www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculteit Wetenschappen

School voor Informatietechnologie

master in de informatica

Masterthesis

Tracing ransomware transactions on the blockchain

Maarten Evenepoel

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :
Prof. dr. Peter QUAX

2021
2022

Faculteit Wetenschappen

School voor Informatietechnologie

master in de informatica

Masterthesis

Tracing ransomware transactions on the blockchain

Maarten Evenepoel

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

Prof. dr. Peter QUAX

Acknowledgements

Writing this thesis would not have been possible without the guidance and support of several people.

First and foremost I would like to thank my promotor, Prof. Dr. Peter Quax, and co-promotor, Prof. Dr. Wim Lamotte, for their guidance throughout the development of this thesis and allowing me the opportunity to develop this thesis within the context of the “Networked and Secure Systems” research group.

Next I would like to thank my close mentors throughout the past year, Prof. Dr. Jori Liesenborgs and Mr. Isaac Meers. I would like thank Isaac for the weekly follow-up meetings we held. During these meetings Isaac would put extensive time and effort in answering the questions I had. He was always there to either help guide me into the right direction, or give me the reassurance I needed when I doubted my own ways. Unfortunately, Isaac had to part from mentoring me due to other obligations.

This is where Prof. Dr. Jori Liesenborgs stepped in to replace Isaac. I would like to thank Prof. Liesenborgs for his flexibility and willingness to guide me, last minute, into the final weeks of writing this thesis. His feedback on my progress and his insightful questions coming from his own expertise thaught me to be critical of my own work and gave me the motivation to successfully finalise my thesis.

I would like to give my thanks to Ms. Masarah Paquet-Clouston, assistant-professor at the University of Montreal, for giving me additional insight into their own research related to tracing ransomware payments and providing me with additional data I could use to debug and verify the tooling developed for this thesis.

I would like the development team of GraphSense [1] for giving me access to their demo environment and being very responsive to questions I had concerning their API.

Similarly, I would also like to thank Mr. Ales Janda, developer of WalletExplorer.com, for granting me access to the JSON API of his webservice and giving me additional insight into the technicalities of how he developed it.

At last, I want to thank my family and my girlfriend for the moral support they provided when necessary.

[1] https://graphsense.info/

Abstract

In the current digital age, with an omnipresence of servers, computers and personal digital devices, ransomware attacks have become an equally omnipresent threat to these devices. Ransomware attacks happen on a daily basis targeting businesses and individuals alike. In today's climate, ransomware attacks often require victims to pay their ransom in Bitcoin or Monero. In this thesis, we try to determine what tracing these payments throughout the Bitcoin ecosystem would entail and which insights can be gained from doing this. The approach taken towards answering this question is the practical implementation of a new, open-source tool named TRaP. With TRaP, every step along the way to generate these insights is automated and universally executable on future ransomware attacks in a manner that is as lightweight as possible. The development of TRaP also revealed the limitations of tracing payments throughout the Bitcoin ecosystem and what could be done to deal with these problems.

Secondly, this thesis also tries to transfer the knowledge discovered for tracing payments throughout the Bitcoin ecosystem to Monero. Monero is another privacy-based cryptocurrency that is heavily on the rise to become a prominent part of the coming ransomware climate. We study which aspects of tracing payments throughout Bitcoin are transferable to Monero, being a privacy coin. From this, it could be concluded that Monero poses many difficulties in tracing payments that are not present, or only limited, in the Bitcoin ecosystem. This justifies Monero's steep rise in popularity in the ransomware community.

Contents

Chapter 1

Introduction

In today's day and age, almost all digital devices are connected to the internet. Businesses and organisations, as well as individuals alike rely daily on digital devices, be it either for personal use or for work. This omnipresence of internet-connected devices therefore makes them a coveted target for various cyber criminal activities, such as attacking them with malware. *Malware*, portmanteau for "malicious software", is any software intentionally designed to bring damage to a computer, server, or computer network [Moi09]. The most prominent, and most lucrative form of malware today is *ransomware* [Pal21].

Ransomware is a specific type of malware designed to infiltrate a victim's servers or personal system, take the data or other resources on the system hostage encrypting it or blocking access to it, and request a ransom to be paid for the data to be released again. Ransomware attackers can pressure victims to pay their ransom using various ways. If the victim is a large corporation, the attackers can threaten to leak confidential hostage data to the general public if the requested ransom is not paid. An example of this is the recent alleged ransomware attack on NVIDIA, a company that is mostly known for the production of video cards. The attack was orchestrated by the "Lapsus$" hacker group. Company information was leaked online, although the attack did not impact the company's operation [Pan22].

If the victim is an individual, the attackers can pressure victims with threatening to delete the hostage data. Often times this is personal data that is inconvenient to lose, thus pressuring the victim to pay the ransom.

The current popularity of ransomware can be attributed to two causes. Ransomware is both lucrative, and easily accessible for potential attackers, meaning they are relatively low-effort. The "big hitting" ransomware attacks usually target businesses or corporations, since the stakes of losing data or the risk of leaked company information in these cases is much higher than for individuals. In 2021, one in ten organizations paid 1 million dollars in ransom payments, while the lowest average payments in the healthcare industry and the government industry were around 200.000 dollars, proving the lucrativity of the attacks [Jon22].

Conducting ransomware attacks is also a very accessible business for cyber criminals. This is partly due to the existence of the *Ransomware-as-a-service (RaaS)* business model. In this model, a RaaS operator orchestrates and deploys a ransomware attack on demand for a third party. Perhaps for a fixed price or for a percentage of the revenue generated by the attack. This makes ransomware attacks essentially accessible for anybody. The largest known RaaS operator in 2021 was the group behind the DarkSide ransomware family. The DarkSide group is mostly known for the Colonial Pipeline attack in May of 2021 [Ker21]. The aftermath of the attack was devastating. For the first time in 57 years, Colonial shut down its operations temporarily, resulting in long waiting lines at gas pumps spread throughout the United States, leading to a temporary increase in gas prices as well [TM21]. In total, 90 million dollars in ransom payments were collected from several different ransomware attacks to the DarkSide group, of which 15.5 million dollars went to DarkSide developers, while the remainder was spread around to other affiliates, according to a report produced by Elliptic, a cryptocurrency forensics company [Rob21].

A ransomware attack is evidently always associated with the request of a ransom payment. To make the ransom payment, attackers typically request the payment to be made in cryptocurrencies. Attackers chose to use cryptocurrencies for their perceived anonymity. Bitcoin is currently the predominant cryptocurrency of choice for cyber criminals [Fin21]. This is mainly due to Bitcoin being the most well known, and the easiest to acquire for victims that have no tech literate background. Alternatively to Bitcoin, Monero is another cryptocurrency that is heavily on the rise in cyber crime activities. The reason for this is that Monero is a privacy coin. Privacy coins try to hide as much information about their users and transactions as possible. This is beneficial for cyber criminals, as their steps throughout these cryptocurrencies are even harder to trace.

This brings us directly to the focus of this thesis. With a steep rise in ransomware attacks in 2021 alone, the need arises for forensics tools that can help in tracing these ransomware payments in order to gain insights into specific ransomware families and how they operate. In this thesis, we will study which procedures are necessary and feasible in order to trace ransomware related payments across the blockchain. This will be done through the development of a custom tool named *TRaP*, short for *Tracer of Ransomware Payments*. The intention of TRaP is to automate as much of the tracing procedure as possible and generate quick insights into an attack, based on the information that the tool is fed. TRaP will be specifically tailored to operate on the Bitcoin blockchain, since Bitcoin is still by far the most frequent cryptocurrency of choice in ransomware attacks to this date [Fin21]. At the end of the thesis, a closer look will be taken at Monero and the features that make it a privacy coin. In this chapter, an assessment will be made about the impact a privacy coin like Monero has on the tracing procedure that is used for Bitcoin. An assessment will be made on the transferability of tracing procedures from Bitcoin to Monero.

We can now formalize the research questions that have driven the development of this thesis:

- Is it possible to trace ransomware related payments across a cryptocurrency blockchain?
  - What would this entail?
  - What are the pitfalls? What is not possible?
  - Which insights can be gained from doing this?
- What effects do privacy coins have on the traceability of ransomware related payments? Do they limit traceability compared to transparent cryptocurrencies like Bitcoin?

This thesis text is divided into 5 chapters, following this introduction.

In order to grasp the procedures that are necessary to try and trace ransomware payments across the Bitcoin blockchain, some background information is required. In chapter 2 the necessary background will be provided to understand the technicalities and inner workings of the Bitcoin ecosystem as a whole. This gives the necessary context for the chapters that follow.

Chapter 3 will describe the steps that can be taken in order to trace ransomware payments in Bitcoin, and gain insight into an attack.

The following chapter, chapter 4, will then make a deep dive in the development of TRaP. This chapter will also include the concrete results that can be obtained with it, as well as an empirical evaluation of the tool and a personal reflection on the final result. The conclusion of this chapter will also discuss the limitations that were discovered along the way.

With chapter 5, another perspective on tracing ransomware payments is given from the perspective of Monero instead of Bitcoin, and by extension transparent coin vs privacy coin. The discussions in this chapter will try to give context to the difficulties that present themselves in trying to trace ransomware payments, or any payments for that matter, across the Monero blockchain.

A concrete answer to the proposed research questions will then be formulated in the conclusions in chapter 6, as well as a nod towards future projects and a personal reflection on the process of writing this thesis.

Chapter 2

Background

The purpose of this chapter is to provide the reader with the basics of ransomware attacks and blockchain technology. It is expected the reader has some background in computer science, equal to that of a student with a related master's degree. First we will describe how a typical ransomware attack works. Secondly, the necessary background in Bitcoin and its related blockchain technology will be studied in order to help understand the concepts necessary to enable some form of transaction tracing in later chapters.

2.1 Ransomware

Ransomware is a form of malware that attempts to encrypt a victim's data and in turn requires the victim to pay a ransom in order to be able to decrypt the files again. In this section, we will discuss the different stages in the timeline of a ransomware infection as also described in [Hua+18].

- **Delivery**: Ransomware can be delivered to a victim through a range of vectors similar to the ones used to spread generic malware [Del15]. One of the most common ways of distributing ransomware, or malware in general, is through malicious links in phishing mails. In this case a victim is lead to a website that unknowingly downloads the malware onto the victim's device by persuading the victim in some way to interact with malicious components in the website or the original email. These components could be simple hyperlinks or flashy buttons.

  Another popular way of distributing malware is through malicious advertising. In this case the malware gets downloaded to the victim's system if the victim visits a web page containing a malicious advertisement. However, these types of infection require the exploitation of some bug or zero-day on the victim's device. This makes these types of attacks more platform dependant. Not only email or web-based technologies can be used to spread malware. Malware could also be installed on a physical drive (such as a USB thumb drive) and left behind somewhere for a victim to be found. If the victim inserts the physical drive into its system, the malware can install itself on the system. In 2020, the *Try2Cry* malware was spread using precisely this method [Arg20].

  In order to mitigate the distribution of malware, it would be best if users consistently update the software they use on their system. This reduces the amount of possible vulnerabilities and in turn reduces the chance of getting a malware infection.

  It is also important to notice that ransomware attacks do not exclusively target PC-based devices, but also mobile devices, NAS devices, servers, etc. Albeit in a smaller capacity.

- **Execution**: When a ransomware binary gets executed on a new system, it silently starts to encrypt data. This data might vary from data that the ransomware deems valuable to the user, or simply all of the data on the host system. When the encryption succeeds, the victim is presented with a *ransom note*. This note informs the user that its system has been infected with ransomware and that the user's data is held for ransom.

- **Payment**: The *ransom note* also informs the victim that they can decrypt their encrypted data by paying a ransom. The first variants of ransomware required the victim to physically mail the ransom to the attacker. The first ransomware that we know of, PC CYBORG/AIDS Trojan-1989 used exactly just this payment method [GJA06]. Modern ransomware, however, makes use of the emergence of cryptocurrencies. Since cryptocurrencies are pseudo-anonymous, they provide an effective way for the victim to pay the ransom to the attacker, while the attacker can remain anonymous.

Typically, the cryptocurrency of choice is Bitcoin [Nak08], [Fin21], since this is the currency most victims already have some familiarity with by hearing of it in media. In some cases, the ransom note might also include a guide on how to purchase Bitcoins from *exchanges*, online services that allow users to trade fiat currency for cryptocurrencies and vice versa.

Ransom notes also include *ransom addresses*. These are the wallets the victim is expected to pay the ransom into. Depending on the ransomware family, each victim might get a unique wallet to pay the ransom into (e.g. Locky and Cerber) or every victim might be instructed to pay into the same wallet. In the latter case, it is often difficult for the attackers to identify individual victims. In this case, the attackers might require the victim to provide the payment transaction hash via email or via some web platform to verify the payment, or they simply do not decrypt the victims data at all.

- **Decryption**: When the ransom payment is confirmed, the attacker might remotely trigger the ransomware executable to decrypt the victim's files, or the victim might be provided with an executable to decrypt the files on his own.
- **Liquidation**: When the ransomware lifecycle ends, the attacker might want to cashout and exchange the cryptocurrency for fiat currency. The attacker might do this by exchanging the cryptocurrency at an exchange service, however, in order to remain anonymous, some attackers first deposit their cryptocoins into a *mixer* [Llc20]. Mixers obfuscate Bitcoin trails by intermixing coin flows from multiple sources. A more in-depth look at this problem will be taken later in chapter 3 and 4.

2.2 Bitcoin

As described in the previous section, ransomware payments are nowadays often made using Bitcoin or possibly other cryptocurrencies as well. Cryptocurrencies are built on blockchain technology. This section will take a closer look into blockchain technology and provide the necessary knowledge in order to later understand how someone might trace payments throughout a cryptocurrency blockchain. The knowledge that is presented in this section is built on how blockchain technology is applied specifically in the Bitcoin network. There are two reasons for this. The first is that blockchain technology was first proposed to enable the creation of the Bitcoin ecosystem, and is thus closely related to Bitcoin. The second reason is that most modern ransomware variants use Bitcoin as a means of ransom payment since Bitcoin is perceived as the most well known and and best established cryptocurrency among the general public [Fin21].

Why blockchain?

On October 31, 2008 an anonymous identity using the pseudonym "Satoshi Nakamoto" published a whitepaper called "Bitcoin: A Peer-to-Peer Electronic Cash System" [Par21] [Nak08]. In this whitepaper the anonymous author suggests that the conventional financial system suffers several weaknesses due to being built on a trust-based model. The argument used here is that financial institutions cannot avoid mediating disputes, as a certain amount of fraud is deemed acceptable. This also means that transactions are not certain to be non-reversible, which means the need for trust spreads. These uncertainties can be avoided in person using physical currency, but no mechanism exists to make payments over a digital communications channel, like the internet, without involving a trusted third party. Because of this, the author suggests an electronic payment system should be created that is built on cryptographic proof instead of trust. This also means transactions have to be computationally impractical to reverse to protect users from fraud. As a response to this, the author presents Bitcoin, which is a digital currency built using blockchain technology that abides to these core ideas. The Bitcoin system is a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The idea is also that the system will remain secure as long as the amount of honest nodes in the Bitcoin network collectively poses more compute power than any amount of cooperating group of attacker nodes.

2.2.1 Wallets

In order to exchange money between users of the network, each user needs to create what is called a *wallet* [Ant17]. A Bitcoin wallet is a piece of software every participant in the network runs on their own device and is personal to them. The term "wallet" can be interpreted in multiple ways. On a high level, a wallet is a user-level application that serves as the primary user interface to control a user's Bitcoin, generate and manage keys and addresses, tracking wallet balance, etc.. On a lower level, a wallet refers to the data structure used to store and manage a user's keys. For the continuation of this chapter, this is the definition of a "wallet" that will be used in the thesis from now on.

A common misconception about Bitcoin wallets is that they hold Bitcoins themselves. This is not the case. Wallets only contain keys. The actual Bitcoins are recorded in the blockchain on the Bitcoin network. Users only control the coins registered on the blockchain by signing transactions with the keys that are managed in their own wallets. Metaphorically, another way to look at a Bitcoin wallet would be to look at it as a *key chain*. More information on transactions and how they are signed is explained in the transactions section 2.2.2 later in this text.

There are two primary types of Bitcoin wallets: *non-deterministic* and *deterministic wallets.*

Non-deterministic wallets

Non-deterministic wallets, otherwise known as *random wallets*, *JBOK wallets* (Just a Bunch Of Keys) or *Type-0 wallets* are wallets where each key is derived from a new, independent random number. This means that all keys generated in these wallets are never related to each other, since they are all calculated from a different, independent random number. Figure 2.1 shows a visualisation of a non-deterministic wallet.

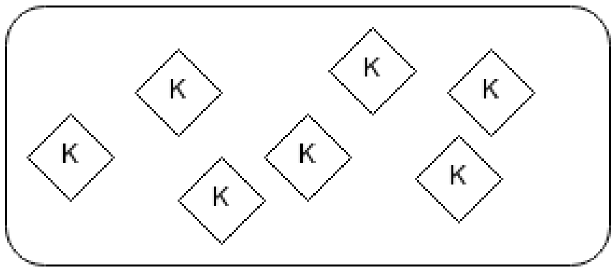


Figure 2.1: Visualisation of a non-deterministic wallet.

Because of the large set of unique keys, non-deterministic wallets have the disadvantage that they require frequent backups of all the individual keys. If a key in the wallet is lost all funds in the blockchain that are associated with the lost key will be inaccessible forever.

The idea of having to backup these keys also conflicts directly with the principle of avoiding *key reuse.* Reusing keys/addresses reduces the privacy of a wallet by associating multiple different transactions to the same address and by extension the same Bitcoin wallet. Because of this, reusing keys is strongly recommended against because it reduces the overall privacy of the wallet. By extension, non-deterministic wallets are also recommended against because they create the need to make frequent backups of the collection of keys in the wallet, which means an attacker that obtained the backup could easily associate all transactions in the blockchain with the keys in the wallet it obtained the backup from. An alternative to the non-deterministic wallet that partly solves these problems is the *deterministic wallet.*

Deterministic wallets

Deterministic wallets, *seeded wallets* or *Type-1 wallets* are wallets that contain private keys that are all generated from the same seed by using a hash function. The seed is randomly generated only once, during the creation of the new wallet. The seed is then the only piece of the wallet that needs backup, since the seed can be used to generate all the keys that are derived from it. On the other hand, if the seed of the wallet is lost, all keys derived from the seed are also immediately lost, meaning all funds associated with the wallet are inaccessible. It is clear the seed must be handled with care an must not be lost.

This mechanism also allows for easy transferring of wallets between wallet implementations, since only the seed needs to be transferred to the new wallet implementations, and all keys can be derived again in the new implementation. Figure 2.2 shows a graphical visualisation of a deterministic wallet.

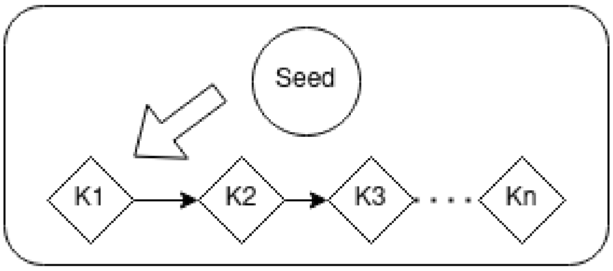


Figure 2.2: Graphical visualisation of a deterministic wallet. A deterministic wallet generates its pairs based on a fixed seed.

No frequent backup of the keys maintained in the wallet is necessary, like it was with the previous non-deterministic wallet. Only the seed of the wallet needs to be stored. Off course, this introduces the vulnerability that an attacker could somehow discover the seed of the deterministic wallet and use this to generate all addresses associated with that wallet, starting from the seed. However, if the seed is kept a secret, it is practically very difficult to try and discover all addresses belonging to the wallet from a third-party standpoint.

Asymmetric cryptography

The idea of private and public keys in wallets is built on asymmetric cryptography [RSA78]. One of the purposes of asymmetric cryptography is that the private and public keys can be used to send messages between two communicating agents that are cryptographically secure, meaning that only the sender and the recipient will know the actual contents of the message. The main idea here is that anyone can generate a pair of private and public keys. Anyone who then wants to send a cryptographically encrypted message can ask the public key of the recipient. The public key can be used to encrypt the message. By design, the public key cannot be used to also decrypt a given message. For this, the corresponding private key to the public key is needed. Hence why anyone should always keep their private key a secret. It should also be clear that it is required that no private key can be derived from a given public key. In figure 2.3 an illustration is given on how the asymmetric encryption scheme works.

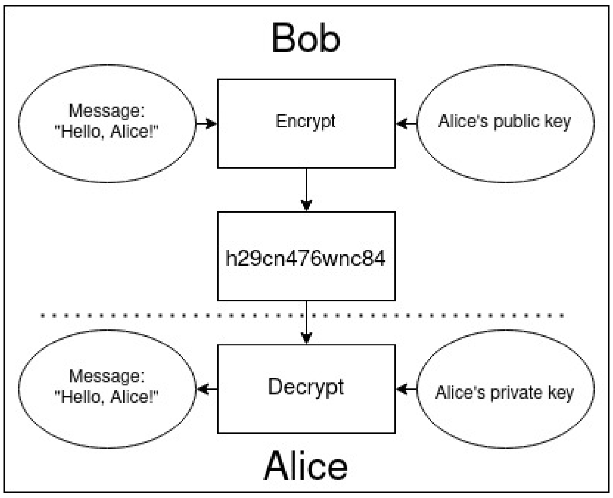


Figure 2.3: Illustration showing an asymmetric key encryption scheme. Bob wants to send a message to Alice and encrypts the message using Alice's public key. Alice can then later decrypt the encrypted message using her own private key.

Besides sending and receiving cryptographically secure messages, asymmetric cryptography also provides a way to digitally *sign* messages. To sign a message, one can generate a digital signature of a piece of data using their own private key. Since their public key will be publicly available, anyone in possession of the corresponding public key can decrypt the contents of the message again and verify it is the same as the original data. This mechanism is especially useful in the Bitcoin ecosystem. Anyone creating a new outgoing transaction out of their own wallet can sign the new transaction, while other members of the network can verify the "author" of the transaction using their public key to verify the corresponding signature. This way users can only generate transactions to spend money out of their own wallets, and not others. Figure 2.4 shows a graphical representation of how a digital signature works.

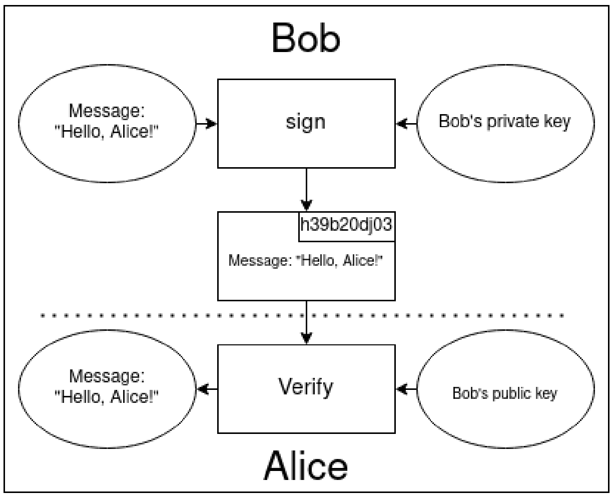
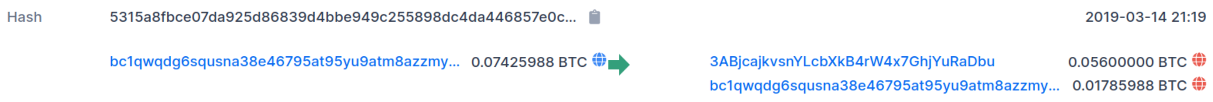


Figure 2.4: Figure showing the different parts of a transaction.

