the first peer to report an outlier transaction is the victim itself.

[1] If the victim reports a transaction 2.3 seconds before everyone else, then the victim's advantage is 2.3.
[2] These two scores are explained in appendix A.
[3] In these experiments, an outlier is a transaction originating from the victim node V.
[4] Due to the small number of samples the accuracy is very sensitive to the used randomness to generate the forest.

Chapter 5

Outlook

This semester thesis presents an automated and integrated way of recording experiments assuming the design discussed in 3.2. These recordings may serve as the foundation for data analysis, timing analysis, machine and deep learning techniques, and other types of analysis. Attacks can be tested against real-world data recording happening on Bitcoin and Ethereum's mainnets with different kinds of parameters, such as the power or transaction advertisement and broadcasting algorithm of the attacker. The framework would directly benefit from the following two extensions:

Advertisement and Broadcasting Mode

As of now, different modes of advertisement and broadcasting of new transactions have to be changed manually in the code. The code must be compiled, and the nodes must be restarted with the new clients. A useful extension would be to integrate automatic mode changes for the attacker's node through an `API`. Three example modes are:

Diffusion

The standard mode for Bitcoin, in which the advertisement is delayed by a random amount per neighbor peer. Implementing this mode for Ethereum is illogical.

Sniping

The standard mode for Ethereum, in which all new transactions are instantly advertised to neighboring peers.

NOOP

A mode in which the attacker's node does no advertisement or broadcasting at all. The attacker node is passive considering transaction broadcasting.

Secure the RPC Server

Since the framework uses Bitcoin and Ethereum's mainnets, transactions consume BTC or ETH worth real money. The Bitcoin and Ethereum nodes have `RPC` servers running in order to interact with the nodes through an `API`. To date, this arrangement is not problematic for Bitcoin since all nodes run on the same machines and the `RPC` servers listen on localhost. As for Ethereum, the `RPC` servers must listen on the public `IP` address of the machines in order for one machine to communicate with another. This setup opens the `RPC` servers to the external world and may lead to node hijacking or a compromised wallet. The current solution is a firewall rule blocking all connections except from selected `IP` addresses.

5.1 Future Work

The following three ideas may serve as the foundation for future theses to build on this framework:

Attacks

The first rather self-evident idea is to use the framework to record experiments and to test different kinds of attacks against the data. Two attack ideas are:

Sniping

In this experiment, the attacker uses the **sniping** mode of advertisement and broadcasting. Although this mode is Ethereum's default mode, it is better to modify the algorithm so that the attacker sends out advertisements only and not transaction messages. Consequently, a peer will always request a transaction, and the attacker has a guarantee that the peer didn't yet see that particular transaction. In this experiment, the goal is to delay the victim's view long enough to give the attacker time to snipe all new transactions to all of the attacker's peers. Thus, the attacker should deteriorate 100% of the connections. If the delay is very large, the attacker will receive external transactions long before the victim and will have time to snipe the transaction to the victim before another peer sends it. The victim will request that transaction, and the attacker has the guarantee that the transaction does not originate from the victim node. Transactions originating from the victim node will be seen by the attacker before any neighbor peer, and the attacker will have time to snipe that transaction to all of the attacker's neighbors, who are also the victim's neighbors. All peers will request that transaction. Since no neighbor will have seen that transaction, it can only originate from the victim.

Two Sets

In this experiment, the attacker uses the **NOOP** mode of advertisement and broadcasting and listens passively. The goal is to create two sets of peers, namely, those who are deteriorated by the attacker and those who are not. If the attacker deteriorates 50% of the connections and the delay introduced by the deterioration is very large, far larger than the average network latency, then transactions originating from the victim should have a pattern seen by the attacker of two distinct sets of peers reporting the transaction with a delay in between. First all non-deteriorated peers will report the transaction since the path of the transactions is in two hops: from the victim to the non-deteriorated peer, and from that peer to the attacker. Since the introduced delay is much larger than the average network latency, the fastest way for the transaction to reach the attacker through a deteriorated peer is in three hops: first from the victim to a non-deteriorated peer, then from that peer to the deteriorated peer, and finally, from that peer to the attacker. Thus, both sets should report the transaction of the victim with some network latency in between. This particular pattern should not appear with external transactions.

Better Network Deterioration

Replace the tool `tc` and make it possible to apply deteriorating conditions on a per packet basis. For Bitcoin, this process is easier. Since packets are not encrypted, the attacker can read the packets and instantly forward packets such as pings or blocks while dropping or delaying packets about transactions, such as advertisements, requests, or transaction messages. The process is more complicated for Ethereum due to encryption. The attacker can only make a guess about the content of the packets. It might be possible to predict which packets are

pings, transactions, or blocks based on the packets' total size. Pings are expected to be very small, transactions somewhat larger, and blocks even larger. With packet size analysis, it may be possible to predict reliably the type of message in a packet and apply deteriorating conditions accordingly.

Test-labs

Extend the basic Ethereum test-lab and scale it to a size usable for testing and even attack execution. The conditions should be comparable to the ones on mainnet and should thus include participating nodes and simulated network conditions that mimic real-world latency. This kind of test lab is especially interesting for Bitcoin, as the transaction fee may increase over 1 USD, and experiments may thus become a cost factor.

Chapter 6

Summary

Pseudonymity can be confusing, as it may give a false impression of anonymity. It is difficult to fully grasp the complexity of the internet stack combined with the complexity of the Bitcoin and Ethereum protocols. This complexity increases the attack surface for bad actors presented with a vast range of potential exploitable processes. As previously mentioned, pseudonymity can be quantified in terms of the probability of successful linkage between a pseudonym and an identity. To help keep the users of Bitcoin and Ethereum pseudonymous, the final goal of this semester thesis was to propose an automated, all-inclusive, and flexible framework capable of executing various types of experiments assuming different levels of attacker power and behaviors. At the same time, simulation of these attacks should be kept as simple as possible within the framework. Each step transforming the attack setup in the real world into the setup in the simulation was verified to not break equivalence. In sum, simple toy examples of analysis were conducted on recorded experiments to introduce the useful nature of the data generated. To encourage projects related to this work, we provide interesting starting points for future theses.

Bibliography

[1] Biryukov, A., and Tikhomirov, S. Deanonymization and linkability of cryptocurrency transactions based on network analysis. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)* (2019), pp. 172–184.

[2] Chan, W., and Olmsted, A. Ethereum transaction graph analysis. In *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)* (2017), pp. 498–500.

[3] Fanti, G., and Viswanath, P. Deanonymization in the bitcoin p2p network. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1364–1373.

[4] Henningsen, S., Teunis, D., Florian, M., and Scheuermann, B. Eclipsing ethereum peers with false friends, 2019.

[5] Klusman, R., and Dijkhuizen, T. Deanonymisation in ethereum using existing methods for bitcoin.

[6] Marcus, Y., Heilman, E., and Goldberg, S. Low-resource eclipse attacks on ethereum's peer-to-peer network. Cryptology ePrint Archive, Report 2018/236, 2018. `https://eprint.iacr.org/2018/236`.

[7] Maria, A., Aviv, Z., and Laurent, V. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *Security and Privacy (SP), 2017 IEEE Symposium on* (2017), IEEE.

[8] Murali, S. *Using ethereum transactions to deanonymize senders.* PhD thesis, 2018.

[9] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf, 2009.

Appendix A

Appendix

A.1 Score Features

Let's recall the setup of the experiments described in 4.2. The attacker sees $f_V = 1$ of the victim's connections and he or she applies a packet drop rate of 0% and packet delay of 1000 milliseconds. The attacker's advertisement and broadcasting algorithm is **NOOP** as described in 5. We define the skipping score S and deteriorating score D as follows:

$$p = \sum_{i=1}^{n} i \tag{A.1}$$

$$S = \frac{\sum_{i=1}^{n} t_i}{p} \tag{A.2}$$

$$D = \frac{\sum_{i=1}^{n} e_i}{p} \tag{A.3}$$

where n is the total number of peers of the attacker, t_i is equal to i if the i^{th} peer to report the transaction is not deteriorated, otherwise 0, and e_i is equal i if the i^{th} peer to report the transaction is deteriorated, otherwise 0.
These scores are especially interesting for the first experiment, with 50% of the victim's peers deteriorated, as they partition the two sets of peers reporting a victim's transaction as described in 5.1. As the attacker deteriorates 50% of the victim's peers, the two sets are expected to have the same size, thus, the first set of reporting peers is expected to produce a low S score and the second set of peers reporting is expected to produce a high D score. Transactions not originating from the victim are expected to have similar S and D scores.