2.2.2 Transactions

Transactions are the core of the Bitcoin system. Everything else in the Bitcoin system is designed and created to enable the creation, propagation, validation and the placement of transactions on the Bitcoin blockchain. A transaction encodes the transfer of some value between two participants of the Bitcoin ecosystem [Devb].

Typically a transaction is assumed to look something like figure 2.5. However, an actual transaction looks very different from one provided by a block explorer. Block explorers add several high level constructs that are typically provided in application-layer user interfaces that operate on the Bitcoin network. These constructs make it easier for humans to reason about what is going on with transactions, but some of these constructs do not actually exist in the Bitcoin system. To show this, the same transaction as shown by a block explorer in figure 2.5 is represented by a data structure as shown in listing 1 in the Bitcoin system.



```
Hash    5315a8fbce07da925d86839d4bbe949c255898dc4da446857e0c...                                      2019-03-14 21:19

        bc1qwqdg6squsna38e46795at95yu9atm8azzmy...  0.07425988 BTC    3ABjcajkvsnYLcbXkB4rW4x7GhjYuRaDbu            0.05600000 BTC
                                                                      bc1qwqdg6squsna38e46795at95yu9atm8azzmy...  0.01785988 BTC
```

Figure 2.5: An example how one might typically recognize a transaction in web-based Bitcoin block explorers.

What is important to notice when looking at an actual transaction as represented in the Bitcoin ecosystem is that there is no direct mention of addresses of sending or receiving entities. This is because these are constructs that are used at a higher level to make transactions more understandable to end users, but they are not used in this form in the actual Bitcoin system. Instead of using these high level constructs, transactions use Inputs and Outputs to manage what is called the Unspent Transaction Output set. Also called the UTXO set.

The UTXO set

The smallest, indivisible chunks of Bitcoin currency that are recognized as valid throughout the Bitcoin system are, perhaps counter-intuitively called "*transaction outputs*". Transaction outputs can possess any amount of currency denominated in *Satoshis* (1 Satoshi equals 0.00000001 Bitcoin [Bra13]). Transaction outputs are atomic and thus cannot be divided or merged with other transaction outputs. They are immutable. Transaction outputs always have an owner, and only *one* owner. What this means is that a

```
{
    "txid": "5315a8...",
    "hash": "ed59a7...",
    ...,
    "vin": [
        {
            "txid": "568da8...",
            "vout": 1,
            "scriptSig": {
                "asm": "",
                "hex": ""
            },
            "txinwitness": [
                "",
                "30440...",
                ...
            ],
            "sequence": 4294967295
        }
    ],
    "vout": [
        {
            "value": 0.056,
            "n": 0,
            "scriptPubKey": {
                "asm": "OP_HASH160 5d2f745c0aaf2fc08e514624f4854aa280e43448
                            OP_EQUAL",
                "hex": "a9145d2f745c0aaf2fc08e514624f4854aa280e4344887",
                "reqSigs": 1,
                "type": "scripthash",
            }
        }
    ],
    ...,
    "time": 1552594745,
    "blocktime": 1552594745

}
```

Listing 1: A transaction as represented in the Bitcoin ecosystem. Only the key fields in a transactions object are kept in this listing. Other, less significant meta-data related fields are kept out. What this listing shows is that the key components of a transaction are mainly its *vin*-set and its *vout*-set, which determine the collections of inputs and outputs to the transaction.

transaction output can only be spent by using one of the keys out of the owner's wallet. A transaction output can thus not be spent by anyone other than its rightful owner, the owner of the corresponding key.

Bitcoin full nodes continually track all transaction outputs that are still unspent. These transaction outputs are called *unspent transaction outputs* or *UTXO* [Del+19]. The set of all UTXO is called the UTXO set and is duplicated in all participating full nodes of the Bitcoin network. Besides keeping track of the UTXO set, Bitcoin full nodes can also create and store wallets. When using a wallet on a full node, the wallet keeps track of all UTXO in the global UTXO set that specifically can be spent with one of the keys in its own Bitcoin wallet, thus one of the UTXO that is in possession of the user owning the wallet. The total balance of a Bitcoin wallet is the sum of the values associated with all UTXO in the UTXO set that can be spent with one of the keys in the wallet. But how does an UTXO get associated with a wallet in the first place?

Because UTXOs are immutable, some mechanism is needed to modify the UTXO set to move currency from one wallet to another (or to several others). This is where *transactions* come into play. A transaction is essentially a mechanism to transform the UTXO set by consuming (removing from the UTXO set) UTXO allocated to the sender's wallet and creating new UTXO that is allocated to the receiver's wallet. In doing so, one can remove value from its own wallet by removing one or more UTXO assigned to itself, and transfer the value to another wallet (or multiple other wallets) by creating new UTXO assigned to the receiving wallets. This is the essential idea behind a transaction and allows for a transfer of currency from one wallet to another. However, since UTXOs are immutable, the UTXO that go into a transaction may not always be exactly the amount someone actually wants to transfer to a receiver. Let's say for example Alice wants to send 15 Satoshi to Bob. Alice's wallet only has possession over two UTXOs, each worth 10 Satoshi. In this case, Alice's wallet will consume both UTXOs the wallet has access to, and creates the appropriate new UTXO worth 15 Satoshi and allocates access of the new UTXO to Bob's wallet, using an address she receives from Bob. When doing this, 5 Satoshis worth of value is still left. This is seen as *change* and gets put into a new UTXO that gets allocated back to Alice. This way, Bob receives his 15 Satoshi, and Alice now has her 5 Satoshi of change back. During the entire operation, all UTXO remained atomically unchanged. UTXO have only been consumed (removed from existence) and created (created back in equal value as the UTXO that have been consumed) A graphical representation of this example can be seen in figure 2.6.

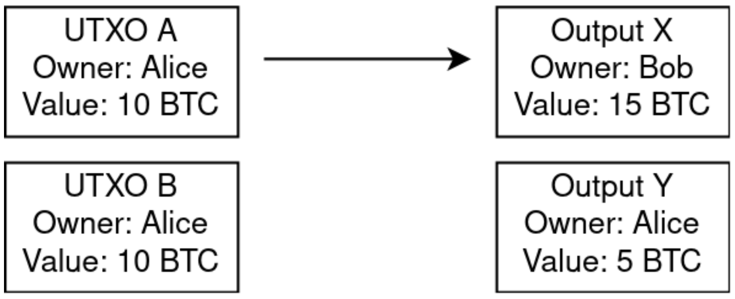


Figure 2.6: Visualisation of a transaction that generates UTXO as change (Output Y) for the transaction sender (Alice). In completion of the transaction, UTXO A and B are removed from existence by being consumed in the transaction input. Output X and Y are newly created.

Locking and unlocking scripts

The previous section explained the general concepts behind the workings of transactions and how they operate to mutate the global UTXO set. This section will try to answer the question on how these transactions are verified and how they are made secure.

As shown in listing 1, a transaction has an array of inputs, called *vin* and an array of outputs called *vout*. We also described in the previous section that a transaction creates new UTXO that can be used as input to transactions in the future. To do this, Bitcoin uses the *transaction validation engine*. This engine relies on two types of scripts to validate transactions: a *locking script* and an *unlocking script* [BMS18].

A locking script is a spending condition that is placed on an output. It specifies the conditions that must be met to spend the output in the future. In simpler words, it "locks" some amount of Bitcoin in the output in such a way that it can only be unlocked by the recipient of the transaction. How this is done will be explained after the next paragraph. Another common name for a locking script is a *scriptPubKey* due to it containing a Bitcoin address, which is a public key. Locking scripts are defined in transaction outputs (vout).

An *unlocking script* is, as the name implies, a script that "unlocks" the lock produced by a *locking script*. It is a script that satisfies the conditions described in a locking script. Unlocking scripts are part of transaction inputs (vin), because they are used to unlock currency locked in the output of a previous transaction to be used as input currency for a new transaction, hence why the currency must be unlocked. A different name to describe an unlocking script is a *scriptSig* or *witness script*. The "Sig" in the term "scriptSig" is a reference to a signature. As we will see in the next paragraph, a signature is the most common way to unlock an output transaction using an unlocking script. However, it must be noted that not all unlocking scripts must contain a signature!

```
{
    "txid": "568da963efc78e67f1c4b8736f25a5436b9
            e25fa6982bcff7c42e75c84e649ee",
    "vout": 1,
    "scriptSig": {
        "asm": "",
        "hex": ""
    },
    "txinwitness": [
        "",
        "30440220514c1e5e21d0c2615c2948bd910d83c63a63839242b88afebb
            0857c8501fa7b102205d3a6dfba48d50bac38f264c260180a07a4d9
            063a800ad3b504cffbed244e9f401",
        "304402202f1b031c1bf27bab44b532016a1767644a8aa4367a978b2b99
            5a6f71a9eedf2302205d8ea0e7a16fb1afc356eca99e4cdccb30cdf
            0efc0706b2808f67a42b81b09a501",
        "52210375e00eb72e29da82b89367947f29ef34afb75e8654f6ea368e0a
            cdfd92976b7c2103a1b26313f430c4b15bb1fdce663207659d8cac7
            49a0e53d70eff01874496feff2103c96d495bfdd5ba4145e3e046fe
            e45e84a8a48ad05bd8dbb395c011a32cf9f88053ae"
    ],
    "sequence": 4294967295
}
```

Listing 2: Example of a transaction input.

Every Bitcoin validating node will validate transactions by executing the locking and unlocking scripts together. Each input contains an unlocking script and refers to a previously existing transaction output (UTXO). The node will copy the unlocking script from this transaction input, then it will retrieve the transaction output (UTXO) the input refers to and copy the locking script from that transaction output. The unlocking and locking scripts are then executed in sequence from left to right. The transaction input is only deemed valid if the unlocking script satisfied the locking script conditions. More on this will follow later, but first, a closer look must be taken at *Script*. This is the language in which locking- and unlocking scripts are written.

Bitcoin Script language In order to further explain how locking and unlocking scripts are used, a closer look must be taken at *Script*. "Script" is the name of the language in which locking and unlocking scripts can be written. Script is a stack-based language. A stack is a data structure used to store data elements in a Last In First Out (LIFO) fashion. A stack can also be defined by its two principal operations. The first operation is the *push* operation, which adds an element to the stack. The second operation is the pop operation, which removes the most recently added element that was not already popped [Knu97].

Script evaluates scripts by executing each element of the script from left to right. Data constants are always pushed onto the stack. Operators can then pop one or more parameters from the stack to act on them, and then push the result of the operation back onto the stack. Some examples of common operators are are the typical arithmetic operators such as "OP_ADD", "OP_SUB", "OP_MUL" and "OP_DIV" which respectively add, subtract, multiply and divide data constants. Besides arithmetic operations Script also supports cryptographic operators such as "OP_SHA256" to calculate a sha256 hash or "OP_CHECKSIG" to check a cryptographic signature. Conditional operators evaluate a condition to produce a boolean result, for example the "OP_EQUAL" operator which checks two constants for equality and then pushes either TRUE or FALSE to the stack depending on the result. A full list of available Script operators can be found on the Script page of the Bitcoin wiki [1].

An important thing to note about Script is that it is not a Turing Complete language. This is because Script does not support loops or conditional flow. The choice to make the language Turing Incomplete is a deliberate one for two reasons. The first reason is to ensure security. Scripts limitations protect it from creating infinite loops or other malicious scripts that could cause a denial-of-service attack against the Bitcoin network. The second reason is to make sure the Script language has predictable execution times. Since there are no complex control structures available, executions times will always be relatively short and predictable.

Evaluation To illustrate how a script written in Script is evaluated, we will look at a simple example. Let's say we write the following simple script:

```
4 2 OP_SUB 2 OP_EQUAL
```

Listing 3: A simple *"Bitcoin Script"* script.

When a script like this is executed, every token is parsed from left to right. As said earlier, when a data constant is encountered it is place onto the stack. When an operator is encountered, it is executed of the latest values it can pop from the stack. In this example, firstly the constants 4 and 2 will be pushed to the stack. At this point, the stack looks as follows:

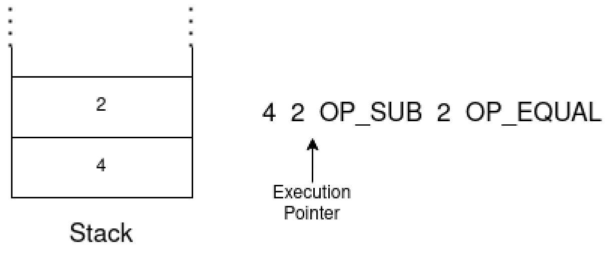


Figure 2.7: Snapshot of the stack during the processing of the script in listing 3.

The next encountered token is the "OP_SUB" operator, this operator pops the two latest elements from the stack and calculates the difference of these constants. The result is then pushed back onto the stack. In the example, the stack would look as follows at this point in the evaluation.

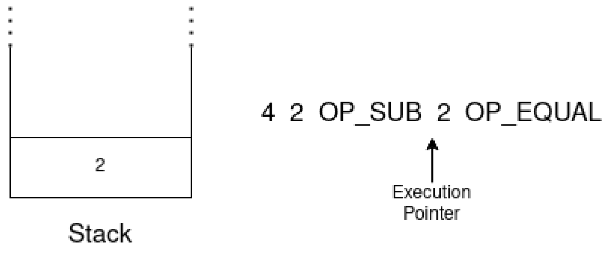


Figure 2.8: Snapshot of the stack during the processing of the script in listing 3.

[1]https://en.bitcoin.it/wiki/Script

After the “OP_SUB” operator a new constant, 2, is seen and pushed onto the stack. This results in the following new stack:

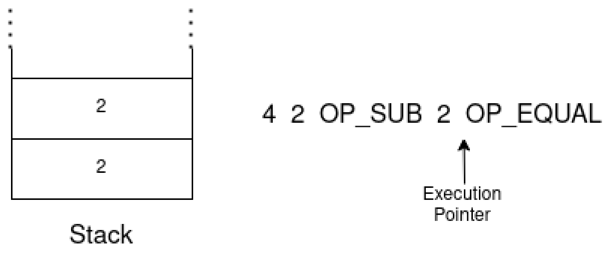


Figure 2.9: Snapshot of the stack during the processing of the script in listing 3.

Lastly, the “OP_EQUAL” operator is seen. This operator pops the latest two elements from the stack and compares them to determine if they are equal. Depending on the equality of the data constants that are compared, this operator will push either a “TRUE” or “FALSE” constant onto the stack. In the case of this example, 2 == 2, hence the operator will push a “TRUE” value onto the stack and the script evaluation is finished, since there are no further constants or operators in the script. A final version of the stack is shown in figure 2.10.

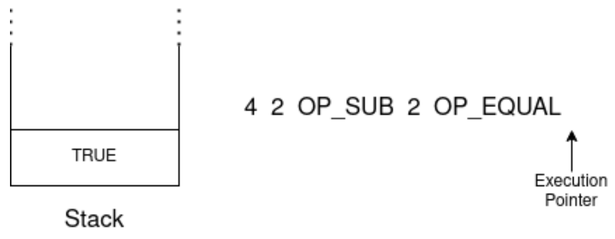


Figure 2.10: Snapshot of the stack during the processing of the script in listing 3.

Pay-to-Public-Key-Hash transactions The most common type of transaction is a Pay-To-Public-Key-Hash (P2PKH) based transaction. A P2PKH script is a type of locking script that locks a transaction output (UTXO) to a public key hash, being a Bitcoin address of the receiver of the transaction. Transaction outputs that are locked by a P2PKH script can be unlocked to be spent by presenting a public key and digital signature created by the corresponding private key. A P2PKH locking script is shown in listing 4.

```
OP_DUP OP_HASH160 <Recipient Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG
```

Listing 4: A P2PKH locking script.

The locking conditions imposed with this script can be unlocked by combining it with an unlocking script as shown in listing 5

Combining the unlocking and locking script leads the following newly constructed script, as shown in listing 6.

When this newly constructed script is executed it will evaluate to TRUE if, and only if, the unlocking script matches the conditions set by the locking script. In this case, the script will evaluate to TRUE if the recipient’s public key in the unlocking script is identical to the recipient’s public key in the unlocking script AND if the unlocking part of the script has a valid signature from the recipients private key that corresponds to the recipients public key.

This shows how transaction outputs can be limited to the ownership of the receiver, since the transactions outputs can only be spent by the owner of the private key that corresponds to the public key that is used in the locking script.

```
<Recipient Signature> <Recipient Public Key>
```

Listing 5: A P2PKH unlocking script.

```
<Recipient Signature> <Recipient Public Key> OP_DUP OP_HASH160
<Recipient Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG
```

Listing 6: A P2PKH locking script combined with an unlocking script.

2.2.3 Blockchain

The previous sections sketched the principles of wallets and transactions, how they are created and how they are verified. This section will describe how transactions can be stored in a historical ledgers, called a *blockchain*.

What is a blockchain?

The term "*blockchain*" has been mentioned several times now in the previous sections of this text. This section will try to bring some clarity in what a blockchain is and what it can be used for.

A blockchain is a data structure representing an ordered, back-linked list of blocks that can hold any type of data. Each block in the chain has a pointer or a link back to the previous block in the chain. It can be conceptually thought of as a vertical stack of blocks, where the first block, also called the genesis block, serves as the bottom one. From here, each new block is stacked on top of the previous block, while also holding a pointer to the previous block, called the *parent block*. The block following a parent block is called the *child block*. The "tip" of the blockchain is used to refer to the latest block in the chain. Of course the tip constantly changes when new blocks get added. Every block, except for the genesis block and the tip, is simultaneously a parent block and a child block.

A block is identified by an identification hash. The hash of a block is calculated by running the SHA256 hashing algorithm twice with the headers of a block as input. When a child block references its parent block, it refers to it by setting its identification hash as one of its own headers. Figure 2.11 shows this visually. Because the parent hash is set as a header, and the collection of headers of a block is used as input to the hashing algorithm that calculates a blocks identification hash, the parent of a given child block is inherently part of the identity of that child block. The identity of a child block would have to change if the parent of the block would change. This key idea is integral to the security of a blockchain.

Suppose a parent block is modified in any way, then this would change its identification hash. Because this has changed, the child block of this block must also update its identification hash, since it is dependant upon the identity of the parent. The "grandchild block" must then also change its identification hash based on the changes in its own parent block, and so on. This effect would cascade throughout the entire blockchain for all generations following the modified block. Executing these recalculations would require significant amounts of computational power, and in turn also energy consumption. By extension, if this were to happen specifically to Bitcoin, these changes would be required to be reproduced in all nodes in the Bitcoin network, since Bitcoin works by distributing the same blockchain to all nodes, counting on the fact that they all have the same copy of the chain. How a blockchain is distributed is discussed in depth in the following section 2.2.4.

Block composition

In the previous section is explained what the key ideas are behind the blockchain data structure. Different cryptocurrencies can implement a blockchain with different properties in its blocks. For Bitcoin, a block is constructed with the following components listed in table 2.1.

It is already revealed that a block contains a header to reference to a block's parent block, and a header to store the block's own identification hash. Beside these headers, blocks can have multiple other headers

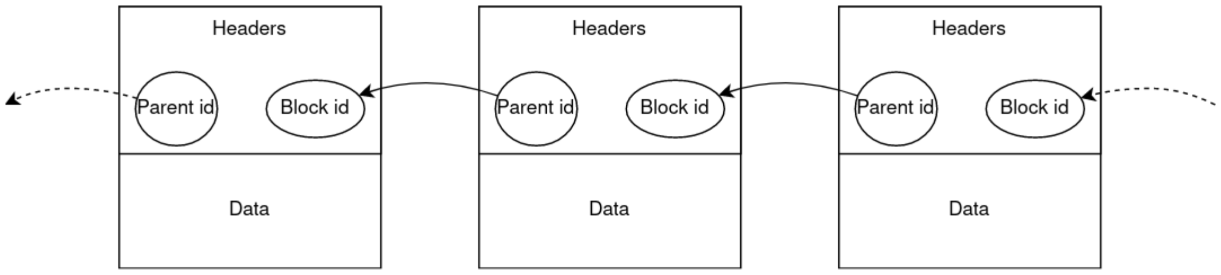


Figure 2.11: Visualisation of a blockchain data structure. Each block has a set of headers and possibly holds some data. A key principle in a blockchain data structure is that each block holds a header referring to its parent block by referencing its block id.

Size	Field	Description
4 Bytes	Block Size	The size of the block following this field, expressed in bytes
80 bytes	Block headers	A collection of block headers
1-9 bytes	Transaction counter	The number of transactions included in the block
Variable	Transactions	The actual transactions recorded in the block

Table 2.1: Table showing the components and the sizes of different block components in a Bitcoin block.

as well, depending on the application at hand. For Bitcoin, all headers and their meaning are listed in table 2.2.

The data section of a block is the reason for a block's existence. In the Bitcoin blockchain, the data section is used to store Bitcoin transactions. The transactions are ordered into a merkle tree, of which the root is stored in the block headers.

Size	Field	Description
4 Bytes	Version	Version number to track software/protocol upgrades
32 bytes	Previous Block Hash	Reference to the parent block
32 bytes	Merkle Root Hash	For efficiency and security reasons, the transactions in a Bitcoin block can be represented as a merkle tree. This header holds the merkle root hash for this tree.
4 bytes	Timestamp	Unix epoch timestamp in seconds when a miner created the block. More on mining will be explained in the section on consensus 2.2.4.
4 bytes	Difficulty Target	The proof-of-work algorithm difficulty target for this block. This will also be further explained in section 2.2.4.
4 bytes	Nonce	A counter used for the proof-of-work algorithm

Table 2.2: Table showing all block headers that are present in the header section of a Bitcoin block.

Identifying blocks

There are two ways to identify a block. The first is the block's identification hash as already talked about in previous paragraphs. However, a block does not contain its own identification hash explicitly inside the block. The identification hash is calculated by using a block's headers section as input to the a double SHA256 hash function.

A second way to identify a block in the blockchain is by referencing its *block height*. Remember when a blockchain was represented conceptually as a stack of blocks in a previous paragraph. The height of a block starting from the bottom block can also be used as a way to uniquely identify a block. The genesis block has height zero. At the time of writing this paragraph in April of 2022, the entire Bitcoin blockchain has a total height of 731702 blocks, meaning the last block in the blockchain, at this time, has block height 731702.

2.2.4 Consensus

The previous section explained what a blockchain is and what the components of an individual block are. This section will go over how new blocks containing new transactions are broadcasted over the Bitcoin network and how they are validated in each node.

Bitcoin network topology

The Bitcoin network is a peer-to-peer network of homogeneous nodes. There exist no coordinating roles and each node keeps a complete copy of the entire history of Bitcoin transactions, stored in a blockchain datastructure. Based on this historical information, each node is able to verify every incoming bit of information on their own, resulting in a minimal amount of trust necessary between the nodes [DW13].

By construction the Bitcoin network topology forms a random graph. When a new node joins the network, the node gets assigned a random set of initial peers that are active in the network at the time of the new node joining the network. Once the new node is connected to its initial peers, the new node can ask his peers about the existence of other nodes of the network and start connecting with them as well. The new node can receive all historical data on the Bitcoin blockchain from its peers and start verifying the ingested information, starting with the *genesis block* and moving its way up to the tip of the chain.

The Bitcoin network topology is a random graph of homogeneous nodes by design. It resembles some of the core characteristics of what Bitcoin tries to achieve. Due to the nature of its network topology Bitcoin is a fully decentralized ecosystem and achieving this can only be done by maintaining a flat consensus network.

Emergent Consensus

Let's now take a step back to see the bigger picture of the design of Bitcoin that is described so far.

Suppose again Alice and Bob want to make a transaction between one another. Alice has installed a Bitcoin node on her computer or laptop, she's setup a wallet with her Bitcoin node and wants to create a transaction to send 0.5 BTC to Bob. Bob opens his wallet on his PC and generates a new asymmetric key pair. Bob's wallet provides him with a hashed version of his public key, which Bob in turn provides to Alice so she can use this to address Bob's wallet as the recipient of her transaction. Alice's wallet software crafts a new transaction that has one of her already owned UTXO of 0.5 BTC in value, unlocks it with an unlocking script, using the correct private key and sets the UTXO as input to the transaction. As output, a new UTXO of 0.5 Bitcoin is created addressed to Bob's wallet address. Alice's wallet then digitally signs the new transaction using her private key. Now Alice wants to broadcast this transaction to all peers of the Bitcoin network so all nodes in the network can update their own local copy of the UTXO set according to the new transaction. When the transaction reaches Bob's wallet, his wallet will pick up that the transaction is allocated to itself and remember the new UTXO in its own UTXO set. However, nothing stops Alice from making ten more of these transactions, perhaps spending more UTXO than she actually owns in her wallet. In fact, any user in the Bitcoin network could create these transactions, however, this would make all transactions meaningless, since everyone can have as much currency as they like. The need arises for some form of consensus in the network. There is a need for a mechanism that can generate agreement between all nodes that a new transaction in the network is valid [Ant17].

But how can all peers in a decentralized network agree on the same transaction ledger, without having to trust other peers? In conventional payment systems trust is placed in a central authority, e.g. a bank, to verify and authorize all transactions. In a decentralized network of homogeneous nodes, there is no central authority present to fulfill this function. Instead, Bitcoin relies on *emergent consensus*. Emergent meaning consensus is not achieved explicitly through agreement by election, neither is there a fixed moment in time when consensus occurs. Consensus "emerges" over time due to how Bitcoin validates new transactions through *mining nodes* or *miners*.

Proof-of-work [Ant17]

So how is emergent consensus implemented in the Bitcoin ecosystem? Suppose again Alice wants to create a transaction to send 0.5 BTC to Bob. Alice again broadcasts the transaction she created to all of her direct peers, who will in turn propagate the transaction to their own direct peers. At some point, the transaction will reach a miner node. The miner will keep collecting transactions until it has enough to craft a new block for the blockchain. “Enough” is a free term, there is no fixed amount of required transactions to be in a block. The amount of transactions that are included in a block is up to the configuration of the miner node. Once the miner decides to create a block, it forms a merkle tree of all transactions in the new block, calculates the merkle root of this tree and stores it in the block headers, among the other headers as well. An important notice is that when any transaction in the merkle tree changes, the root of the tree also changes. This means the validity of the entire transaction tree can be verified with the merkle root of this tree.

At this point, the miner has to calculate the new block’s identification hash. By design, in the Bitcoin system, the identification hash is calculated by hashing the collection of headers of the block twice over with the SHA256 hashing algorithm (Equation 2.1).

$$identificationHash = SHA256(SHA256(Headers)) \tag{2.1}$$

However, the hash that is calculated also has to satisfy a specific requirement: the hash has to have a fixed number of leading zeroes. For example, the block at block height 731832 has identification hash *00000000000000000009f758318113ce7aa03bfa587124695e9eefb17252a17f*, which is required to have nineteen leading zeroes. The amount of leading zeroes required for a new block is determined by the *difficulty field* of the block’s headers, as explained in table 2.2. The miner can influence the amount of leading zeroes in the identification hash through the alteration of two parameters the miner has free control over. The first free parameter is the *nonce header*, which was mentioned in table 2.2. The nonce header is a 4 bytes header that can be given any arbitrary value. Changing the value in this field directly influences the resulting identification hash of the block. However, updating the nonce header alone will not be sufficient in order to obtain a block hash with a fixed amount of leading zeroes. This is why the miner also tries the influence another parameter it has free control over: *the merkle root* of the merkle tree structurally stores the transactions in the block. A miner is not allowed to alter any transactions in the block, but it has freedom over how it chooses to organize the merkle tree, which directly influences the resulting merkle root hash, which in turn influences the block hash. Secondly, the miner can influence the merkle root through adding additional data to the coinbase transaction. The coinbase transaction is the one transaction the miner can include in the block themselves in order to assign themselves the block reward, minting new Bitcoin. Since the coinbase transaction has no need for a transaction input, since it mints new unexisting coins, the transaction input can essentially hold any arbitrary data. The miner can use this free data to influence the coinbase transaction hash, which in turn again influences the merkle tree root and then the entire block hash. This mining mechanism is typical to Bitcoin and is called the *proof-of-work*, since some work has to be done to calculate the correct hash, while other nodes can easily verify the results.

The reason the proof-of-work mechanism exists is trifold [Ant17]. The first reason is that it functions as a way to fairly bring new BTC into circulation. Since calculating an identification hash in the proof-of-work model is both computationally and energy consumption intensive, a miner will be rewarded for what it does. When a miner creates a block, it is allows to add a *coinbase* transaction to the collection of transactions in the block. A coinbase transaction is a special type of transaction that allocates BTC to an address owned by the miner. It has no existing UTXO as input, only an UTXO output addressed to the miner. The amount of BTC a miner receives for mining a block halves over time. This is to limit the amount of BTC that are able to come into circulation.

The second argument for proof-of-work is security. Since calculating an identification hash is a computationally heavy operation, it also provides security. If an attack would want to change the contents of a given block in the blockchain, they would have to recalculate the identification hash of the altered block, as well as all blocks following it. The older the altered block is, the larger the resources would have to be to rewrite and distribute the entire blockchain. This makes it practically impossible to perform this type of attack, and warrants the integrity of the blockchain.

The third reason is to control the flow of newly created blocks. The Bitcoin protocol aims to create a new block once every 10 minutes. Over time, computational resources become cheaper and faster, meaning the rate at which a block can be mined also speeds up. To keep the rate of newly mined blocks consistent over time, the difficulty of mining a block is increased over time. The difficulty to mine a block can be simply done by adding the amount of required leading zeroes in a block’s identification hash.

Synchronizing the blockchain across nodes

Once the miner is done creating the new block and has found a correct identification hash for it, it broadcasts the new block to all of its direct peers in the network. These peers will then add the new block to the tip of their local copy of the blockchain, and propagate the new block further to their own direct peers. These peers do the same again and again, until the new block has propagated throughout the entire Bitcoin network. This is what would happen in an ideal situation.

In practice it might occur that two or more miners have mined a block intended for the same block height at the same time. When this happens, nodes that receive these blocks will temporarily keep two variations of the blockchain, both having one of the different blocks as the new tip. A situation like this is called a *fork*. When more new blocks reach the node, they will also be added to the version of the blockchain they fit to. Eventually, one of the local copies of the blockchain will be the longest. A node will always pick the longest copy of the blockchain it has as the legitimate one. Blocks in the parallel chain will be dropped and thus do not become part of the blockchain. These dropped blocks are also called *orphan blocks*. A visual representation of this situation is made in figure 2.12.

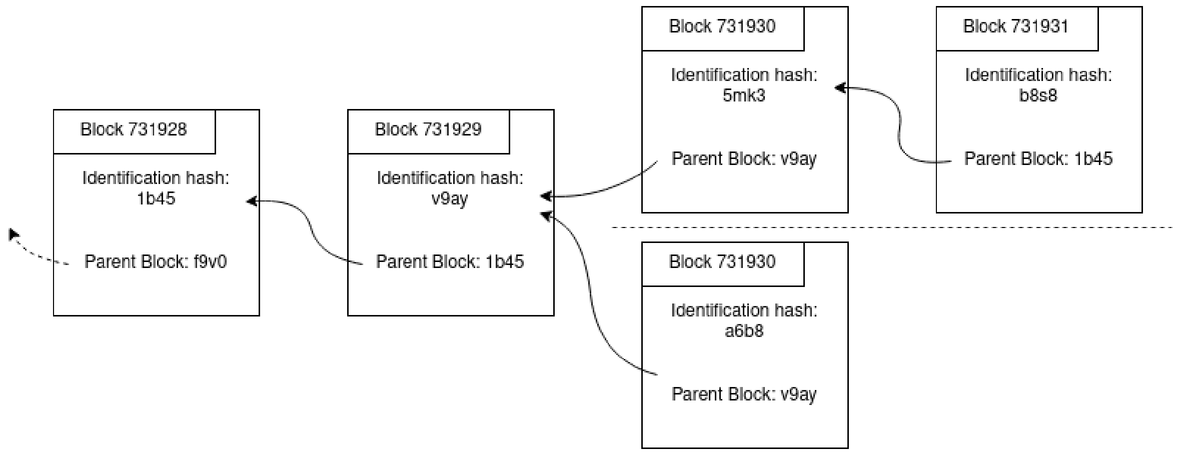


Figure 2.12: Visual example of a fork happening in a blockchain when a node in the network receives multiple different copies of blocks that are supposed to go in the same place. A node will keep two or more copies of the chain and eventually continue on the longest version of the chain. Blocks from the other chains will be dropped. These are then orphan blocks (Block 731930 in this example).

Since blocks in a shorter fork of the blockchain have the possibility to get dropped, this also means all transactions in the orphaned blocks are dropped. For users of the Bitcoin network this means that when a new transaction is created, it does not necessarily mean the transaction will become part of the blockchain with absolute certainty. To measure the certainty of a block being part of the blockchain, the amount of confirmations on the given block can be counted. A confirmation of a block is the sum of all blocks following the given block in the blockchain. The greater the amount of confirmations, the greater the possibility that the block will remain part of the blockchain. Wallets can also keep track of the amount of confirmations on the blocks their transactions are a part of. Users can use this as a metric to consider their transactions as "completed" or not. In general, six confirmations is considered to be safe and secure enough to prove a transaction will be permanent [2]. The probability of a transaction with six confirmations still being dropped is less than 0.1% [bit].

2.2.5 Anonymity in the Bitcoin ecosystem

The previous subsections in this section have all helped to give a fundamental look into the technical aspects of Bitcoin. This subsection will try to make use of this knowledge and give more insight into what levels of anonymity can be expected in the Bitcoin ecosystem. This is an important topic to cover since anonymity is one of the key reasons malware attackers use Bitcoin as a form of payment instead of fiat currency or other, better traceable payment methods.

Bitcoin has a reputation for being an anonymous, untraceable form of payment. However, this reputation is not at all justified. The wrong perception that Bitcoin is anonymous stems from the way Bitcoin wallets

[2]https://bitcoins.net/faq/bitcoin-confirmations

work. As explained, a wallet is a piece of software that a Bitcoin node can run which manages a collection of asymmetric key pairs. When a user is involved in a transaction, instead of being addressed through the user's real name, they can be addressed by the addresses in their wallet. However, the transactions these addresses are involved in are completely out in the open. By design, they are stored in the blockchain of which each node in the network has a copy. Anyone can track the movement of currency for a given address [OSu22]. The general idea that Bitcoin is anonymous is wrong. Bitcoin is rather a *pseudonymous* form of payment.

Besides tracing transactions, there is another issue as well. If a user of the Bitcoin network is not a miner, and they don't own any Bitcoin, chances are they will buy their first BTC at one of the many exchange services. At an exchange service it is possible to exchange fiat currency for cryptocurrency. In many countries, crypto exchange services are required to implement the *know-your-customer (KYC)* principle, just like regular banks. Implementing this principle means that an exchange service has to require some proof of identification from their customers, mostly in the form of a government ID card. Since exchanges know the users identity, they can also link wallets operated by their customers to real world identities. This combined with the fact that transactions can also easily be traced removes a significant portion of the privacy and anonymity that is often wrongly associated with Bitcoin [OSu22].

To counter these flaws in Bitcoins anonymity, efforts have been made to increase its anonymity and reduce the traceability of transactions.

Address reuse and stealth addresses

Something all users in the Bitcoin ecosystem can do to increase their privacy is to avoid reusing addresses to receive payments on. Ideally, for each new incoming transaction into a Bitcoin wallet, it would be advisable to create a completely new private/public key pair and use this new public key as the receiving address of a new transaction. Although this is not completely waterproof, avoiding reuse of addresses limits the possibility of associating all addresses to the same wallet can help increase a wallet's privacy [Deva]. When an address is used exclusively as a one time address, it can also be called a *stealth address*[3]. [Fra22].

In the context of ransomware, stealth addresses are especially useful. If a ransomware family creates a stealth address for each victim to send its payment to, it can use this as a mechanism to identify which victims have paid their ransom. Besides helping a ransomware family on an organisation level, this also makes it harder to associate other unknown addresses in the same ransomware family to the known addresses.

Mixing Services

Another approach Bitcoin users can resort to to increase their own privacy is the use of *mixing services*. Mixing services are based on creating CoinJoin transactions, which will be further elaborated on in the next chapter in section 3.2.2. The legality of using mixing services depends on the jurisdiction where the person using the service lives. Besides this, mixing services also require the user to trust the operator of the mixing service to not steal the funds or log the transactions the user has requested, which would make the idea of decreasing traceability meaningless.

[3]A note to make here is that the term "stealth address" differs slightly from the same term that will be used later within the context of Monero. The appropriate definition in the context of Monero is given in chapter 5

Chapter 3

Studying Ransomware Families in the Bitcoin Ecosystem

Based on the understanding of Bitcoin technology as described in chapter 2, this chapter will explore how this knowledge can be used to trace ransomware related payment across the Bitcoin blockchain. We do this in order to gain insight into the organization and inner workings of ransomware families. Existing research has already been done in [Hua+18], [PHD19] and [BPS18]. The general steps taken to trace ransomware payments as described in each of these research papers are generally the same, sometimes using different tooling between the different research papers.

3.1 Collecting seed addresses

The most straightforward starting points to start a ransomware payment trace are most likely the addresses in the *seed address set*. A *seed address* is an address that is presented to a victim of a ransomware infection. The purpose of the seed address is for the victim to deposit a ransom into the address in order to receive a decryption tool of some kind from the ransomware attackers. These addresses are hashes of public keys associated with the wallet(s) of ransomware attackers, as was also discussed in chapter 2. There are two primary ways to obtain a collection of initial seed addresses: consulting ransom notes from *real victims*, or consulting ransom notes from *synthetic victims*.

3.1.1 Real victims

Real victims are, as the name suggests, people or organisations that have fallen victim to a ransomware infection. Seed addresses presented to real victims can possibly be scraped from reports of ransomware infections on online forums. These reports can contain screenshots of ransomware dialogue screens or ransom notes. Scouring the web for real ransom notes can be a legitimate way of obtaining seed addresses but they might be hard to come by, or simply impossible, as was the case in [Hua+18] for Locky and Cerber ransomware clusters.

An additional difficulty that I stumbled upon when trying to use this method is that many blogs or news outlets tend to hide specifically the seed addresses shown in these ransom notes. The reason for these outlets to do this is possibly to avoid giving these malicious addresses more exposure than necessary, or to protect the victims these seed addresses belonged to. This would be an understandable argument, but from a research perspective this was a minor annoyance.

3.1.2 Synthetic victims

An alternative to scouring the web for real victims is to deliberately create synthetic victims. These are created by installing known ransomware binaries on air gapped devices. After these devices are manually infected, possibly new, previously unknown seed addresses can be read from the ransom notes.

Legitimate ransomware binaries can be found on services such as VirusTotal, virusshare.net or malware-traffic-analysis.net. Services like these might also own different versions or iterations of a specific ransomware binary which might also produce a variety of different seed addresses, but this is highly dependant of the ransomware family and how it operates.

A side note that has to be made here is that acquiring seed addresses this way becomes more difficult when the ransomware binary that is being used is vendor or platform specific. Many modern day ransomware attacks are exploiting vulnerabilities in well known operating systems such as Microsoft Windows in the end-user space and Linux based operating systems in the server space. Less popular ransomware binaries could also be targeted at proprietary operating systems or vulnerabilities in vendor-specific platforms. If this is the case, the conditions necessary to run these binaries *could* be harder to reproduce, depending of the age of the ransomware binary, the configuration of the device, the version of the operating system, etc.

Another point of discussion using this method could be of a more ethical kind. In many cases, ransomware families choose to operate in a way that every time a new victim is infected, a new seed address is generated specifically and uniquely for the new victim. If a seed address is created for a synthetic victim, the address will not become "active" in the ransomware family until a ransom has actually been paid to the address. Using this method with a ransomware family that operates in this way would require paying the ransom for every new address, since this is costly and directly promotes the existence of ransomware even more, I chose not to take this approach.

3.1.3 Third party data

A third alternative to gather seed addresses is to use sets of addresses already discovered in previous research or other third party sources. These sets of addresses could then possibly be extended by clustering them with unknown addresses, as will also be described in the next section. This way, a solid starting set of seed addresses can be built fairly quickly once a clustering implementation is developed.

3.1.4 Searching existing transactions

A fourth and final method is one that has not yet been proposed in existing research. If multiple characteristics of a ransom payment are known beforehand, previous transactions in the history of Bitcoin can be checked to see if they match these same characteristic. Typical characteristics that can be known beforehand are e.g. the ransom sum that is requested, as well as the window of time in which an attack was deployed. If it is known, for example, that a ransomware attack has happened in the first week of February, and every victim was demanded a sum of 0.055 BTC, all Bitcoin transactions since the beginning up until the current day can be scanned for this ransom amount. This method does not guarantee to produce one hundred percent true positive transactions, but depending on the amount of criteria that are known about the attack, it could be used as a heuristic to narrow down the set of possible ransom transactions, and thus the possible set of seed addresses.

A way to strengthen the viability of detecting true positive transactions could be to take all receiving addresses in the discovered transactions and try to cluster them together using the multiple-input spending heuristic. If some of the potential seed addresses can in fact be clustered together, this strengthens the claim that they are seed addresses of the same ransomware family. The multiple-input spending heuristic will be explained in detail in the following section 3.2.

3.2 Seed address clustering

The size of the seed address dataset that can be collected by studying real and/or synthetic victims is usually quite small. Think in the order of around 100 addresses per ransomware cluster. For some types of ransomware, this is not really a problem. For example, reports on the WannaCry virus (sometimes also called WannaCrypt, WannaCrypt0r 2.0 or Wanna Decryptor) that hit multiple victims worldwide in May of 2017 suggest that victims from multiple different infections are asked to all deposit Bitcoin into the same seed addresses [Kan17]. This suggests that WannaCry attackers by design do not manage a lot of different wallet addresses to receive bitcoins on, and that the seed addresses that are available form a sufficiently representative dataset.

Other families of ransomware, such as Locky, Cerber and Spora, work differently in collecting ransoms. These ransomware families generate a unique new RSA key pair in the wallet for every new victim. [Lab21]. The reason for these families to do this could be for the attackers to easily discern victims. This helps in keeping track of which victims paid their ransom and which did not. This way the victim is not required to prove they paid the ransom after their transaction is "completed", since this payment checking can be done at the attacker's side by checking incoming transactions on the uniquely designated victim's seed address. As mentioned previously in this section, the WannaCry ransomware family did not operate this way, which leads to the assumption that WannaCry attackers were also not really intending to release all of the victims' files after a ransom was paid, since they did not implement a robust payment checking mechanism.

The fact that some ransomware families generate new unique seed addresses for individual victims motivates the need to find ways to expand on a small, existing collection of seed addresses. Luckily, this can be done by *clustering* known addresses with unknown addresses on the blockchain, using the *multiple-input clustering heuristic*, or sometimes also called the *co-spending heuristic* [PHD19].

3.2.1 Multiple-input spending heuristic

The multiple-input spending heuristic is built on the *common-input-ownership heuristic* [Nak08], [22c]. This is a heuristic or assumption that *all inputs going into a transaction must be owned by the same entity or wallet.* This concept was already briefly discussed in chapter 1 in the section about transactions. The heuristic usually arises if a user's wallet does not have a single UTXO large enough to carry out a payment requiring a specific payment amount. In this case the user's wallet will typically use multiple UTXO owned by that same wallet as input to the transaction. The combined value of these UTXO's are usually larger than the actual required amount of Bitcoin that must be paid. In this case, the transaction will have an additional output that is meant to return the change to the sender's wallet by creating a new UTXO having the value of the change.

An example of the common-input-ownership heuristic can be found in listing 3.1:

```
A (2 BTC) -> X (5 BTC)
B (2 BTC)    Y (1 BTC)
C (2 BTC)
```

Listing 3.1: This listing shows an example to illustrate the common-input-ownership heuristic. Suppose a payment has to be made with a payment amount of 5 BTC. The sender of the payment owns a total of 6 BTC, which is the combined value of all the three UTXOs the sender owns. Suppose each UTXO of the sender is worth 2 BTC each. When making the payment, a transaction is created by the sender's wallet that uses all of the UTXOs it already owns, with addresses A, B and C, as inputs and creates two new UTXOs as output. The output UTXOs are respectively addressed to X, which is a wallet address of the receiving party, and Y which is a wallet address that is owned by the senders wallet in order to retrieve the 1 BTC worth of change. In this example, we can use the common-input-ownership heuristic to assume addresses A, B and C are owned by the same party. In this example, it could also be assumed Y belongs to the same wallet, but when studying transactions in the wild, this could not be claimed with certainty if Y is not the same address as any of the input addresses. It could then belong to virtually anybody.

The idea explained in the example above can in theory be applied to all regular payment transactions occurring in the Bitcoin transaction history. The idea can thus also be used to cluster the already known seed addresses extracted from ransomware victims or third parties. To apply this idea, an algorithm can be written that iterates through all known ransomware seed addresses and looks up all the transactions that the address occurs in as an input. If an address occurs as input to some existing transaction, we can assume that all other addresses used as inputs for the same transaction are also owned by the same ransomware family as the seed address. When retrieving new addresses by using this clustering strategy the new addresses can recursively be scanned the same way to expand the seed address set even further. The algorithm can halt when no new addresses can be added to the total seed address.

An important note to make is that common-input-ownership heuristic is still only a heuristic. In this case, this means that while clustering using this heuristic allows the detection of previously unknown wallet addresses associated with the ransomware attackers, it is not an exhaustive approach. It is not guaranteed to find *all* of the existing addresses associated with the original seed address set, since the attacker's wallets may also own addresses in clusters that are disjoint from the ones that are detecting by using the clustering strategy that is described.

A second pitfall when using this heuristic is the existence of *CoinJoin transactions*. This will be discussed in the next section.

3.2.2 CoinJoin transactions

Bitcoin is often wrongly promoted as a private and anonymous payment system. However, upon closer look, Bitcoin is at best only a pseudonymous payment system. Bitcoin suffers from the *linking problem* [Zag+20]. The linking problem states that, despite Bitcoin users being identified by a random wallet address, they can still sometimes be linked to real identities. For example, some blogs or websites could ask for donations through Bitcoin. In this case, the identity (being the blog or website) is immediately linked to the corresponding wallet address. Then, by clustering using the multiple-input-ownership heuristic, other wallet addresses associated to the donation address can also be discovered. This can jeopardize the anonymity of the blog or website within the Bitcoin network, since all transactions using the discovered address also expose information about the value of the website or blog's wallet, and the addresses they transfer money to.