A.2 Tutorial

```
# This tutorial is only a sketch. Please refer to the framework's source code,
  script and configuration files as it makes some assumptions about folder
  structure and IP addresses.
# Bitcoin Core and Go Ethereum's code base contain strings "TODO CORE" and
  "TODO GETH" to mark additional or modified code.
$ REPO=<absolute path to the root directory of this framework>
```

A.2.1 Bitcoin Core

Compile

```
# Install main dependencies.
$ sudo apt install build-essential libtool autotools-dev automake pkg-config
  bsdmainutils python3 libevent-dev libboost-system-dev libboost-filesystem-dev
  libboost-test-dev libboost-thread-dev

# Install dependency for wallet.
$ wget http://download.oracle.com/berkeley-db/db-4.8.30.NC.tar.gz
$ tar -xvf db-4.8.30.NC.tar.gz
$ cd db-4.8.30.NC/build_unix
$ mkdir -p build
$ BDB_PREFIX=$(pwd)/build
$ ../dist/configure --disable-shared --enable-cxx --with-pic --prefix=$BDB_PREFIX
$ make install

# Compile Bitcoin Core with configuration.
$ cd $REPO/bitcoind
$ ./autogen.sh
$ ./configure CPPFLAGS="-I${BDB_PREFIX}/include/ -O2"
  LDFLAGS="-L${BDB_PREFIX}/lib/" --without-gui --without-miniupnpc
$ make check

# Compile Bitcoin Core when already configured.
$ cd $REPO/bitcoind
$ make check
```

Run

```
$ cd $REPO/bitcoind/src
$ ./bitcoind -conf=$REPO/bitcoind/bitcoin-attacker.conf
$ ./bitcoind -conf=$REPO/bitcoind/bitcoin-victim.conf
$ ./bitcoind -conf=$REPO/bitcoind/bitcoin-ref.conf
```

Experiment

```
$ cd $REPO/bitcoind/src

$ LOSS_PERC_FOR_DETER_CONN=<packet drop percentage>
$ DELAY_MS_FOR_DETER_CONN=<packet delay in milliseconds>
$ CONN_DETER_FRACT=<percentage of peers to deteriorate for the victim node>
$ CONN_NON_DETER_STILL_CONNECT=<percentage of non-deteriorated peers the
  attacker still connects to>
$ NEXT_TX_ID=<ID of the next transaction to execute>
$ HOW_MANY_TX_TO_RECORD=<how many transactions to record in total>

# In a first terminal.
```

```
$ python3 ../sync.py --exec 100 $LOSS_PERC_FOR_DETER_CONN
  $DELAY_MS_FOR_DETER_CONN $CONN_DETER_FRACT $CONN_NON_DETER_STILL_CONNECT

# In a second terminal.
$ bash ../auto.sh $LOSS_PERC_FOR_DETER_CONN $DELAY_MS_FOR_DETER_CONN
  $CONN_DETER_FRACT $CONN_NON_DETER_STILL_CONNECT $NEXT_TX_ID
  $HOW_MANY_TX_TO_RECORD
```

A.2.2 Go Ethereum

Compile

```
# Install main dependencies.
$ sudo apt install build-essential
$ sudo snap install go --classic

# Compile Go Ethereum.
$ cd $REPO/ethereum
$ make all
```

Run

```
$ cd $REPO/ethereum

# On the attacker's machine.
$ ./attacker.sh
$ nohup ./build/bin/geth --type ref --verbosity 3 --cache 4096 &>
  .ethereum/log/ref.log &

# On the victim's machine.
$ ./victim.sh

# On the attacker's machine again.
$ nohup python3 -u sync.py --exec &> .ethereum/log/sync.log &
```

Experiment

```
$ cd $REPO/ethereum

$ LOSS_PERC_FOR_DETER_CONN=<packet drop percentage>
$ DELAY_MS_FOR_DETER_CONN=<packet delay in milliseconds>
$ CONN_DETER_FRACT=<percentage of peers to deteriorate for the victim node>
$ CONN_NON_DETER_STILL_CONNECT=<percentage of non-deteriorated peers the
  attacker still connects to>
$ NEXT_TX_ID=<ID of the next transaction to execute>
$ HOW_MANY_TX_TO_RECORD=<how many transactions to record in total>

# In a first terminal on the victim's machine.
$ python3 deter.py --exec $LOSS_PERC_FOR_DETER_CONN $DELAY_MS_FOR_DETER_CONN
```

```
  $CONN_DETER_FRACT $CONN_NON_DETER_STILL_CONNECT

# In a second terminal on the victim's machine.
$ bash auto.sh $LOSS_PERC_FOR_DETER_CONN $DELAY_MS_FOR_DETER_CONN
  $CONN_DETER_FRACT $CONN_NON_DETER_STILL_CONNECT $NEXT_TX_ID
  $HOW_MANY_TX_TO_RECORD
```