Since Bitcoin is thus not inherently private or anonymous, mechanisms have been developed that improve the anonymity of the system. CoinJoin transactions are one of those mechanisms.

Defining CoinJoin transactions

CoinJoin is a trustless method for combining multiple input transactions from multiple spenders into a single transaction to make it more difficult for outside parties to determine which spender paid which recipient or recipients [22b].

An example of how this would be used in practice is the following. Assume user A wants to make a payment to user B, user C wants to make a payment to user D, and user E wants to make a payment to user F. When making these transactions without CoinJoin, each payment would have its own separate transaction, meaning that each individual transaction would be out in the open, and thus also not private. This situation is depicted in figure 3.1.

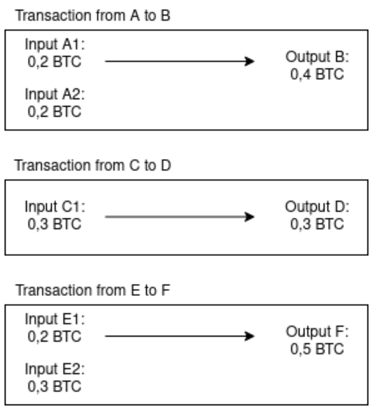


Figure 3.1

When making a CoinJoin transaction, owners of transaction inputs (UTXO) allow their transaction inputs to be combined with transaction inputs of others. By doing this a new transaction is created with inputs from several different users to make payments to several different receivers. When a transaction like this appears in the Bitcoin transaction history, it is much harder to figure out which users provided the inputs and to whom they made a payment. The CoinJoin transaction that can be created from the example in figure 3.1 can be seen in figure 3.2.

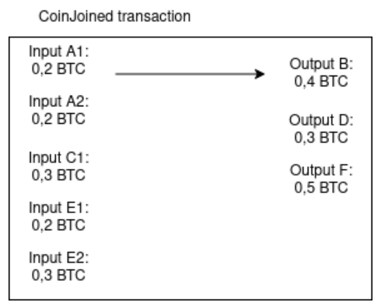


Figure 3.2: the transactions as shown in figure 3.1 combined into one CoinJoin transaction

CoinJoin transactions are not natively supported in the Bitcoin system. In order to create them, coordination is needed among all users providing the transaction inputs. Typically, a coordinator will collect all the necessary input and output information from the participating users [22a] . They announce the transaction inputs and outputs they would like to be included in the CoinJoin transaction. The coordinator will then use this information to craft the new transaction and allow every participant in the transaction to sign their transaction inputs away. The transaction can then not be modified anymore without becoming invalid. This prevents the coordinator from claiming all the funds for themselves.

Defeating CoinJoin transactions

Now that it is explained what CoinJoin transactions are and how they are used, it is clear that they can also be used within the context of operating a ransomware cluster. CoinJoin transactions can be used by ransomware operators to obfuscate where and how they move the ransoms that they collect. Another problem CoinJoin transactions introduce is that there cannot be a one hundred percent certainty that a newly discovered address using the multiple-input spending heuristic actually belongs to the same ransomware operators. Due to the implications CoinJoin transactions have on the ability to analyse ransom transactions, it would be useful to be able to recognize these transactions when they occur. Several heuristics to try and detect these transactions have been proposed in the past.

In [Gol+18] it is suggested that *JoinMarket* is the most popular service to coördinate the creation of CoinJoin transactions. At least in 2017, when their research was officially published. In their research, they propose an algorithm to specifically identify JoinMarket transactions. The algorithm they propose is a series of heuristics to filter transactions based on several different observations that are specific to JoinMarket transactions.

Similarly, in [Sto+21], Stockinger et al. study how CoinJoin transactions created with the help of *Wasabi* and *Samourai* wallets can be detected using two novel heuristics they propose. The heuristics they propose are the following:

- *Wasabi CoinJoin Detection Heuristic*: If a transaction t has at least ten equal value outputs, with 0.1 +- 0.02BTC being the most frequent one, and if it has at least three distinct output values with at least one being unique, and if it features at least as many inputs as occurrences of the most frequent input, then t is a Wasabi Wallet CoinJoin transaction.
- *Samourai CoinJoin Detection Heuristic*: If a transaction t has exactly five uniform outputs that equal p BTC and if it has precisely five inputs, with at least one and at most three equal p BTC, while the remaining two to four inputs are between p an p + 0.0011 BTC, then t is a Samourai Whirlpool CoinJoin transaction.

They also disclose the effectiveness of their heuristics. For the Wasabi CoinJoin Detection Heuristic, they state the heuristic yields a precision of 0.824, a recall of 0.999, and an F1 score of 0.899. Precision, recall and F1-score are performance metrics typically used within the field of pattern recognition, classification problems and, in this case, information retrieval. *Precision* is the fraction of relevant instances among all received instances. *Recall*, also called the sensitivity, is the fraction of relevant instances that are retrieved from all ground truth relevant instances. The F1 score is calculated using both the precision and recall and expresses the accuracy of a retrieval method in a single score ranging between 0 and 1.

For the Samourai Whirlpool Detection Heuristic no specific numbers were disclosed. This was partly due to Stockinger et al. not having access to a strong, significant ground-truth set of Samourai Whirlpool CoinJoin transactions to measure the performance of their heuristic against.

A third, more general CoinJoin detection approach is the one proposed in [Wu+20] by Wu et al. In their research, a feature detection framework-based analysis is provided that is used to identify statistical properties of transactions made by mixing services. With it, a machine learning model is trained that is able to detect CoinJoin transactions made by mixer services. Their method proves to be successful in detecting CoinJoin transactions. However, it must also be noted that a concern with this approach is that mixing service providers can update their mechanisms to eliminate the typical behaviour that identifies their transactions. Another critical issue is that the CoinJoin transactions that get detected by their method all belong to mainly three of the same mixer service providers: *Bitcoin Fog*, *BitLaunder* and *Helix* (which has been offline since 2017).

Difficulties in detecting CoinJoin transactions

It is clear from the research examples mentioned above, and several others as well, that the detection of CoinJoin transactions is being actively researched. The existing research exposes two of the primary difficulties in battling anonymisation through CoinJoin transactions:

1. *CoinJoin detection techniques are mostly successful when they are designed to detect transactions forged by specific mixer service providers.* As far as current research in this area goes, all successful detection techniques have been designed to only detect very specific transactions that are created by one mixer service provider, or a group of very similar mixer service providers. This seems evident since each mixer service provider will leave behind a specific "fingerprint" (a set of features) on each of their transactions. Once these fingerprints are discovered a heuristic can be designed to test existing transactions for these fingerprints.

2. *Research is often one step behind on the state-of-the-art.* While most research related to the area of CoinJoin detection is successful in what they do, since it is built on the idea of recognizing CoinJoin transactions based on specific mixer fingerprints, it is often one step behind the latest versions of mixer providers or non-existent mixer service providers at the time the research is done. Mixer providers also gain insight into the vulnerabilities of their services and will improve upon on them, or in some case discontinue their service.

These two difficulties make it challenging to design a one-size-fits-all approach to detect CoinJoin transactions. The list of available mixer service providers is continually evolving. Old mixer services get updated to increase their anonymity in response to the research done about them, and new and improved mixing services are created each year. In the context of this thesis, which partly relies on the ability to trace ransom payments end-to-end throughout the Bitcoin blockchain, it does not make much sense to implement one or more of the detection strategies discussed above. The main idea of the implementation part made for this thesis is to be able to quickly detect and inspect the impact of newly occurring ransomware attacks. Since mixing techniques are mostly one step ahead of the detection techniques, no big improvement can be made from it, while it would instead add a bigger computational, mostly unnecessary, load on the analytics implementation. However, detecting CoinJoin transactions would still be a large benefit in the analytics process. Because of this, a CoinJoin detection analysis should be performed manually by using one of the methods described in existing research for some of the most popular mixer services provided at the time of performing the analysis on the ransomware impact. This would produce the most accurate results possible. It must also be noted again that CoinJoin detection techniques until the writing of this thesis have always been heuristics, not algorithms. This means that the output they produce is never complete or one hundred percent accurate. Not every transaction that is analysed can be considered as a true CoinJoin transaction or a true "legitimate" transaction.

3.3 Representing transactions in a graph

Once a set of seed addresses is obtained, all addresses and related transactions can be represented as a graph. The translation of address and transaction data to a graph is natural, since the entire history of Bitcoin can be viewed as a large and continually growing transaction graph. The graph that is built with the set of seed addresses will then be a direct sub graph of the complete Bitcoin transaction graph.

Creating a graph representation also allows the use of graph algorithms to extract useful information by the use of a graph query language. More in depth information on how this is practically done is explained in the following chapter 4.

3.3.1 Representation types

When representing Bitcoin transaction in a graph, a choice has to be made on the representation format of the graph. This section will discuss those possibilities.

Address Graph

The first representation type is to represent the transaction history is an *address graph.* In this graph type, each vertex represents an address and each transaction between addresses represents an edge. Transactions are represented as directed edges. A visual example of this graph representation is given in figure 3.3.

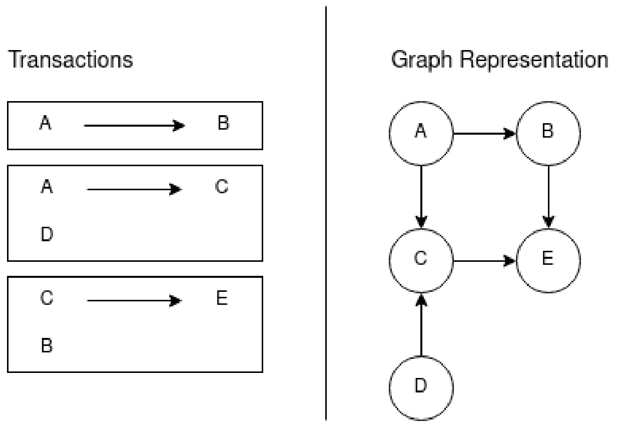


Figure 3.3: An example of an address graph representation. The transactions on the left are represented in a graph on the right. Vertices represent Bitcoin addresses, edges represent transactions between addresses.

Cluster Graph

A second graph type is the cluster graph. In this graph type, each vertex represents a cluster of addresses that are likely to be controlled by the same owner. These clusters can be calculated using the multiple-input spending heuristic as described earlier in section 3.2.1. The directed edges between vertices represent transactions made between the clusters.

Transaction graph

A final graph type is the transaction graph. In this graph type, there exist two types of vertices. Both addresses and transactions are represented by vertices in this graph type. Directed edges between vertices also come in two types: vin edges and vout edges. Vin edges represent transaction inputs and are drawn as a directed edge from an address to a transaction. Vout edges represent transaction outputs and are drawn from transactions to addresses.

3.3.2 Mapping other data related to ransomware families

Throughout this chapter, 2 key features in the mapping of ransomware families to a local data structure have been discussed. A clustering approach has been discussed that can be used to expand a given set

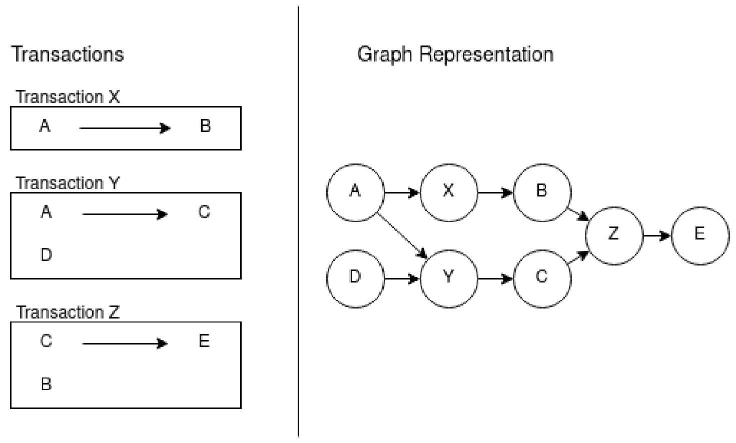


Figure 3.4: An example of a transaction graph representation. The transactions on the left are represented in a graph on the right. Vertices represent Bitcoin addresses and transactions, edges represent transaction inputs and transaction outputs.

of ransomware seed addresses to a possibly much larger collection of seed addresses. Secondly, a graph representation of the collected seed addresses is discussed. Both of these concepts form the foundation of any insights that can be gained into a given ransomware family. In the next chapter, a practical approach is taken in the form of the development of a tool specifically created to fulfil these tasks and to try and gain additional insights into ransomware families as a whole.

Chapter 4

TRaP

In this chapter, a new command line application, TRaP (Tracer of Ransomware Payments), is presented. The incentive behind the development of this application is to create a single tool that groups all the procedures that can be used to quickly gather information on the impact of newly occurring ransomware attacks. The information that can be automatically gathered and calculated using the application can then be used to make further analysis of a new ransomware attack easier.

4.1 Motivation

With the ever growing interest into the world of cryptocurrencies, the need for cryptoasset analysis tools is also on the rise. At the moment, cryptoanalysts can choose from two main options. The first is using commercial services. These crypto analysis services are often provided through the use of APIs, for which a paid API key is required. The upside to this approach is the wide availability of tools and APIs to pick from. The downside is the cost. Depending on what API is used and what provider is providing the service, service costs can greatly vary.

4.1.1 Open-source solutions

Online Block Explorers

On the other hand, free and open-source cryptoasset analytics tools are also, albeit more sparsely, available. One can resort to some of the freely available online block explorers. Examples of these online explorers are blockstream.info and blockchain.info, but several others exist that offer the same basic block exploring functionalities. Some of these free block explorers may be rate limited to a number of requests per hour, day or month. They might also have a paid tier where the rate limit is increased or completely removed.

Hosting a Bitcoin full node

Beside using readily available block explorer APIs online, one can also resort to hosting their own blockchain datastore and consulting it locally. The most rudimentary way to do this is to host a Bitcoin full node. A full node is a program that fully validates transactions and blocks locally. To validate new and incoming blocks, the entire history of the blockchain also needs to be downloaded locally and sequentially verified. Besides operating as a validating node in the Bitcoin blockchain, a full node keeps a full history of blocks in the blockchain since the genesis blocks. This means it stores all historic blockchain data locally. A full node also offers the option to host a *JSON-RPC server* over HTTP that can be used to query the full node for information on blocks, transactions and addresses. This is helpful in many situations, however, it has its limitations. Since a full node can also be used as a Bitcoin wallet, it only tracks extensive information about addresses it owns. This means that it cannot provide a transaction history or an UTXO balance of a completely unrelated Bitcoin address to the full node wallet. In order to retrieve this information with the readily available RPC calls, a lot of processing would have to be done each time this information is requested, since large parts of the Bitcoin transaction history would have to be parsed to retrieve and calculate the requested information. To put this to scale, the Bitcoin blockchain, at the time of writing this chapter, is 393GB in size. To retrieve the transaction history of any given Bitcoin address, the entire history of 393GB would have to be parsed to give a conclusive result. This would not be a sensible approach to calculate the query responses every time. A better approach would be to parse the entire Bitcoin history once, from start to finish, and continually keep track of address information and transaction histories along the way and store them in a database. Luckily, several open source implementations exist that do just that. These implementations are sometimes also used to serve as a backend for one of the many available block explorers online.

Bitcore

Several open source implementations of block explorers exist. A popular open-source implementation is Bitcore [Bit22]. As stated in the Bitcore GitHub repository: Bitcore is infrastructure to build Bitcoin and blockchain-based applications for the next generation of financial technology and is developed by Bitpay, Inc. Bitcore envelops several services: a Bitcoin full node, built around the reference implementation of Bitcoin Core [1], a wallet client- and server implementation, and *"Insight"*. Insight is a blockchain explorer web interface. In order to run Bitcore, a MongoDB server is required to store data, as well a "make", "g++" and "gcc" to build and run the source code. When setting up Bitcore, it comes with its own Bitcoin full node installed. This full node is then queried through its JSON-RPC server to build a database of blocks, addresses, transactions, etc.

ElectrumX

A second open-source alternative is to run an *ElectrumX* server [NJ22]. An ElectrumX server is part of the ElectrumX wallet system. ElectrumX is designed to serve as a Bitcoin wallet first. To do this, the ecosystem primarily exists of two main parts, the ElectrumX client, which performs all the wallet related operations, and the ElectrumX server, which the client uses to track information about Bitcoin addresses. The server indexes and stores information about Bitcoin transactions, addresses and their transaction history. The information is stored in a *RocksDB* [Fac22] database, developed and maintained by Facebook, Inc (now renamed to Meta, Inc). An ElectrumX server can also be used as a standalone server without the need for a wallet client. In this case, the server can be queried for information on any block, address, transaction and more. The API is reachable over HTTP.

GraphSense

Servers such as the Bitcore Insight app and the ElectrumX server are excellent providers of Bitcoin related data. However, they are not yet complete cryptoasset analytic tools. They only provide data, but no other additional functionality that enables more advanced types of analysis. This is where GraphSense comes in [Has+21]. GraphSense is a complete and open-source cryptoasset analytics platform. Like the several commercial and web-based solutions, GraphSense also provides a dashboard for interactive investigations. Where GraphSense creates value over other solutions is that it builds different pre-computed graph abstractions of the Bitcoin blockchain. These pre-computed graphs reflect the main structural elements of a cryptoasset ecosystem: *actors*, who own *cryptoassets* and interact amongst each other through cryptoasset *transactions*. GraphSense discerns three types of graph representations:

- *Address graph*: In this graph type, each address is represented by a vertex, and each aggregated set of transactions between two addresses is represented as an edge between the related address vertices. The is principally the same as the graph representation already discussed in 3.3.1
- *Entity graph*: In an entity graph, each node represents a real-world actor. In this graph, a vertex represents a set of addresses known to be controlled by the same real world actor. Transactions between these real world actors are represented as edges between the graph vertices. This is principally the same graph representation type as discussed in 3.3.1.

The power of GraphSense lies in the graph representations of the Bitcoin blockchain. This allows the developers of GraphSense to query these graph abstractions using graph-based algorithms. This makes GraphSense a great candidate to use as a tool to perform investigations on Bitcoin or other cryptocurrency blockchains. GraphSense currently has major support for Bitcoin, Bitcoin Cash, Litecoin, Zcash and Ethereum.

[1]https://bitcoin.org/en/bitcoin-core/

While GraphSense proves to be a very powerful tool, it also requires extensive computational resources to host and operate, since the Bitcoin address- and entity graphs can grow very big. To give some scale to this, on the official GraphSense website documentation page [2], it is stated that they are using the following hardware in their production environment:

- *1 master node: Dell PowerEdge R740, 2x Intel Xeon 2.60GHz (14 cores), 320GB RAM, 3x4TB HDD (Raid 5), 2x512GB SSD (Raid 0)*
- *8 worker nodes: Supermicro 815TQC-R501WB, 2× Intel Xeon 1.7 GHz (6 cores), 256GB RAM, 4TB HDD, 3×2TB SSD*

All nodes are connected via 10Gbit ethernet interfaces and a corresponding switch. With this setup the computation time to create the address and entity graph using the transformation component is currently 13h for Bitcoin.

This is a non-trivial hardware setup for a crypto analyst to host their own GraphSense instance. There is a demo instance available, but when playing around with it after requesting a demo API key, the interface felt very slow and unresponsive.

4.1.2 A custom implementation

Multiple commercial and open-source solutions exist that can aid the process of Bitcoin related investigations. The solutions mentioned previously in this section are some of the many examples, whom all mostly provide comparable functionalities. The full node JSON-RPC servers, Bitcore Insight and the ElectrumX server are excellent data providing services. They provide extensive information on the state of the Bitcoin blockchain and all its key elements, such as individual blocks, addresses and transactions. The downside is that they are only providers of data and require a client to consume their data to execute custom investigative logic. GraphSense, as a complete investigative tool, does provide this missing logic part, but it requires a high toll in terms of hardware requirements.

These constraints on existing solutions incentivise the creation of a new open source application that is specifically tailored to automate the steps needed to investigate ransomware clusters in the Bitcoin ecosystem. The application should be as lightweight as possible, in order to be easily deployed by anyone who would choose to use it. With these goals in mind, *TRaP* was developed. In the continuation of this chapter, a discussion on the architecture, the development and the evaluation of TRaP will be given.

[2]https://graphsense.info/documentation/

4.2 Architecture

The following diagram 4.1 shows the components that are necessary for the ransomware family analysis, as well as how they relate to each other. Each of these components and why they are needed is described in the following sections.

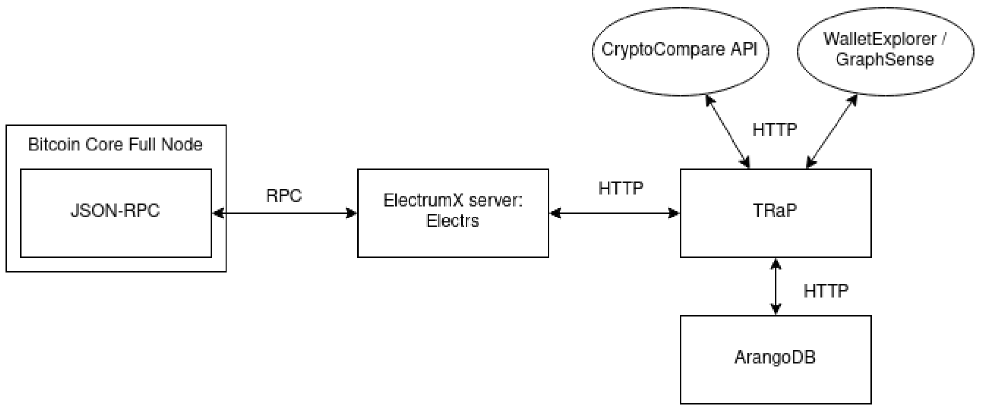


Figure 4.1: Diagram showing the architectural components used in the local testing environment and how the components are related to each other. Squared components are components that are hosted locally. Rounded components are hosted online and consulted over the web.

4.2.1 TRaP

The tool itself is a command-line application that has all features discussed in the analysis pipeline in chapter 3 implemented. The tool can be used to individually trigger every task in the pipeline and can produce an automatically generated first insight into the impact of a ransomware cluster, based on the information that it is presented with. The tool can be presented with a collection of seed addresses. Alternatively, it can search for possible ransomware related addresses based on a set of given transaction parameters to start analysing a ransomware cluster without a readily available set of seed addresses.

The programming language used to develop the tool is Typescript [3]. Typescript is a strongly typed programming language that is a superset of JavaScript.

TRaP interacts with its neighbours, the Electrs server, the ArangoDB database, and the CryptoCompare API over HTTP. To do this, the well known Axios [4] node module is used.

4.2.2 Electrs server

Electrs is the name of an efficient re-implementation of Electrum server, inspired by *ElectrumX*, *Electrum Personal Server* and *bitcoincore-indexd*. Electrs is developed in Rust [5], hence the name is a contraction of “Electrum” and “Rust”. The original version of Electrs was mainly developed by Roman Zeyde and is available on GitHub [6]. The original version has since been forked many times. One of these forks is the *Blockstream/electrs* fork [7]. This fork implements the same functionality as the original, but it is optimised to be hosted as a public server, in contrary to the original one. The Blockstream/electrs fork is the version that is hosted locally in the environment the experiments in this thesis have been conducted in.

The reason for choosing this version of Electrum as a backend, is because it is also available as a public server on `www.blockstream.info`. Because of this, when running TRaP it can either be configured to use the public Blockstream.info server or a locally hosted one. Locally hosting still has a significant advantage however. There are no network bottlenecks and the locally hosted electrs is not rate limited. This makes running an analysis with TRaP using a locally hosted electrs server significantly faster.

[3]https://www.typescriptlang.org/
[4]https://axios-http.com/
[5]https://www.rust-lang.org/
[6]https://github.com/romanz/electrs
[7]https://github.com/Blockstream/electrs

When running the electrs server for the first time it needs to index all the data it fetches at the JSON-RPC of a Bitcoin full node. The indexation process fills a database containing all information on blocks, transactions and addresses. It also calculates information that is not directly available in a JSON-RPC of a full node and stores this information. Examples of this are the balance of an address or a transaction history of a given address. This is also what makes an electrs server a necessity in the use case of this thesis. A blockchain is by design a linear data structure. At the time of writing this section, the Bitcoin blockchain has a total size of around 600GB. If someone wanted to get a transaction history of a given address, the entire blockchain would have to be linearly parsed to give a closing result to this query. This leads to a linear time complexity in terms of the size of the blockchain. By indexing this information once in a database like electrs, each lookup can be done in constant time. However, it must also be noted that the initial indexation process takes a lot of time as well. When setting up the electrs server for the environment of this thesis, the indexation process took around four and a half 24-hour days.

4.2.3 Bitcoin Core full node

A Bitcoin Core full node is necessary to operate as a data backend for the electrs server. There are two cases in which the full node will be consulted. Primarily the full node's JSON-RPC server is used by the electrs server during its initial indexation process to retrieve all the data it needs. Secondly, the full node also stays connected to the Bitcoin network and keeps track of new blocks in the blockchain. When this happens, the electrs server also needs to fetch these updates from the JSON-RPC and update its own database. It is important the full node is running at all times when the electrs server is running, since it relies on the JSON-RPC to fetch new and updated data. There are several full nodes available for use. The one that is used in this setup is the Bitcoin Core full node. Bitcoin Core is the reference implementation of Bitcoin [8].

4.2.4 ArangoDB

In section 3.3 of the previous chapter it is explained that a network of Bitcoin transactions can be represented as a graph. TRaP also stores the addresses it analyses and their related transactions in a transactions graph. Storage of the graph could have been done by implementing a graph data structure, or using an existing one from an external node package. However, TRaP relies on *ArangoDB* for storing its graph data.

ArangoDB is a multi-model database system developed by triAGENS GmbH [FB18]. Data can be stored as key-value pairs, documents or graphs. All data stored in an ArangoDB database can be queried with the same query language: Arango Query Language (AQL).

There are good reasons to warrant using a graph database. The first one being that data can be stored persistently, contrary to storing a graph data structure in memory and having to load it in memory again every time TRaP is used. The second, and perhaps the most significant reason is that a query language like AQL allows running graph queries on the data. This makes it easier to query the transaction graph for relevant features and allows for the calculation of collector addresses in ransomware families.

The graph representation type that was chosen for this ArangoDB instance is the address graph representation type, as described in section 3.3.1. The reason for this is that there is no added benefit to build a cluster graph, since TRaP already calculates these clusters itself. Then between an address graph and a transaction graph, the address graph is best suited for the graph queries that will follow which are used to calculate collector addresses.

4.2.5 CryptoCompare API

The next component that is necessary in the setup is the API provided by CryptoCompare [9]. CryptoCompare is an internet forum where people can discuss the latest trends in cryptocurrency. They also provide an API which gives access to, among other things, the current and historical prices of several different cryptocurrencies. The reason for choosing CryptoCompare is mainly because it is free, unlike other similar API providers. Besides being free, it also provides extensive control on what historical data can be queried.

[8]https://github.com/bitcoin/bitcoin

[9]www.cryptocompare.com

This API is not used in the final version of TRaP because it became obsolete, but it is left as a part of this text because the code that uses this API is still part of the current codebase just in case it could come in handy in any future upgrades.

4.2.6 WalletExplorer/GraphSense

The final component is a duo of external services that offer the possibility to link a given Bitcoin address to a real world actor or service. The intention is to use these services to try and link ransomware addresses to exchange services in order to determine where the collected ransoms are exchanged to fiat currency. The GraphSense instance that is consulted is the demo instance hosted by the GraphSense development team. A demo access key had to be explicitly requested in order to access this demo instance.

4.3 Features

In this section, all of the features that are implemented in TRaP will be laid out. In each of the following subsections, the goal, the relevant implementation details, as well as a small evaluation of each feature will be given.

4.3.1 Clustering using multiple-input spending heuristic

An operation that is useful in several different scenarios when working with Bitcoin addresses is the clustering operation. A cluster represents a collection of addresses that is known to belong to the same real-world actor. Clustering can be done based on the multiple-input spending heuristic as explained in section 3.2.1. Clustering can be used as a way to discover new, previously unknown seed addresses that have been co-spent with known seed addresses as inputs for a transaction. If two addresses appear in the same cluster, they can be assumed to belong to the same wallet or real world actor.

Another scenario in which clustering can be useful is in the search of seed addresses directly on the blockchain as an alternatively method to collecting seed addresses by one of the conventional ways. This search strategy is elaborated on further in the following section 4.3.2.

TRaP has a clustering algorithm built in. The algorithm takes a list of known seed addresses. Then, for every address a, all transactions t are looked up. Then, for every transaction t where a is an input, all addresses that are in the input of t are added to a new cluster c. When this is done for every address, a set of individual clusters remains.

The next step is to calculate if there exist clusters that overlap. Two clusters overlap if the size of their intersection is greater than 0. When two clusters overlap, they are merged and considered a new cluster. Overlapping clusters are being calculated so long as the set of clusters doesn't change anymore. These steps are recursively done to new clusters until the set of clusters stops changing. At this point, each remaining cluster contains addresses that belong to the same real world actor or wallet.

Evaluation

Testing the algorithm for correctness can be done by running it on an existing set of seed addresses. Luckily, there are several sets of seed addresses related to ransomware attacks in the past publicly available. The one used for the evaluation of the TRaP clustering algorithm is available on GitHub [10]. This set of addresses is the same one use by Paquet et al. in [PHD19]. Table 4.1 shows a comparison of the results obtained in Paquet et al. versus the results obtained by using TRaP.

Ransomware family	Paquet et al.		TRaP	
	Clusters	Exp. Addr	Clusters	Exp. Addr.
Locky	1	6827	2	6837
SamSam	11	41	11	72
JigSaw	4	17	4	17
WannaCry	1	6	1	6
DMALockerV3	3	147	3	155
CryptXXX	1	1304	1	338
NoobCrypt	1	17	1	8

Table 4.1: Table comparing the results produced by Paquet et al. and the clustering implementation in TRaP. For Paquet et al. as well as for TRaP two columns are shown. The *"clusters"* column shows the amount of clusters detected. The *"Exp. Addr."* column shows the expanded address set of seed addresses belonging to the same ransomware family after clustering.

For the ransomware families of Locky, SamSam, JigSaw, WannaCry and DMAlockerV3 almost identical results are obtained. TRaP discovers a slightly greater number of addresses per ransomware cluster. This is due to the difference in time when both clustering operations were run. TRaP managed to discover some addresses that only appeared in transactions after the publication of [PHD19].

CryptXXX and NoobCrypt show largely different results. It is not immediately apparent why this is the case. To try to explain this difference, a manual clustering of the NoobCrypt family was made with

[10] https://github.com/behas/ransomware-dataset

the help of an online block explorer, starting from the same two seed addresses that exist in the source dataset. The manual clustering confirmed the results of TRaP and even showed that the results obtained by Paquet et al. could simply not occur. A guess at an explanation for this could be that the source dataset provided on GitHub is not fully up-to-date with the addresses used in the final research of the original authors. It could be they eventually obtained more seed addresses but did not include these in the dataset they published.

To further investigate this assumption, the original authors of [PHD19] were contacted to find out if this could be the case. With a quick response they verified that the source data set they provided on GitHub is indeed correct and up-to-date. This means the differing results have another cause. In their response, they also included the clusters they found for the NoobCrypt ransomware family when running their solution again in the present day. This rerun yielded a collection of 28 addresses compared to the 17 addresses in the original research paper from 2018. Thanks to now having access to this address collection, the difference in results could be more easily debugged.

The differing results appeared to be caused by a difference of personal interpretation of the clustering operation.

In TRaP, the clustering approach up until this point was to take a seed address and build a cluster around it using the multiple-input spending heuristic. Then, after calculating a cluster for each seed address, merge all intersecting clusters until no intersecting clusters exist anymore. The resulting collection of clusters is the definitive collection of clusters.

In [PHD19], the clustering approach went as follows: take a seed address and build a cluster around it using the multiple-input spending heuristic. Then, this is where the difference takes place, **do the same thing for all newly discovered addresses as well**. In the approach taken by TRaP previously, this step was skipped, yielding less addresses from the clustering heuristic. Finally, merge all intersecting clusters until no intersecting clusters exist anymore. The resulting collection of clusters is the definitive collection of clusters.

The approach used by Paquet et al. can yield more addresses and possibly larger clusters in many different cases. After gaining this insight, the clustering approach in TRaP was changed to operate using the same approach. The results after recalculating clusters for the same ransomware families as in table 4.1 are shown in table 4.2.

Ransomware family	Paquet et al.		TRaP	
	Clusters	Exp. Addr	Clusters	Exp. Addr.
Locky	1	6827	1	7094
SamSam	11	41	11	72
JigSaw	4	17	4	23
WannaCry	1	6	1	6
DMALockerV3	3	147	3	177
CryptXXX	1	1304	1	1742
NoobCrypt	1	28	1	28

Table 4.2: Updated version of table 4.1. Notice the difference in the amount of expanded addresses found by Paquet et al. in comparison to table 4.1. The numbers for NoobCrypt in this table have been altered based on the collection of addresses received for the NoobCrypt family from the authors of [PHD19].

As can be seen in table 4.2, TRaP now yields results in the same order of magnitude as in [PHD19]. In all cases, except for the WannaCry and NoobCrypt ransomware families, TRaP produces more seed addresses in the same amount of clusters. The reason for this remains the same as before, but is now consistently visible for all ransomware families. The additional addresses only become active after the publication of [PHD19]. This is confirmed by the dataset received afterward containing the newly updated NoobCrypt addresses. TRaP produces the identical addresses in the NoobCrypt cluster as was shared by the authors of [PHD19] via email. With these conclusions, this evaluation can be finalized by saying that TRaP manages to correctly cluster and detect the expected set of seed addresses, starting from a given set.

4.3.2 Searching seed addresses

In section 3.1 of chapter 3, four different ways of collecting seed addresses related to a ransomware cluster are laid out:

1. Finding real victims on the web
2. Create synthetic victims
3. Use third party sets of addresses
4. Searching existing transactions

The first three of these methods have been proposed and used in existing research related to tracing ransomware payments in the blockchain. The fourth one, to search for ransomware addresses in existing transactions on the blockchain, is a new one that is implemented and tested in this thesis.

The search strategy that is implemented in TRaP has been discussed on a high level in section 3.1.4. The idea is to search for addresses throughout the blockchain that fulfill certain constraints that might increase the chance that they belong to a specific ransomware cluster. When a set of addresses is found that adheres to the stated constraints, it is called a *candidate set* of seed addresses.

There are two types of constraints. The first are constraints that can be universally used and defined for any ransomware cluster. Examples of this are the timestamp around which we know an attack occurred. Knowing this timestamp drastically narrows down the amount of blocks that must be looked through to find candidate addresses. Another constraint of this type is knowing the ransom sum and in what currency it was demanded. This makes it so only addresses that have received the known ransom sum will be considered as a candidate address. The constraints in this example can be passed as command line arguments to TRaP using the following arguments:

- *timestampRange*: A range expressed in epoch timestamps (in seconds) defining the start and the end of the timestamp range in which the ransomware attack is known to occur. These parameters can be taken broadly.
- *currency*: The currency in which the ransom that is specified in the following argument is given. Currently TRaP supports BTC (Bitcoin) and USD (US Dollar) as arguments for this parameter.
- *ransom*: The actual ransom sum that is known to be asked of the ransomware victims.

The second type of constraints are those that are inherently specific to what is known about a new ransomware attack. For example, it could be that it is known from collecting seed addresses that the ransomware family always uses the same type of address. For example, the Deadbolt ransomware family seems to be using only BECH32 formatted addresses. These addresses are recognizable because they always start with the “bc1” substring. This could be a useful constraint when looking for Deadbolt candidate addresses specifically.

Since the second type of constraints are non-standard and could practically be any constraint, these are not definable through the command line arguments when using TRaP. They can, however, be written down in code in a single, centralised filter function if needed. This leaves full flexibility to define any constraint possible, but it is a big con in terms of usability, since it requires some level of coding.

Experiment: searching Deadbolt addresses

Since the described approach to discover ransomware related seed addresses is new, some evaluation is needed to determine the effectiveness of this approach. In order to measure the feasibility of this approach it will be applied to a recent ransomware attack on Asustor NAS systems [Tou22]: the *Deadbolt* ransomware attack. Based on results obtained through Google Trends for the combined search term “deadbolt ransomware” it can be derived that the first occurrences of the attack happened in the week of February 13th, 2022. The attack ”peaked” somewhere between February the 20th and the 26th, declining again to leave the spotlight around the 15th of March. The entire attack lasted almost a month in total. Also based on Google Trends, it is known the main country of interest for the Ransomware attack seemed to be Australia, followed by Canada, Italy, the United Kingdom and the United States. No other countries seem to have had search queries for the same ransomware attack, neither within the same time period, nor outside of it.

The attack was specifically targeted at Asustor NAS systems. Many users, as well ass the manufacturer believe the attack happened through the exploitation of a vulnerability either in the PLEX media server or EZ Connect, which allows users access to their device [Tou22]. The ransom that is requested is fixed for all victims at 0.03 BTC. From the few ransom notes that could be found on web forums and blogposts, it can be concluded that the seed addresses used in the ransomware family are BECH32 addresses, meaning they all start with the “bc1” substring, as described in a previous paragraph above.

Based on these known constraints, a search operation can be done throughout the blockchain for transactions that adhere to the information that is available on the Deadbolt ransomware:

```
$ trap search --timestampRange 1644710400-1645228800 --currency BTC ransom 0.03
```

Through the parameters passed to the command, TRaP knows to search for transactions between the interval of February 13th, 2022 and February 19th, 2022. It will only search for addresses that have received a one-time input of 0.03 BTC within the given time interval. Hardcoded into the code is also that the addresses it finds must be BECH32 addresses, and that they have at least one outgoing transaction as well. An outgoing transaction means that the ransom in the address is being moved to a possible collector address, or to further services like mixers or exchange services. The same search operation is also performed for the weeks following February 19th. By doing this, week by week, the results can be compared to the data obtained in Google Trends. The results of this can be seen in figure 4.2. In total, TRaP “discovered” 1278 candidate Bitcoin addresses that adhere to the aforementioned constraints.

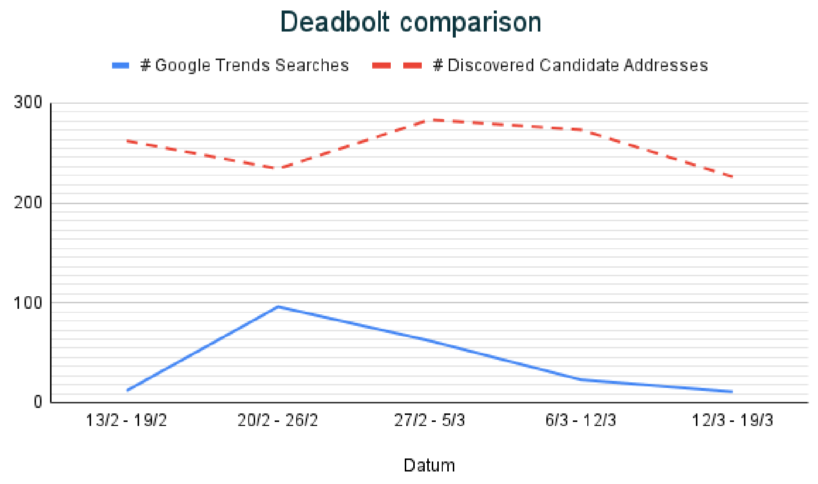


Figure 4.2: A graph showing a trend line for the amount of searches for “deadbolt ransomware” on Google Trends versus the amount of candidate seed addresses found by TRaP for the Deadbolt ransomware family within the same time intervals.

The idea behind the visualisation in figure 4.2 was to try and determine if both trend lines could be correlated to each other. If they had shown the same trend, this could be used as an argument for the accuracy of the search strategy implemented in TRaP. However, as can be seen in the figure, the trend lines do not seem to match the same trends at all. Even more, their trends seem to go the opposite ways in the week between the 20th of February and the 26th of February. The expectations were that both trend lines would show a distinct peak within that same week, but the actual amount of discovered transactions seems to be at its second lowest point in that week.

The conclusion that can be drawn from this is that the search strategy implemented in TRaP cannot be proven to be effective based on this comparison. It could be possible that the set of discovered candidate addresses is too broad and includes many addresses that have no correlation to the Deadbolt ransomware family. Further measures are needed to narrow down the set of discovered candidate addresses to a smaller, hopefully more likely set of addresses to belong to the Deadbolt ransomware cluster. Clustering the set of candidate addresses could be a solution to this.

Updated experiment: Clustering candidate addresses

Searching the blockchain for addresses that satisfy a set of constraints is a useful way to construct a set of candidate seed addresses. However, as the name says, it is only a set of *"candidate"* seed addresses. There is no way to know for certain that all the addresses that are found this way are in fact part of the same ransomware cluster, or even any ransomware cluster at all. There is a need to further strengthen the claim to which addresses in the candidate set belong to the ransomware cluster. One way to strengthen this claim for a given address is to check if the address has been co-spent with other addresses that satisfy the same set of constraints. If this is found to be the case, it greatly increases the probability of these addresses belonging to the ransomware cluster.

Expanding this idea from one address to all addresses in the candidate set can be done by running the multiple-input spending clustering algorithm on the entire candidate set, with the additional rule that all addresses that are put into clusters must all satisfy the same set of constraints. Adding this rule is necessary to eliminate generating false positive addresses. It could be that an address has been used as input to a transaction together with other addresses that do not satisfy the same set of constraints. In this case, it is likely that the transactions in which this situation occurs are part of transactions generated by an exchange service or a mixer service, or are just general person-to-person transactions. These addresses would then very likely not be owned by the ransomware cluster and are thus not interesting to add to the clusters of seed addresses.

With the clustering step added, the experiment to find Deadbolt seed addresses has been conducted again. The results of the updated experiment are shown in figure 4.3.

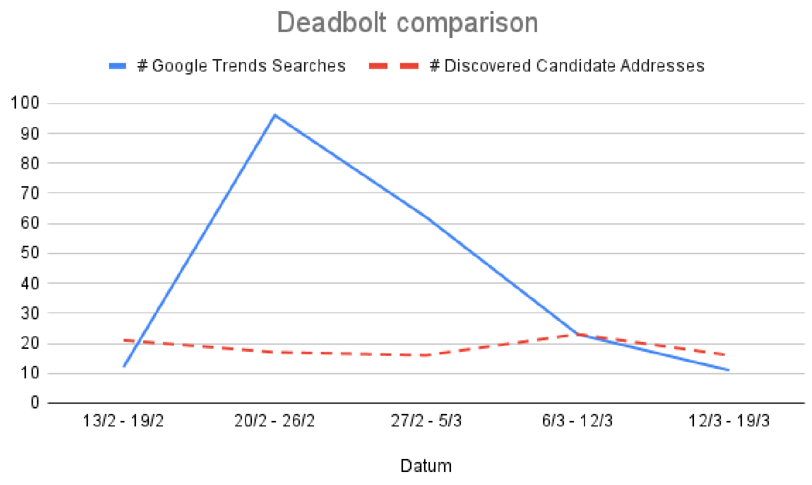


Figure 4.3: A graph showing a trend line for the amount of searches for "deadbolt ransomware" on Google Trends versus the amount of candidate seed addresses found by TRaP for the Deadbolt ransomware family within the same time intervals. The difference with this graph compared to figure 4.2 is that the candidate addresses that are "discovered" here have the additional property that they have been co-spent with other addresses that adhere to the same set of constraints.

Compared to the previous experiment where no clustering step was present, now only a total of 93 addresses are found that could qualify as seed addresses, compared to 1278 addresses in the previous experiment. Just to keep things clear; the experiment that is conducted here is the same as in the previous section 4.3.2, now with an added clustering step introduced in this section. In figure 4.3 it can be seen that the amount of discovered candidate addresses still does not follow the same trend line as the Google Trends search query data. Unfortunately, this also means nothing can be said with certainty about the discovered candidate addresses and if they belong to the Deadbolt ransomware family at all. This also leads to the conclusion that this search approach to discover seed address cannot be considered a legitimate strategy. When studying new ransomware attacks, it would be better then to use one or more of the alternative ways to gather seed addresses, as listed in the beginning of this section.

Something thing that *could* be argued from the collection of candidate addresses that TRaP can discover this way is that they could be used to calculate an upper bound for the financial impact the Deadbolt ransomware family has made in their February attack. The upper bound, for the Deadbolt ransomware February attack, would then be 2.75 BTC.

4.3.3 Identifying collector addresses

A key insight into the organization of a ransomware cluster in the Bitcoin ecosystem is how the outgoing relations of the seed addresses behave. It is useful to know where the cryptocurrency is moved once it is received in a seed address and where it is going from there. One key idea that recurs in several ransomware clusters is that the currency received in the seed addresses first gets collected in one or more collector addresses. From there it can be moved away into a mixer service to anonimize all following transactions, or it could go straight to a crypto exchange service. Finding these collector addresses is relatively easy. The initial set of seed addresses can be looked at as the first layer of the organizational structure. Then, all addresses that have subsequently received currency from addresses in the first layer can be looked at as the second layer. Collector addresses are situated in this second layer, since they aggregate currency from the seed addresses. A collector address can be identified by calculating the in-degree of the given address. In [PHD19], the in-degree of a given address is defined as follows:

Definition 4.3.1. The in-degree deg^{-} of a Bitcoin address is the sum of all unique incoming relationships within the scope of a ransomware cluster specific outgoing-relationships graph.

Since TRaP stores the seed addresses and their outgoing-relationships graph in ArangoDB, the Arango Query Language (AQL) can be used to query the outgoing-relationships graph for all collector addresses. The AQL query looks as follows:

```
FOR a IN Address
    RETURN {
        "address": a.address,
        "inDegree": LENGTH(FOR v IN INBOUND a GRAPH graphName RETURN true)
    }
```

Listing 7: AQL query to calculate the inDegree of second layer addresses. The variablename “graphName” must be substituted for the actual graphname that is used in the ArangoDB system.

The AQL query in listing 7 yields the in-degree for every address in the second layer. When the in-degree of an address is higher than a chosen threshold, an address can be called a collector address. But how must this threshold be chosen? Also in [PHD19], Paquet et al. choose to put the threshold at $deg(a)^{-} >= 2$ in order for address a to be identified as a collector address. However, they also measured that, over the 14 different ransomware clusters they studied, the average in-degree of an address was 12, with a standard deviation of 27.66. The highest in-degree they measured was 742 incoming relationships. These numbers show that choosing the exact threshold is very case specific. For example, a very large ransomware cluster like Locky, with over 7000 seed addresses might produce collector addresses with a very high in-degree, while smaller clusters like e.g. SamSam, which consists of 44 known seed addresses, might have collector addresses with on average an in-degree threshold lower than 5. It should also be noted that maybe in a very specific case, no collector addresses are used at all. In this case, all second layer addresses would have an in-degree of at most 1.

Evaluation

The correctness of this approach can again be tested by trying to reproduce results previously obtained in the research by Paquet et al. [PHD19]. Table 4.3 shows, per ransomware family, the amount of collector addresses that are obtained by using TRaP compared to the amount of collector addresses found in [PHD19].

Table 4.3 shows that TRaP and Paquet et al. produce identical results for the Locky and CryptXXX ransomware families.

For the other families results differ very little and are always within the same order of magnitude. The small differing results can be attributed to the differences also measured in the clustering step from section 4.3.1. The identification of a collector address is dependant on the amount of incoming transactions from seed addresses in the same ransomware family. Since table 4.2 showed that TRaP managed to find slightly

Ransomware Family	Paquet et al.	TRaP
Locky	571	571
SamSam	37	38
JigSaw	17	26
WannaCry	≤ 6	1
DMAlockerV3	71	79
CryptXXX	926	926
NoobCrypt	54	71

Table 4.3: Table showing a comparison of the amount of collector addresses found in Paquet et al. versus TRaP for a given ransomware family. Specifically for WannaCry, we only know from Paquet et al. that the amount of discovered collector addresses must be lower than 6, but no specific value is given. WannaCry is still included in this table to maintain consistency with table 4.1, 4.2 and 4.4.

more seed addresses in all cases, this also generates more inputs for possible collector addresses. This explains why there is a small increase in collector addresses for some families.

For families such as Locky and CryptXXX, the same collector addresses that were already known by Paquet et al. just get re-used as collectors by the new seed addresses discovered by TRaP.

4.3.4 Estimating lower bound financial impact

Based on the expanded seed address datasets found for each ransomware family in section 4.3.1 it is possible to make a lower bound estimation of the financial gains each ransomware family made. To calculate the financial yield of a given family, the values of all incoming transactions into the seed addresses of the family can be added together. It must be noted that this sum only forms a lower bound estimation. It is perfectly possible that there exist clusters of seed addresses that went undetected in the previous clustering steps from section 4.3.1. These undetected clusters would consists of addresses that have no overlap with the already known clusters. The incoming transactions into these undetected clusters would then also not be accounted for in the calculation of the financial impact of a ransomware family.

Evaluation

Ransomware Family	Paquet et al.	TRaP
Locky	15 399.01	16 355.16
SamSam	632.01	668,72
JigSaw	≤ 10.75	5.81
WannaCry	55.34	59.66
DMAlockerV3	1 505.78	2 185.30
CryptXXX	3 339.68	3 427.22
NoobCrypt	54.34	1142.44

Table 4.4: Table showing a comparison of the lower bound financial impact of a given ransomware family published by Paquet et al. versus results produced by TRaP. The results are always expressed in BTC, not in Euros or USD. Specifically for JigSaw, it is only known from Paquet et al. that the lower bound must be lower or equal to 10.75. JigSaw is still included in this table to maintain consistency with table 4.1, 4.2 and 4.3

Table 4.4 again shows a comparison between results obtained by Paquet et al. and results produced by TRaP. All results are expressed in BTC, not in Euros or USD. The reason for this is that the exchange rate between BTC and Euros or BTC and USD is at times very volatile. Also, most of the clusters receive payments spread over several months, which makes it difficult to give an exact approximation of a given family in fiat currency.

For all ransomware families, except NoobCrypt, the results found by Paquet et al. and TRaP are similar and within the same order of magnitude. Again, as was also the case in table 4.2 and 4.3, the results produced by TRaP are slightly larger due to including more transactions that have been made after the publication of [PHD19].

Besides this, a significant increase is measured in the NoobCrypt family. Upon taking a closer look at the transaction timestamps of transactions involved with the NoobCrypt seed addresses, it became clear the NoobCrypt cluster became very active after the publication of [PHD19]. NoobCrypt seed addresses received over 1000 BTC in the time frame between 2018 and 2022, when this thesis is written. Another conclusion that can be drawn from this is that NoobCrypt is a ransomware family that reuses the seed addresses it already owns, since newer attacks in the last few years keep receiving ransom payments on these same seed addresses as already discovered in [PHD19].

4.3.5 Determining payment trend over time

A second method to gain insight into the financial impact of a ransomware family is to determine the payment trend of a given family over time. Determining and visualizing this trend can help in gaining insight into the effectiveness of a given ransomware family or the duration of an attack. The payment trend in a ransomware family can be determined by plotting all incoming transactions on the x-axis of a graph based on their timestamp, and plotting the value they brought into the ransomware family in BTC on the y-axis. Figure 4.4 shows a comparison for each ransomware family of the payment trend determined by Paquet et al. versus the one determined by TRaP. An important note to make concerning this figure is that the graphs plotted by Paquet et al. use USD as a unit for the y-axis, while the output graphs of TRaP use BTC instead. The reason for this is that it seems to make more sense to denominate these values in BTC, since their value always remains the same. If denominated in USD, there is more fluctuation possible over time as the exchange rate between USD and BTC changes. Paquet et al. did not elaborate in their paper which exchange rate they utilized to plot these graphs. It is currently unsure if they used a fixed average exchange rate over time, or if they used accurate exchange rates for each timestamp of each transaction. However, the fact that the trend lines seem to be very similar compared to the ones produced by TRaP, it can be assumed they used a fixed average exchange rate. This would mean their graphs depict a correct shape of the trendline, but the associated values in USD could deviate from the actual value at each point in time.

Evaluation and discussion

It is clear that TRaP produces the same results as Paquet et al. This confirms again that each of the steps in the analysis pipeline is correctly implemented in TRaP. What else is interesting is how these payment trends can be interpreted.

Both CryptoLocker and WannaCry show very steep curves followed by a nearly flattened out continuation of the curve. What this means is that both of these ransomware families benefited from a short-term, viral campaign (no pun intended). This shows the attackers decided to infect as much systems at once in a short period of time. The attack strategy could have been to exploit a vulnerability that quickly got patched after exploitation by the attack. For WannaCry, the claim can be confirmed. Europol stated the WannaCry attack of 2017 infected over 200.000 victims in 150 different countries [17].

Another reason to help explain the steep rise in the payment trend curve could be that these families targeted victims' companies or victims with high value data. High value in the sense that they are required for the continuation of a business' operation. For CryptoLocker, this is the case. CryptoLocker is a ransomware virus that is spread as an attachment to a seemingly innocent email, coming from a legitimate company [Nar13]. After infecting a victim, CryptoLocker specifically encrypts data files with specific extensions, including Microsoft Office, OpenDocument and AutoCAD files [13]. These are file types that are essential within the operation of many businesses worldwide. Encrypting these files for a ransom will then put a higher pressure on the need to pay a ransom fast, hence why also CryptoLocker's payment trend curve makes a steep ascend.

Locky and SamSam show different behaviour. Locky's payment trend shows a slower but steady inflow of money into the ransomware family. Locky's attack seems to be spread over around six months, where CryptoLocker and WannaCry happened within the span of a few weeks. The Locky ransomware is, like CryptoLocker, spread via an email attachement in the form of a Microsoft Word document that contains malicious macros [Sea16]. When the document is opened the victim is instructed to enable macros. If they do, they save and run a binary file that then actually downloads the Trojan that encrypts the victim's data. After encryption, the victim is instructed to visit a website through the Tor browser that explained what happened to the victim's data and how they can retrieve it by paying a ransom in Bitcoin.

SamSam shows more of a staircase pattern in its payment trend. This could insinuate that this ransomware family makes several smaller attacks spread over time. This could be to stay under the radar. Another reason for this could be that they might change their attack vectors multiple times over time. They could also have tried to attack different victims at different times. I could however not find any conclusive reason to definitively explain SamSam's payment behaviour.

4.3.6 Uncovering real world actors

In section 4.3.1 is explained how TRaP uses a clustering approach to expand a given set of seed addresses to a larger set of addresses that belong to the same real world actors. In section 4.3.3 was then explained how this expanded collection of seed addresses can be used to construct an outgoing relations graph and how this graph can be used to determine collector addresses related to the ransomware family. In many ransomware families collector addresses serve the purpose of collecting ransoms from the seed addresses, hence their name. A logical question to ask is who "owns" the collector addresses. Possibilities are that the collector addresses belong to either crypto exchange services to exchange the money to fiat currency, mixing services in order to increase anonymity before moving the money to an exchange service, or possibly other more exotic services such as gambling sites or online games. Knowing which services these collector addresses belong to can be a useful feature for law enforcement. If this could be quickly determined after the occurrence of a new ransomware attack, law enforcement could possible intervene at the correct crypto service providers to help minimize the impact of an attack. The problem that poses itself is how this identification can be done. There are different available solutions that provide this functionality, depending on the budget one might have and the goals that should be achieved.

Walletexplorer

Walletexplorer [11] is a website made by Ales Janda. Walletexplorer is essentially another online block explorer, but this time with the added functionality that it labels addresses with real world actors it thinks the addresses belong to. It does this by taking addresses that are known to belong to real world actors, and also running the clustering approach using the multiple-input spending heuristic on these sets of addresses. Original addresses belonging to real world services are discovered by manually registering to the services and saving the addresses that are used internally after making a small transaction. By doing this, walletexplorer serves as a a database of addresses belonging to real world actors. Beside offering an in-browser website, walletexplorer also serves a JSON API over HTTP. The JSON API is available on demand after inquiry via email.

The downside of walletexplorer is that the database of original addresses and wallets has not been updated since 2016. This means newer exchange services or newer wallets created after this year are not included in the data. The original author of walletexplorer since joined Chainalysis to do similar work there. This also means that using walletexplorer as a means of identifying real world actors is not very useful when studying newer, more recent ransomware attacks in the present day and in the future. However, the functionality is still implemented in TRaP just in case, and also because there are no other free or open source alternatives immediately available.

Chainalysis

An alternative to walletexplorer, as mentioned in the previous paragraph as well, is Chainalysis [12]. Chainalysis is a company that offers cryptocurrency investigation and transaction monitoring solutions as a service. As part of one of their services they offer the functionality to identify real world actors, such as crypto exchange services or other based on given addresses or transactions. However, their services are not offered for free and are therefore also not further investigated in this thesis, since one of the objectives of developing TRaP is to develop a free and open source alternative to other cryptocurrency analytics platforms that is as lightweight as possible.

[11]https://walletexplorer.com/
[12]https://www.chainalysis.com/

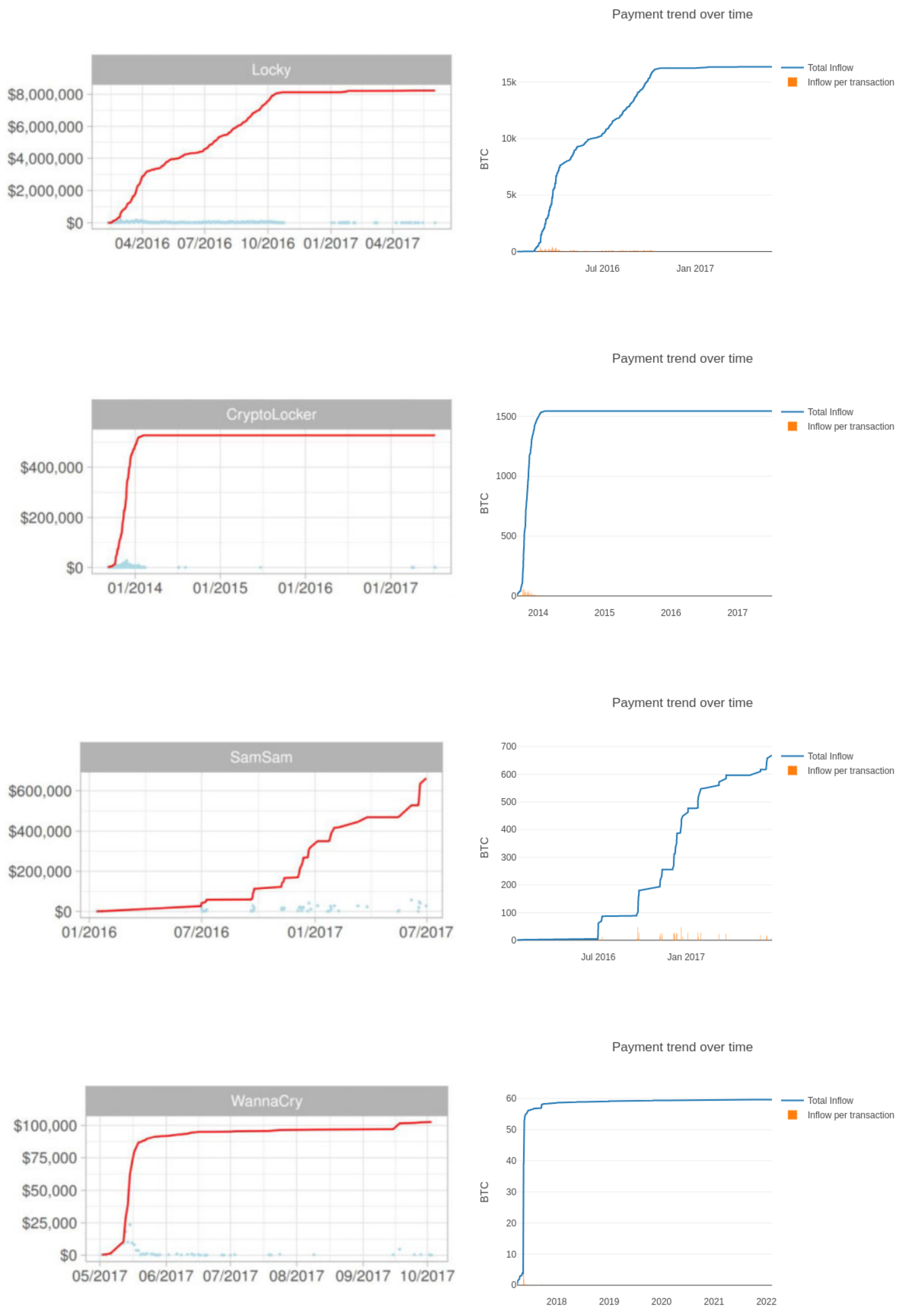


Figure 4.4: Figure showing, for each ransomware family (horizontally), the payment trend over time determined by Paquet et al. compared to the payment trend determined by TRaP. The x-axis represents time, while the y-axis represents the value of an incoming transaction into a ransomware family. Each transaction belonging to the given family is plotted cumulative as well as non-cumulative. Respectively red and blue in the plots by Paquet et al. and respectively blue and yellow in the plots produced by TRaP. Notice again that the value of an incoming transaction is expressed in USD for the plots by Paquet et al., and in BTC for the plots produced by TRaP, as argumented in the text. The trend lines still represent the same behaviour however.

GraphSense

In the motivation section 4.1 at the beginning of this chapter is explained what GraphSense is and what its pro's and cons are. Although GraphSense is a computationally heavy tool to host and run, it does provide a wider set of general purpose functionalities. One of these is that GraphSense also tries to keep track of which addresses belong to which real world actors. This is a feature that maybe can be used to help fill the voids where walletexplorer fails due to not being completely up-to-date. To do this, access to the demo environment of GraphSense can be granted by requesting an API key for the environment via email to the creators of GraphSense. A side note that must be made is that the GraphSense demo environment is still limited to 1000 requests per hour.

In order to not leave any stones unturned in terms of possibilities to identify real world entities behind Bitcoin addresses, TRaP has also been implemented to consult a GraphSense demo instance in order to try and retrieve the real world entity tags associated with the addresses it discovers. Unfortunately, this turned out to be a dead end. GraphSense could not provide any additional information for the addresses TRaP encountered in any of the previous experiments in this chapter. What it *did* know was that most of the addresses that were inspected in these experiments were in fact part of a ransomware family, but this was not new information.

4.4 Discussion and Conclusions

The current chapter up until this point has been entirely devoted to the motivation, design and evaluation of TRaP, a command line application to automate important or useful steps taken when studying or tracing ransomware attacks throughout the Bitcoin blockchain. In this section, I would like to reflect on the final version of this application, where I think its place could be in the bigger scheme of things, and which difficulties presented themselves that stood in the way of the vision I had for the application at the beginning of this thesis.

The initial, ideal vision for TRaP was that it would be an application that could, given a collection of seed address as input, do the following for a given ransomware attack:

1. Expand the seed addresses it takes as input to a larger collection of clustered seed addresses, using the multiple-input spending heuristic

2. "Follow" a trace of transactions starting from the addresses in the clusters of the previous step, and graphically visualize all the transaction traces that originated from these addresses.

3. Determine collector addresses related to the seed addresses

4. Determine the financial impact of the ransomware attack

5. Determine the payment trend over time related to the ransomware attack

6. Determine *who* owns the addresses that are associated with the ransomware family. E.g. which exchange or other service owns them.

This idealized vision has been successfully implemented for four of the six main requirements. Only requirement 2 and 6 have not been implemented or used as initially intended.

The second requirement stated the ability to visualize all transaction traces throughout the Bitcoin blockchain starting from the collection of seed addresses. Initially this seemed like a good and "natural" idea to implement, since the network of transactions that runs throughout the Bitcoin blockchain is inherently a graph datastructure. It would only be one extra step to calculate the subgraph showing all transactions related to the seed addresses in a given ransomware family. This is still the way it is implemented, and a rudimentary visualization of the problem can still be consulted in the ArangoDB web interface coupled to the ArangoDB instance that *is* used. However, I saw no real benefit in this graphic visualisation, since only the seed addresses (and the collector addresses to some extend) can be considered members of the ransomware family at hand with complete certainty. This is due to "the law of diminishing certainty" and the inability to identify real world actors behind a given Bitcoin address, as discussed in the following subsections.

Law of diminishing certainty

As discussed earlier, there is the problem of anonymity-enhancing services such as mixers and CoinJoin transactions. Essentially, for any given Bitcoin transaction there is no certain way to tell if the transaction belongs to a mixer service or if it is a CoinJoin transaction. More so, anything that happens in a mixer service, or what follows a mixer is completely ambiguous in terms of knowing if the trace that follows is still related to the same ransomware family.

When studying a ransomware family on the Bitcoin blockchain, the only addresses that belong to a given ransomware family with complete certainty are the seed addresses. Every transaction that follows from one of these addresses, and every address that appears in these transactions are increasingly less likely to still belong to the ransomware family the further away they are removed from a seed address (in terms of transactions originating from the seed address). The reason for this being that every transaction has some unknown probability of being a mixer or CoinJoin transaction. I have jokingly named this phenomenon the "law of diminishing certainty".

The law of diminishing certainty is not only applicable to transactions, but also to addresses. For any given Bitcoin address, it is hard to say with absolute certainty to which real world actor the address belongs. This also goes for addresses that belong to Bitcoin exchange services. Some Bitcoin exchange services have the tendency to reuse some of the addresses they own for internal management, e.g. to move around coins amongst their "own" wallets and/or the wallets of their customers. From a third party perspective looking at the Bitcoin blockchain, these internal transactions can be hard to attribute to exchange services. These transactions can, from the third party perspective, also seem to behave similar to mixer services or CoinJoin transactions, because they might aggregate money from several internal sources and spread the coins around again, without the specific purpose of increasing anonymity. This goes to show that not only intentional services like mixers or CoinJoins increase the anonymity of transactions, large organisations like exchange services active on the blockchain can also show this behaviour unintentionally.

An example address that shows this type of behaviour is "*1FLAng62VTeHsPuER2WzHzi5MbemPw48o5*". The address is known to belong to the Bitfinex exchange service according to walletexplorer.com. When taking a closer look at the address in block explorer [13], it can be seen that the address is involved in 53 transactions. Most of these transactions show either the address collecting relatively large amounts of Bitcoins, or spreading them out again to sometimes more than a hundred different addresses. If a trace of payments starting in a ransomware family would hypothetically lead to this address without knowing beforehand if the address is related to an exchange service, this would make it hard, if not impossible to determine if the address is either an exchange service, a mixer, or perhaps a legitimate address belonging to the ransomware cluster.

To revert back to the features of TRaP that are not implemented as intended, the law of diminishing certainty is the reason why requirement 2 has not been implemented as intended. Since nothing can be said with certainty about transactions or addresses by tracing payments starting in seed addresses, they cannot be considered members of the ransomware family, hence why they should not be included in the graph of transaction traces related to the ransomware family.

The only addresses, other than the seed addresses, that can be considered members of the ransomware family with some form of certainty are the addresses that are reachable within one transaction starting from the seed addresses. Often times, these are the addresses that manifest themselves as collector addresses, hence why there is good reason to think they still belong to the ransomware cluster. By trying to enter the head of an attacker, it can be assumed the attackers will move the coins they collected in the collector addresses to either a mixer, a CoinJoin transaction, or directly to an exchange service. However, without the proper means to link these entities to their addresses, this still remains a void of uncertainty.

Difficulties with identifying real world actors

This brings us to the second requirement that has not been implemented as originally envisioned: the ability to link addresses related to ransomware families to real world entities, such as exchange services. The main issue with this requirement is that the ability to do this relies on external sources that that keep track of wallets, addresses and who they belong to. Services such as WalletExplorer, GraphSense

[13]https://www.blockchain.com/btc/address/1FLAng62VTeHsPuER2WzHzi5MbemPw48o5

or Chainalysis could provide this information, but they are themselves also reliant on data gathered manually throughout the web or from other third party sources. Because of this reliance, there is no way to know for sure if the information these services gather is complete or fully up-to-date, since a Bitcoin wallet can be constantly evolving and spawning new addresses that are still unknown.

The entire problem statement of identifying real world actors related to Bitcoin addresses could be an extensive research topic on its own. The value of researching this topic would lie in the possibility of also creating a similar, open source solution to what commercial providers like chainalysis provide for their customers. Personally, I think a possible approach to this problem could be through the use of machine learning. Perhaps a machine learning model could be trained that recognizes whether a given transaction is a mixer transaction or a CoinJoin transaction. Likewise, a similar model could, for a given Bitcoin address, identify to what type of entity the address belongs to. Examples of this could be exchange services, online gambling websites, or just relatively small, privately owned wallets. The biggest difficulty in trying to make this machine learning approach work would then be gathering the necessary data to train such a model. Since the qualitative data that is necessary to train on is already scarce in the current climate, this is probably the reason why such a model has not been perfected already in the non-commercial space.

A second possible approach to this problem could be to repetitively, perhaps every day or even every hour, run the already described clustering approach on the entire Bitcoin blockchain. Although this seems like a straightforward approach, applying it in practice is not as trivial as it seems.

The first apparent difficulty this approach poses is the large requirement of computational power. To put this to scale, GraphSense takes around 13 hours to “simply” create an entity graph alone with the very extensive hardware setup their own production environment runs on, as described in section 4.1.1. Clustering the entire blockchain would take a similar amount of time, since the GraphSense entity graph already represents an approximation of what the clustering operation on the entire blockchain would produce.

The second difficulty that poses itself is dealing with the possibilities that the clustering operation wrongly clusters addresses together due to them being co-spent in mixer or CoinJoin transaction inputs. As described in section 3.2.2, detecting these transactions is also not trivial due to the large evolving set of mixer services that are out there.

The existence of these non-trivial difficulties in this seemingly straightforward approach is likely the reason why commercial services like Chainalysis exist and why they thrive. From a forensic perspective, the need to study malicious transactions is not likely to leave any time soon, but the tools necessary to perform these forensics require extensive development and maintenance, which makes for a good business model.

Search functionality

During the testing phase in the development of TRaP it quickly became apparent that it is not trivial to find a substantial set of seed addresses for any given ransomware attack. More often than not when browsing blog posts, forum threads or other mentions of a ransomware attack for ransom notes, seed addresses were either redacted purposely, or completely unused. With an “unused address” meaning they did not have any transactions registered to them, thus rendering them useless to analyse the ransomware attack they are a part of.

Of course, alternative ways to collect seed addresses have also been presented. Other ways are creating synthetic victims by installing the ransomware ourselves and collecting the seed addresses directly from the self-infected hosts. However, the seed addresses collected this way need to have a ransom paid into them in order to become active addresses that participate in later transactions in the ransomware family, which is something that should be avoided if possible in order to not promote the use of ransomware. A third option was to use third party data, which has been done for the validation experiments earlier in this chapter.

The limited range of “good” possibilities to collect seed addresses led to the development of another alternative way: searching seed addresses directly in the blockchain based on a set of known criteria, as was described in section 4.3.2. This functionality was not planned to be implemented at the beginning of this thesis, but was thought of and developed along the way. Unfortunately, this method did not lead to any successful results, and was thus a dead end that was not pursued any further. For now, at least, the only viable ways to collect seed address still remain unchanged.

TRaP's place in the bigger scheme of things

Personally, I think TRaP has proven to fill the unique "selling point" it was designed for: to be a ransomware analytics tool that is as lightweight as possible and is open-source.

Lightweight meaning it does not have any extensive or exotic hardware requirements like GraphSense or other open source crypto-analytics platforms. TRaP can essentially be run as a standalone tool on any laptop or personal computer in combination with a Bitcoin full node and an electrs server. While these last two additional components do require a relatively significant amount of disk space (around 1.3TB combined), they do not require a large amount of available computational power to operate.

While it is lightweight, it also compromises in its feature set compared to a more powerful crypto-analytics platform like GraphSense. GraphSense is a general purpose crypto-analytics platform that can be used to do many types of forensic operations on the Bitcoin blockchain. Besides Bitcoin, GraphSense also has support for Ethereum, Bitcoin Cash, Litecoin and Zcash. In comparison, TRaP compromises by being a tool specifically tailored the analytics of ransomware attacks, leaving out other general purpose features. Because this is specifically what TRaP was designed for, I feel like these compromises do not necessarily detriment the final result that TRaP has become.

The only significant feature I personally think would have been nice to have is the ability to identify real world actors or services behind a given address. This was not possible due to reasons described previously in this conclusion section. Because this feature is still missing, TRaP is not a fully standalone tool yet. TRaP should therefore be used in conjunction with other services or tools available, such as WalletExplorer-like services if they should become more available in the future, the GraphSense demo environment, or perhaps a commercially available service to fill in where TRaP or other free, open-source solutions fall short. When used in conjunction with other services, TRaP could see its full potential as a shortcut for the major parts of analysing a ransomware attack.

Chapter 5

Monero

Until this point in the thesis, everything that has been done was from a Bitcoin centric perspective. There is good reason for this.

Every year, the Financial Crimes Enforcement Network (FinCEN) publishes a report with financial trends related to what they call *Suspicious Activity Reports* (SAR). FinCEN is a United States government bureau whose goal is to prevent and punish criminals and criminal networks that participate in money laundering and other financial crimes. FinCEN is administered by the United States Department of Treasury and operates domestically and internationally [Che21].

According to their report for the first half of 2021 [Fin21], Bitcoin accounted for 5.1 billion USD in transactions that are related to ransomware payments. Other sources state that Bitcoin is representative for approximately 98% of ransomware payments [Mar]. These sources show that Bitcoin is the payment method of choice for the vast majority of ransomware attacks. Alternatively, the FinCEN report also stated that they identified 17 ransomware-related SARs that requested *Monero* (XMR) as a payment method. In some of these 17 cases, the attackers provided both an XMR address as well as a Bitcoin address to pay the ransom on and imposed an extra fee if the payment was made in Bitcoin, to encourage the usage of Monero. In other instances, they exclusively requested the payment in Monero, but accepted payment made in Bitcoin after negotiation. The total value of payments made in Monero amounts to 37 million USD. Compared to the results for Bitcoin, this is around 137 times less, but it is still a significant amount of money that is lost to ransomware attackers.

Why Monero?

One of the currently increasing trends cybercriminals take with ransomware payments is that they incentivise victims to pay their ransom in *anonymity-enhanced cryptocurrencies* (AED). Victims are usually incentivised to use these coins by offering a discount on the ransom. AEDs reduce the transparency of their financial flows through the use of mixing services, as are already discussed, or cryptographic enhancements. The most prominent AEC that is increasingly demanded by cybercriminals is Monero. Because of this, this chapter will take a look into the key differences Monero has with Bitcoin that increase its privacy and anonymity. Secondly, a look will be taken at the limited feasibility of tracing Monero transactions.

5.1 Background

Monero is a cryptocurrency that is built on the *CryptoNote* protocol introduced by Van Saberhagen in 2013 [Van13]. It aims to improve upon three drawbacks Bitcoin poses:

- **Traceability**: Bitcoin allows anyone to follow a trace of BTC due to its completely public and transparent ledger of transactions.
- **Linkability**: In Bitcoin, the recipient address of a transaction is an unambiguous identifier. Because of this, it is possible to infer that two transactions that spend outputs belonging to the same address were issued by the same user.

- **Limited supply**: Bitcoin has, by design, limited amount of BTC that will come into circulation. At some point in the future, Bitcoin miners will stop receiving rewards for mining new blocks in the form of coinbase transactions. When this happens, miners will be rewarded in larger transaction fees. However, there are doubts about the viability of the mining market at this point due to the main incentive to mine blocks being lost.

Monero proposes a concrete solution to each of these problems. The traceability and linkability problems will be addressed with added cryptographic methods that create untraceable, unlinkable and confidential transactions.

The limited supply problem is solved by simply setting a lower bound on the block reward. This way, miners will always be rewarded for their work in the form of coinbase transactions.

Stealth Addresses

Bitcoin suffers from *linkable transactions*. What this means is that when two transactions have the same recipient address, they certainly belong to the same wallet, and thus also the same real world entity. Bitcoin users can partly avoid giving away this information by issuing a new address for each new transaction they will be the recipient of. When spending the currency received in these different addresses, they have to be careful not to combine outputs belonging to different addresses as input to the same transactions to avoid giving away that both addresses belong to the same wallet. Third parties could discover this information using the multiple-input spending heuristic discussed in section 3.2.1. These linkability traits have already been "exploited" by techniques explained in previous chapters of this thesis.

Monero addresses these linkability traits using *stealth addresses* [18]. Stealth addresses specifically are meant to protect the identity of transaction recipients. To avoid recording the recipient's wallet address onto the Monero blockchain, each transaction is instead sent to a one-time, disposable address. It is the responsibility of the transaction sender to calculate the stealth address by deriving it from the recipients public address. The recipient will always be able to access the funds sent to the stealth addresses that are derived from its own public address, without exposing links to their wallet's public address or other transactions they are involved in. This way, third parties observing the blockchain will be unable to determine to whom a transaction is directed, while the involved parties (sender and recipient) are the only one that know.

Ring Signatures

While stealth addresses take care of the privacy of a transaction's recipient, it does not protect the senders privacy in any way. To protect transaction senders, Monero uses *ring signatures* [18].

In transparent cryptocurrencies, like Bitcoin, when a new transaction is created, the sender of the transaction digitally signs their new transaction before broadcasting it to the network. This is necessary for the sender to prove that they are the owner of the UTXO they are spending as input of the new transaction. If a sender would try to spend the same UTXO twice, other nodes in the network would immediately notice when trying to verify the new transaction because the UTXO has already been spent and signed in a previous transaction. This stops Bitcoin users from spending the UTXO they own more than once. This concept is called *proof-of-ownership*. However, the way Bitcoin approaches proof-of-ownership is detrimental to the privacy of the sender of the transaction by explicitly indicating the source of the UTXO that are being spent, and when an UTXO is spent.

Monero approaches proof-of-ownership differently, by using *ring signatures*. When a new transaction is created, for every transaction input, the transaction sender semi-randomly drafts a collection of past transaction outputs from Monero's transaction history that do not belong to the same sender. The sender will then mix the public keys associated with these inputs into a ring with its own transaction input as *decoys*. The sender can then sign the transaction, but it can do this in a way that other parties in the network can correctly verify the transaction without giving away *who* signed the transaction. This proof-of-ownership scheme obfuscates which user created the transaction, as well as which outputs are being actually spent in the new transaction.

This also raises additional difficulties. Since it is impossible to tell if a transaction output is actually spent as input to the new transaction, or if it is just a decoy, what prevents a malicious user from spending the same transaction outputs multiple times? In a transparent cryptocurrency like Bitcoin this is trivial.

Any output that has been signed and spent once cannot be spent again. In Monero, transaction outputs can validly appear as transaction input as part of a ring signature multiple times.

Allowing spending of transaction outputs multiple times can be avoided using *key images*. A key image is generated and included in each new transaction. Key images are uniquely derived from the transaction output that is *actually* being spent in the transaction. Think of it as an irreversible hash of the transaction output that is being spent. Other nodes in the network cannot derive which transaction output in the ring the key image is derived from. When a malicious user tries to spend the same output multiple times, the same key image will also be included in the transaction. When this occurs, other nodes will reject the new transaction since the key image has already been seen in a past transaction.

RingCT

Stealth addresses and ring signatures are mechanisms that respectively protect the privacy of transaction recipients and transaction senders. They prevent users from being identifiable in a transaction. The final remaining part of a transaction that is not yet private is the content of a transaction: the amount of Moneroj[1] that is being spent. To hide this information, Monero uses *ring confidential transactions*, or *RingCT* [18] in short. In transparent cryptocurrencies like Bitcoin, the public visibility of a transaction's value has made it possible in the previous chapter to calculate a lower bound financial impact of a ransomware attack, and to determine a payment trend over time for a given ransomware family.

Because the specifics of RingCT are heavily reliant on a cryptography background, this section will try to give an abstraction of the ideas used in RingCT that allows the reader to understand the general concepts behind it.

RingCT is built on the idea of *commitment* and *range proofs*. Commitment means that the transaction sender commits a specified amount of their funds to be sent in a way that miners, and other nodes in the network, can verify the legitimacy of the transaction without publicly disclosing the amount of the transaction. Conceptually, this can be done simply by taking the transaction amount and multiplying it with a random number. For example, if Alice wants to send Bob 4 Moneroj, Alice generates a random number A that she multiplies the funds in the transaction with. To all other nodes, the transaction amount will be visible as A times 4 Moneroj. Other nodes can verify the validity of the transaction by making the sum of all inputs and outputs of the transaction. For the example transaction, this sum would be $-4 * A + 4 * A = 0$. If a transaction sum does not equal zero, it either means a malicious user tried to create new Monero from thin air, or the sender did not create enough transaction outputs to match the transaction's inputs. If this is the case, the transaction will be rejected by the miner. Alice and Bob also agree on a shared cryptographic secret. Because only Alice and Bob known this secret, they can share the value of A amongst each other and use this to both know the actual transaction amount.

Secondly, an important mechanism used in RingCT are *range proofs*. Simply put, a range proof is another check done by the miner to verify the committed amount of funds by the sender is greater than 0 and smaller than a certain number. This is to prevent sender from committing negative or impossibly high amounts of Moneroj.

Attacks on privacy in Monero

Now that the concepts stealth addresses, ring signatures and RingCT are explained, a summary of the bigger picture is given in figure 5.1.

Monero takes a three-pronged approach to create a fully private payment system. The measures Monero takes to improve its users; and transactions' privacy make it significantly more difficult to trace transactions that are made in the Monero blockchain, compared to Bitcoin in the previous chapters. Of course, this also impacts the traceability of transactions made to ransomware families. The continuation of this chapter will take a deeper dive in what types of tracing might still be possible in the Monero blockchain, and how this would work in practice. In research done by Kumar et al. [Kum+17], four "attacks" on the Monero blockchain are presented that can aid in circumventing some of the privacy measures taken by Monero and allow minimal tracing of certain transaction properties. In [HH18], Hinteregger et al. have applied two of these methods and published the results they achieved in practice.

[1]The English noun "coin" translates to "Monero" in Esperanto. "Moneroj" is the plural of this.

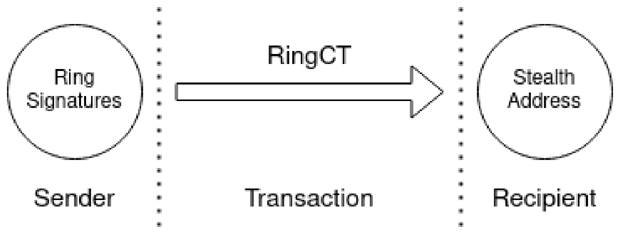


Figure 5.1: A graphical summary of the measures that are taken by Monero to enhance its privacy. Ring Signatures are specifically meant to protect the transaction sender's privacy by involving other users as "decoy senders" in a transaction, while stealth addresses are used to protect the transaction recipient's privacy through the use of generating new, one-time throwaway destination addresses. RingCT is meant to allow confidential transactions, by hiding the actual transaction amount while still allowing valid verification methods by miners or other nodes.

5.2 Zero Mixin Removal

The first attack presented by Kumar et al. is called *zero mixin removal*. The key idea behind this attack is to leverage the existence of zero mixin transactions. A zero mixin transaction is a transaction that uses a ring of transaction inputs without any decoy mixins. In other words, only the sender's own transaction input is present in the ring for the ring signature procedure. The ring size of a zero mixin transaction is one.

Originally, ring signatures were not mandatory in the Monero network [18]. These early transactions then showed the same privacy weaknesses as other transparent cryptocurrencies by publicly identifying both sender and receiver. In 2016 ring signatures became mandatory for all new transactions with a minimum requirement of two ring members per ring signature. Since then, the required minimum amount of ring members has grown almost yearly to a minimum of 16 at the time of writing this text [2].

Zero mixin transactions cannot be created anymore since 2016, but the existence of pre-2016 zero mixin transactions still has a cascading effect on the effectiveness of current day ring signature transactions. Suppose a pre-2016 zero mixin transaction exists, called TX1. This transaction has only one transaction input. The input key in this transaction's input can then certainly be categorised as *spent*, since it is the only input. Suppose then another transaction exists, let's call it TX2, that uses the input key from the input from TX1 as a decoy mixin. If TX2 has a ring size of two inputs, it can be said with certainty which of the two ring members in TX2 is the real transaction input and which is the decoy, since the decoy is already categorised as *spent* by looking at TX1. The true transaction input can in this case also be categorised as *spent*.

The attack can be generalized. If the number of mixins used in TX2 is $m > 1$, then the effective anonymity-set size becomes m, instead of the desired $m + 1$. The "effective anonymity-set size" is the size of the set of ring members that cannot be categorised as *spent* by looking at previous transactions in the Monero blockchain. Figure 5.2 schematically shows a possible cascading effect of de-anonymized ring members triggered by a single zero mixin transaction.

Evaluation

In [Kum+17], Kumar et al. measured the success of their zero mixin removal attack by applying it to all transactions in the Monero blockchain. The results they obtained are plotted in figure 5.4. To summarize, they were able to trace the inputs of 87.9% of all Monero transactions up until somewhere in 2016, when the paper was written. The exact date up until they ran the attack was unfortunately not specified.

In retrospect, the attack being able to trace the real transaction inputs of over 87% of transactions until that point in time in 2016 is quite significant, especially for a privacy coin that claims to take increased privacy measures compared to transparent cryptocurrencies. This high accuracy is explainable however. Until January 2017, ring signatures were not mandatory in Monero transactions [Devc], they were merely optional. Before this update, many of the transactions that were created did not use ring signatures. This means all of these transactions were trivially traceable. In the current day, anno 2022,

[2]https://github.com/monero-project/monero/pull/8178

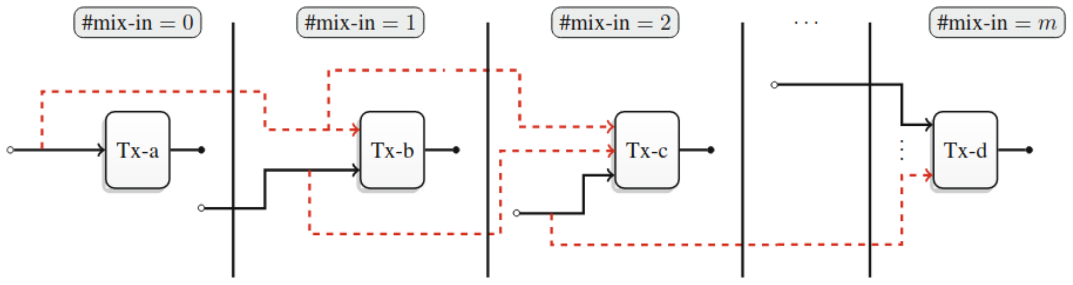


Figure 5.2: Visual example of the cascading effect triggered by zero mixin transactions. Each transaction has only one "real" input. The number of mixins used increases from left to right. Dashed lines represent input keys already categorised as *spent* based on information from earlier transactions. Lines in bold represent the "real" inputs being spent. Image source: [Kum+17]

ring signatures have been mandatory for almost 7 years. One can reason this also influences the amount of traceable transactions created following the update that made ring signatures mandatory, especially as the mandatory ring size has grown over the last few years to the current mandatory ring size of 16 ring members. This influx of sizeable ring signatures in Monero transactions means less and less of "modern" transactions should be traceable. This claim can be backed up by research done by Hinteregger et al. in [HH18]. They have performed the zero mixin removal attack once again, now until August of 2018.

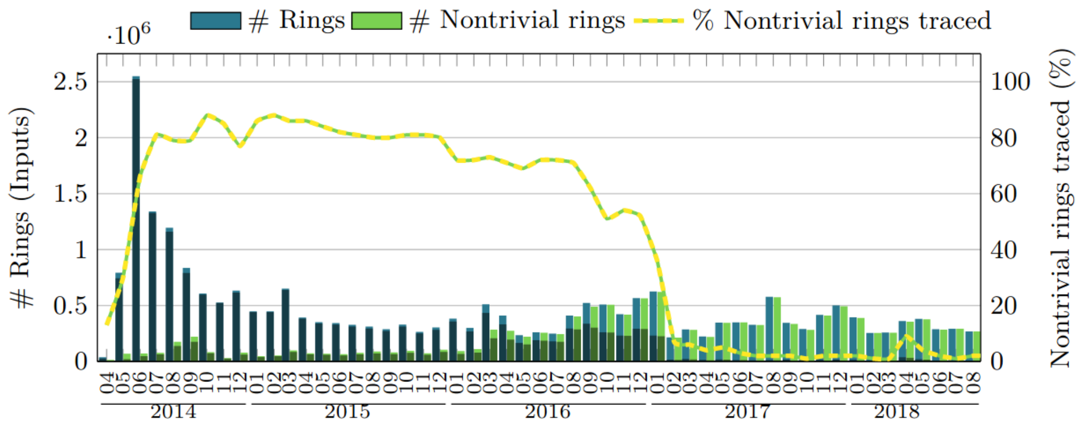


Figure 5.3: Chart showing monthy number of ring signatures, nontrivial rings and the percentage of nontrivial rings that can be traced. Once ring signatures become mandatory in 2017, the percentage of traceable ring signatures quickly decreases, reaching almost zero. Image source: [HH18]

Figure 5.3 shows a visualisation depicting the amount of ring signatures present in Monero transactions, the amount of nontrivial rings, and nontrivial rings that can be traced. A ring is deemed nontrivial if it has more than one ring member. The figure shows that since January 2017, when ring signatures became mandatory, the proportion of traceable rings quickly declined, reaching almost zero by the end of 2018. The bump in traceable rings that is visible in the spring of 2018 can be attributed to a hard fork that was happening at the time between Monero Original and MoneroV, according to Hinteregger et al. The reason they were able to trace some of the rings in this timeframe was due to their outputs being traceable in the forked blockchain. The bump in traceable rings is thus due to a cross chain analysis between Monero Original and MoneroV.

5.3 Output Merging Heuristic

The second attack on Monero's privacy builds on the assumption that when a transaction is created, it is unlikely that two or more mixins in a ring of transaction inputs will be the output of the same previous transaction [Kum+17]. It can be assumed this would be an unlikely case since the set of mixins in a ring is chosen completely at random from previous transaction outputs in the blockchain. If the situation

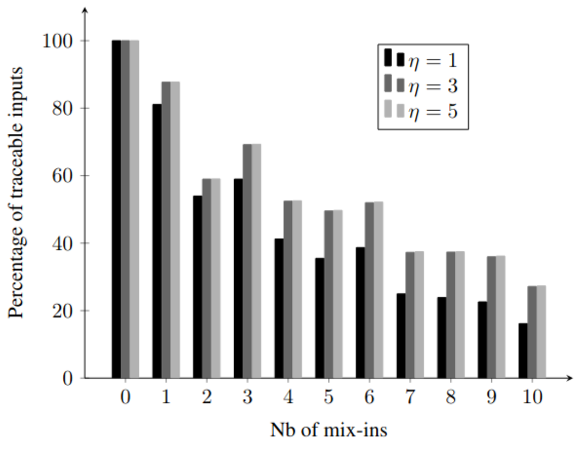


Figure 5.4: Results of the zero mixin removal attack applied to all transactions in the Monero blockchain as published in source: [Kum+17]. η represents the amount of iterations they ran their attack over the blockchain data. Running the attack multiple iterations creates new information that can be used in the next iteration. η stops at three iterations because there is no difference anymore between results of the second and third iteration.

would occur that a transaction input contains two or more mixins than there are outputs of the same previous transaction, then they are likely to be the real ones being spent, instead of being mixins. A visual example of this situation is given in figure 5.5. Let's call this attack the *output merging attack* [Mös+17].

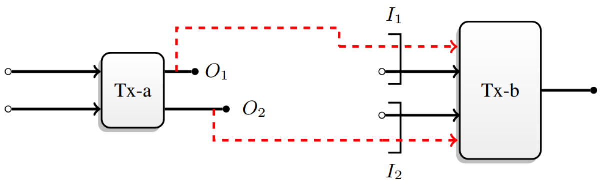


Figure 5.5: Visual example of an attack leveraging output merging in a Monero transaction. Tx-A is a transaction two outputs, O_1 and O_2. Tx-B is a later transaction with two input rings, I_1 and I_2. Each input ring has ring size 2, so they have 1 mixin and 1 real input each. Both I_1 and I_2 include outputs from the same previous transaction: Tx-A. According to the output merging attack, O_1 and O_2 are likely to be the real transaction inputs to Tx-B, since it would be very unlikely they would both be randomly chosen as mixins. Source: [Kum+17].

An important note to emphasize in this attack is that it is a *heuristic*. It is not guaranteed to produce true positives exclusively. The validity of the attack is based on the fact that members in a transaction input ring are chosen at random, which makes it unlikely, but not impossible that multiple outputs of some transaction are used as mixins in the same input ring of a new transaction. In order to further elaborate on this uncertainty, four situations that can occur when using this attack will be sketched and discussed. In these situations, a *source* is a transaction like "Tc-A" in figure 5.5. A *destination* is a transaction like "Tc-B" in figure 5.5. It uses transaction outputs from a source address in one of its ring inputs.

1. For a given source, it can occur that it has no destination. In this case, the attack will fail to yield any results for the transaction outputs of the source.
2. For a given source, it might find several different destinations. In this case, it is hard to say anything about the transaction outputs of the source. Since none of them occur together as transaction inputs in a destination transaction. Uncertainty remains in this case.
3. For a given source, one or more different destinations might exist, where the same source output appears in more than one destination input. This would be the worst case scenario, since not only is the previous scenario valid, but there is also uncertainty as to which destination input might actually be spending the source outputs, if they are spending them at all.

4. For a given source, one or more different destinations might exist, where more than one source output appears in a single destination input. In this case, the attack yields a set of candidate keys, one of which is likely to be actually spent in the destination transaction.

Evaluation

To determine the effectiveness of the output merging attack, Kumar et al. compared the results obtained by this heuristic to the results produced by the previous zero mixin removal attack. Since the zero mixin removal attack exclusively produces true positive results, the results of that attack can be used as ground truths for a comparison with the new output merging heuristic attack. In order to compare both attacks, the following terms have to be formalized:

- True Positive: the output merging heuristic produces a true positive result if it categorises a transaction input key as *spent* and the zero mixin removal attack also categorises the same transaction input key as *spent*. In summary, both attack produce an identical result for a given transaction input.
- False Positive: The output merging heuristic produces a false positive result if it categorises a transaction input key as *spent*, but the zero mixin removal attack categorises the same transaction input as *spent* in a different transaction than the output merging heuristic. In summary, the attacks are not in agreement about which transaction really consumes the given transaction input.
- Unknown positive: The output merging heuristic produces an unknown positive results if it categorises a given transaction input as *spent*, but the zero mixin removal attack does not. This is an uncertain case, while the zero mixin removal attack produces ground truth results, it does not give outputs for all inputs.

In summary, Kumar et al. showed that the output merging heuristic has a true positive rate of 87%, a false positive rate of 0.78% and an unknown positive result for around 12% of inputs. Something else that became apparent in their testing was that as the amount of mixins for a transaction input increases, the percentage of true positive results decreases, while the amount of unknown positives increases. The amount of false positives remained close to zero. An explanation for this could be that as the amount of mixins increases, the effectiveness of the zero mixin removal attack decreases, which makes it so less ground truth results are produced to compare the current attack with. It does not necessarily mean that the current output merging attack is less effective when the amount of mixins increases, there is just not a conclusive way to prove this due to lack of ground truth results.

5.4 Temporal Analysis

The third and final attack that is introduced in [Kum+17] builds on the straightforward idea that a given transaction output does not remain *unspent* for an infinite amount of time. Generally, for any given transaction output, the probability of the transaction output being spent again continually increases with time. Utilizing this idea, Kumar et al. formulate the following attack strategy: “Given a set of input keys used to create a ring signature, the real key being spent is the one with the highest block height”. Like the previous output merging heuristic attack, this is also a heuristic. It is not guaranteed to produce only ground truth results.

Evaluation

Using the results of the zero mixin removal attack again as ground truths, Kumar et al. have shown that the temporal analysis attack has a true positive rate of 98.1%. This shows that that the temporal analysis attack can in fact be very accurate, yet so simple.

Similar to the zero mixin removal attack, Hinteregger et al. have also executed the temporal analysis attack again in a more recent “Monero” climate. The result of their experiment is shown in figure 5.6. Conclusions that can be drawn from their experiment are that the accuracy of the attack has significantly declined since the attack was proposed in 2016. The decrease in effectivity can be explained. Throughout 2016, 2017 and 2018, several updates to Monero were released that changed the age distribution of transaction outputs that was used to randomly select decoy ring members. The initial distribution was a uniform distribution. This was later changed to a triangular distribution, and eventually settled on a gamma distribution by August 2018 [HH18]. The final distribution showed to be the most effective at countering the temporal analysis attack.

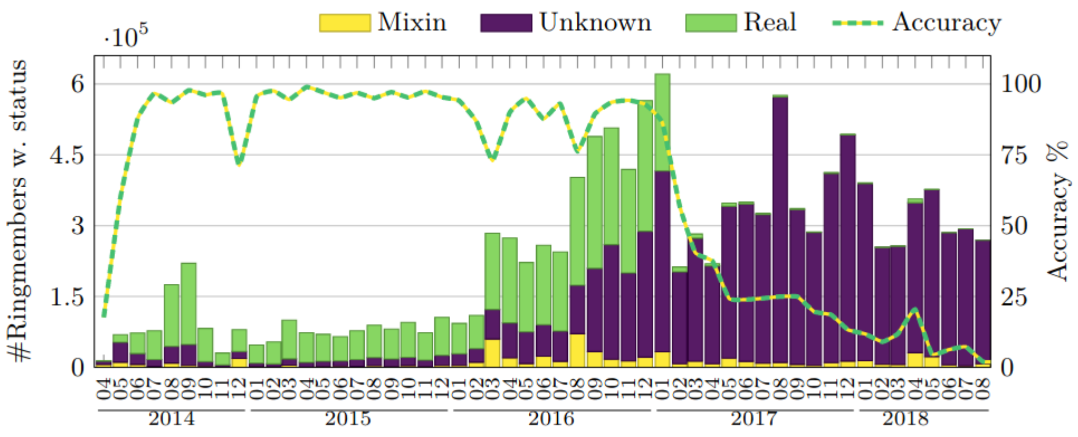


Figure 5.6: Chart visualising the results of the temporal analysis attack by Hinteregger et al. It depicts per month, for each ring signature, the amount transaction inputs that are identified as mixin, as real, or as unknown. Most importantly, the dotted line shows the accuracy of the attack over time. Since 2017, the accuracy of the attack strongly declines. Image source: [HH18]

5.5 Impact on traceability of ransomware attacks

The current chapter has thus far provided an overview of the differences or improvements Monero makes compared to a transparent cryptocurrency, like Bitcoin. These improvements are made in order to try and increase privacy and anonymity of users and their transactions within the Monero blockchain. Furthermore this chapter has also given an overview of three known attacks on the Monero blockchain, and their impact on the integrity of Monero's privacy. The purpose of this section is to now, based on this knowledge, discuss what impact these additional privacy measures and their weaknesses have on the traceability of tracing ransomware transactions in the Monero blockchain. The discussion will be made by using the tracing process described for the Bitcoin blockchain in earlier chapters as a starting point and studying what parts of this process can or cannot be extended to the Monero blockchain.

5.5.1 Clustering

The first and most immediate step of the ransomware tracing process throughout the Bitcoin blockchain was the ability to use the multiple-input spending heuristic to expand a collection of gathered seed addresses to a larger set of addresses, thus increasing the breadth of the tracing process. In Bitcoin this is possible because all transaction inputs and outputs are transparent to everyone. In Monero however, transaction inputs are obfuscated through the use of Ring Signatures, as explained in section 5.1. Ring Signatures hide the real transaction input to a transaction by combining it with other decoy transaction inputs, randomly selected from the blockchain, without giving away which transaction input is the real one.

At first glance, it seems clear that Ring Signatures make it impossible to use the multiple-input spending heuristic to cluster a set of given seed addresses. Since every transaction is obfuscated, there is no way to know which addresses can be clustered together, making the heuristic useless. From this, the question arises if any of the known attacks on the Monero blockchain can offer resolution in this regard. Two of the three attacks come to mind: the zero mixin removal attack and the temporal analysis attack.

To determine the feasibility of using the multiple-input spending heuristic in a Monero context, we can try to determine the steps a hypothetical, TRaP-like tool would have to take to be able to successfully run the heuristic.

Recap: Multiple-input spending heuristic

For every given seed address, the tool has to make a lookup operation to obtain all transactions the seed address occurs in. Next, a cluster is created comprised of the seed address and all addresses that are co-spent as a transaction input together with the seed address. The same is done for all addresses that are added to a cluster in this way, until the set of known clusters does not change any further. Finally, all clusters that have an overlap of at least one address are merged together. The resulting set of clusters should have no clusters left that overlap.

Suppose now the hypothetical tool tries to apply this heuristic on the Monero blockchain. Obtaining transactions for a given Monero address could be possible through the use of an Electrum server variant for Monero. Next, the tool has to able to identify all transaction inputs into the transactions it observes. The tool will not be able to do this due to Ring Signatures. However, the tool could try to make use of the two aforementioned attacks.

Using the zero mixin removal attack

Assume the zero mixin removal attack is used to help identify real transaction inputs in order to be able to use the multiple-input spending heuristic. In the context of such an analytics tool, the attack could be used to build a small local database of transaction inputs. In this database, for every transaction input on the Monero blockchain, a status is stored. The status of a transaction input can be either *spent* or *unspent*. To populate this database, the zero mixin removal attack has to be executed once on the entire Monero blockchain, starting from the genesis block, as described in section 5.2. To keep the database up-to-date, the database can be updated each time a new block of transactions is added to the blockchain, using and building on information that is already in the database.

Suppose now the hypothetical tool observes a transaction and tries to determine the real input for any of the transaction's input rings. It can do this by consulting the locally built database it created using the zero mixin removal attack. Three cases can occur.

1. All transaction inputs in the ring signature are marked unspent. Nothing can be said about the real transaction input.
2. Two or more transaction inputs in the ring signature are marked unspent. Nothing can be said about the real transaction input. The other transaction inputs in the ring will be marked as spent, but they do not contribute anything to the traceability of the ring.
3. Only one transaction input in the ring signature is marked as unspent. According to the zero mixin removal attack, this transaction input is the real one. The others are decoy.

In order to be able to start building a cluster, two or more ring signatures have to adhere to case number 3, since this is the only case in which a real transaction input can be identified. As explained in section 5.2 and figure 5.3 specifically, the likelihood of this happening is almost zero, hence the zero mixin removal attack does not offer any hopes to be able to implement an effective clustering heuristic for the Monero blockchain. As figure 5.3 also shows, the attack could in theory contribute to the clustering approach if the transactions that are observed occurred before January 2017. However, The FinCEN report [Fin21] stated that only recently (2021 and onward) Monero became prominent as a ransomware payment method. For "modern" ransomware attacks, clustering is not feasible anymore.

Using the temporal analysis attack

Alternatively, perhaps the temporal analysis attack can be used to identify real transaction inputs in a ring signature. The approach to use this attack would be simpler in comparison to using the zero mixin removal attack. Essentially, when a transaction is observed, the "real inputs" in the transaction's input rings can be identified by simply considering the newest ring member as the real input. No local database of any kind has to be built and maintained.

Unfortunately, the effectiveness of the temporal analysis attack was also severely limited for transactions occurring after August 2018, as was explained in section 5.4, and visualized in figure 5.6. For ransomware attacks occurring before August 2018, the temporal analysis attack could theoretically still be used to identify real transaction inputs in ring signatures, although the output of the attack is still in the realm of uncertainty. The temporal analysis attack is still only a heuristic and does not produce true positive results in every case. To use this attack in a real clustering strategy would create clusters with increasing unlikelihood of being completely correct clusters as they grow in size, since every new addition to the cluster has some uncertainty associated with it. The probability that a given cluster generated using this approach, based on the temporal analysis attack, is comprised exclusively of true positive transaction inputs that are effectively all real transaction inputs is the size of the cluster multiplied by the accuracy ratio of the attack at the point in time when each transaction occurred. The accuracy ratio varies over time as was visualized in figure 5.6.

5.5.2 Collector addresses

After clustering, the next critical step in tracing ransomware payments in a transparent blockchain like Bitcoin is the ability to detect collector addresses, especially for large ransomware families. In Bitcoin this could be done by building a graph, where nodes represented addresses and edges represented transactions between those addresses. The collector addresses could then be queried using a specific graph query. Bitcoin allows for the interpretation of its transactions in the form of a graph data structure since every address is fully transparent. Any incoming or outgoing transaction of an address is completely visible to third parties, which allows the construction of a graph.

In Monero, incoming and outgoing transactions to a user are hidden through the use of ring signatures and stealth addresses. In the previous section about clustering, the effects of ring signatures on traceability have already been discussed, as well as the attacks that exist specifically targeted towards de-anonymising ring signatures. Unfortunately, the conclusion of that section was that nothing significant can be done to increase traceability of ring signatures in the current day Monero climate, that we know of.

No attacks are currently known to exist that can de-anonymise the stealth address. In theory, such an attack would entail the ability to derive a user's Monero address from a given stealth address. In order to do this the cryptographic procedure that is used to securely generate a stealth address from a given Monero address would have to be attacked. No attacks are known to this day that are able to do this.

From the inability to trace neither a transaction's input, nor its output can be concluded that Monero does not allow the construction of a transaction graph, as was possible in Bitcoin. By extension, this also avoids any possibility to calculate a collection of collector addresses.

5.5.3 Estimating financial impact

Next to clustering and the ability to calculate collector addresses, Bitcoin allowed the possibility to estimate the financial impact of a ransomware attack. To make this estimation in Bitcoin, the sum of incoming transactions could be calculated, as well as the trend of incoming ransom payments over time. Especially the payment trend was a useful metric to gain insight into a ransomware cluster. It could help determine if a ransomware family operated as a one time operation with an extremely viral campaign (WannaCry), or if it operated as a more discrete attacker, striking multiple times over several weeks or months (SamSam).

In Bitcoin these estimations were made possible by the transparency of transaction contents. The amount of Bitcoin that is moved from one address to another is always publicly visible in Bitcoin. In Monero, the amount of Moneroj that is moved is hidden behind Ring Confidential Transactions. There are no known attacks that result in giving away the content of a Monero transaction, thus inhibiting the possibility to make an estimation of the financial impact directly from the Monero blockchain.

There are be some possibilities however. Suppose for example a ransomware attack that operates like the Deadbolt ransomware attack that was earlier described in section 4.3.2. An attack like this asks the same fixed ransom from all of its victims. If a collection of seed addresses is known beforehand, strictly speaking an estimation of the total amount of Moneroj going in to the attacker's address can be made by multiplying the fixed ransom by the amount of known seed addresses.

A payment trend analysis can also be estimated. Since the transaction timestamp of a transaction is not hidden, a payment trend line can be drawn for a given set of seed addresses using the fixed ransom for each incoming transaction into a seed address, and the timestamps of when these transactions occurred.

5.5.4 Uncovering real world actors

One major aspect of tracing ransomware families in Bitcoin that was left partly unresolved by TRaP was the ability to uncover real world actors or entities behind a given Bitcoin address. If possible, this would have opened the possibility to identify in which specific (exchange) services the money collected with a ransomware attack would be exchanged to fiat currency. Monero adds even more difficulty to this problem, due to the use of stealth addresses. In the previous subsection on clustering in Monero was already explained why stealth addresses are not reversible to Monero addresses. Even if this reversal would be possible, Monero would still suffer from the same problem Bitcoin does with respect to identifying real world actors or entities.

5.6 Conclusion

In this chapter, an overview was given of the additional measures Monero takes over other, transparent cryptocurrencies in order to increase its users' privacy and anonymity. Then, three known attacks on the Monero blockchain were discussed that attempt to primarily de-anonymise transaction inputs, increasing the traceability of transactions in the Monero blockchain. These attacks were proven successful in the past, but they have been caught up with by the Monero developers, meaning their useability in tracing modern day transactions is almost non-existent.

The purpose of this chapter was to discuss Monero as a medium for ransomware payments, and the impact it has on the traceability of these ransomware attacks, compared to Bitcoin. The conclusion that can be drawn from this chapter is that the significantly increasing use of Monero in ransomware families, as stated in the FinCEN report [Fin21], and other cyber crime activities is certainly justified. Almost none of the conventional tracing mechanisms that work in Bitcoin, and most likely all transparent cryptocurrencies by extension, are transferable to Monero. This means cyber criminals that use Monero for their malicious activities are indeed significantly better protected from digital forensics operations compared to when they use a transparent payment method like Bitcoin, decreasing their chances to get caught.

From the perspective of user privacy for cryptocurrency owners, this is of course great news. Monero proves to be a solid privacy coin. However unfortunately, as with many significant technologies or inventions in the past, what is designed for the greater good, will eventually be misused for the wrong reasons.

Chapter 6

Finishing Thoughts

6.1 Conclusions

Roughly one year ago from the time I am writing this conclusion, I started formalising the topic I wanted to study in my master's thesis. The general idea back then was to study the traceability of cryptocurrencies and see how this could be applied within the context of tracing ransomware attacks specifically, since they are one of the prominent forms of (cyber) criminality that rely on cryptocurrencies to increase the criminals' own anonymity. From this initial idea stemmed 2 primary research questions, as were posed in the introductory chapter.

In the initial weeks of executing this thesis, I naïvely assumed the steps I needed to take in order to answer these questions would be relatively straightforward. To the question "Is it possible to trace ransomware related payments across a cryptocurrency blockchain?", I automatically assumed the answer would be a resounding and simple "*yes*". As time went by, it quickly became clear the real answer would be a bit more nuanced.

The first sub question that was posed was *what exactly it would entail to trace ransomware payment throughout the blockchain*? In order to formulate an answer to this question, a practical approach was taken in the form of the development of TRaP. Choosing to develop a tool that tries to calculate insightful information into a ransomware attack pushed me to properly understand the organizational aspects of ransomware, as well as cryptocurrencies and their underlying blockchain technology. Without understanding these concepts properly, it would be next to impossible to properly create a tool like TRaP. Almost everything I thought I knew about blockchain technology turned out to be only an abstraction of reality. Developing TRaP also meant I had to fully understand every step that was taken throughout the tracing process, and being able to understand why these steps were necessary within their own context. From a learning perspective, choosing this approach was probably the best choice I could have made, since it offered hands-on experience with the problem statement at hand.

What the approach also did was offer first-hand insight in hard to tackle topics. This brings us to the second sub question: "What are the pitfalls? What is not possible?". The main conclusion in this regard that can be drawn from the research done with TRaP is that there is an inherent difficulty in tracing transactions throughout the Bitcoin ecosystem due to the existence of anonymity enhancing services or transactions. The inability to correctly identify a given transaction as being a mixer service in turn hinders the ability to say anything with certainty about transactions and addresses that follow it. This is a field where I believe some future work can still be done. Currently, no general method exists that is able to make these identifications that is not specifically tailored to only one mixer service. Perhaps some form of machine learning could be used to tackle this problem in the future, however this would require a representative training data set which is also not trivially available or created. I'm personally also not sure if such a method would be possible at all. Perhaps some future work could run a feasibility study on this topic.

A second feature that would have been great to have implemented is the ability to identify real world actors or services related to a given Bitcoin address. The ability to do this would mean that law enforcement operators could easily track where ransomware money is flowing and contact these services appropriately in order to counter the exchange of ransomware money to fiat currency. This could also be a topic for future work. Currently, commercial services like Chainalysis [1] partly offer this feature, but no open-source or free alternatives exist that are completely up-to-date with the present day Bitcoin blockchain. Part of the reason for this is that collecting this data, curating it and keeping it up-to-date required significant amounts of resources and man hours. This is most likely also why only commercial services offer this feature, since it makes a good business model.

So what *can* be done then? This formulates an answer to the third sub question: "Which insight can be gained from tracing ransomware related payments?". While the anonymity enhancing services hinder some of the intended features of TRaP, there is still plenty that TRaP can analyse and produce results on. TRaP can be used as a tool to quickly determine the financial impact of a given ransomware family, determine how viral a ransomware campaign is through determining payment trends, determining collector addresses and identifying new seed addresses previously undiscovered. While this feature set is more limited than initially intended, I feel like I have learned the most from the features that were not able to be implemented, as they posed the real challenge and pushed me to think thoroughly about the problem statement.

A final conclusive thought must also be given to Monero, being a privacy coin. Monero has been gaining more and more popularity within the ransomware scene over the last few years. I think, with this thesis, that we have proven that this increase is justified. Monero takes relatively intuïtive measures to increase its anonymity and protect which information it exposes to third parties. Due to these measures, third party analysts like ourselves can transfer almost none of the useful tracing techniques that could be used on Bitcoin to Monero. Apart from a very limited and very minimal lower bound financial impact estimation, nothing much can be said about ransomware related payments across the Monero blockchain, because the measures Monero takes prohibit extraction of this information. Unfortunately, this doesn't bode well for future ransomware victims, as cyber criminals will be more protected and harder to find than ever before thanks to privacy coins.

6.2 Reflection

Throughout my earlier years in my education at this university, for some reason I often wondered what it would be like to write a master's thesis. How does one determine a topic? How does one navigate the challenges that present themselves along the way? Will I have the necessary skills by then to bring this final, large scale, independent project to a successful conclusion? *Will I succeed*? The task always seemed daunting to me to say the least. It cannot be compared to any course, project or exam that came before it. After all these years, I can finally answer some of these questions, and reflect on my experience.

I feel like I can divide the past thesis year into two main halves. The first half, going from October up until halfway in January I felt like I was metaphorically afloat on a raft in the middle of an ocean, not knowing how to navigate my way back to land. My plan at the beginning of October was to delve into all the background information I thought I needed to understand to be able to progress. Along the way, I planned to start developing a tool that could "trace ransomware payments". However, this is of course a very ambiguous and not well defined goal to start developing a tool. I remember reading literature and being quite intimidated by the results produced in existing research. I knew I probably had to follow a similar strategy in the development of my own techniques, but at that point I had absolutely no idea how to get there from where I was. This went combined with another struggle I had at that time. In order to analyse anything in the Bitcoin blockchain, or any cryptocurrency for that matter, I needed access to the blockchain data. In retrospect, solving this problem was not inherently hard or anything, but getting to the point of discovering services such as the Bitcoin RPC and the Electrum server took more exploration and digesting the necessary information than I expected at that time. The cryptocurrency landscape is currently flooded with sources on almost anything related to it, both good and bad. I learned that sometimes it takes time just to digest everything and select what is useful and throw out the rest. Looking back on it now, I know what could have gotten me to my goals quicker, but starting from scratch this did not feel like an easy feat. Many frustrated hours and sleepless nights have been spent in these first couple of months, with fears of not even being able to finish the thesis. Luckily, I understand now that all of this is part of the experience and moments like these teach you the most valuable things in the end. To be patient and to push through.

Luckily the tides changed halfway of January. At this time I started (slowly) working up sections and

[1] https://chainalysis.com

paragraphs in this text. I also managed to setup everything I needed to quickly access the data I needed in the development of TRaP, going from upgrading my computer's disk requirements, to setting up all the environments I needed. These weeks formed the seedbed for the rest of the thesis. At this time I also realised that I had in fact been making progress in the first few months as well. I properly understood what I needed/wanted to implement in TRaP, thanks to all the research I had already done. I also quickly figured out what I wanted the thesis to roughly look like in the end. The thesis had clear goals I wanted to meet and there was finally a general sense of direction. The second half of the year was by far the most productive. New material was produced each week, be it in the form of code, experimental results, or additions to this text. My morale went up significantly compared to the first few months.

Something else I learned throughout writing this text is the difficulty that is writing a good, qualitative, well-structured text. I gained an immense respect for everyone out there that writes scientific material, or any written material for that matter, in a clear, concise and understandable way. I tried my best attempt at this as well, but writing a qualitative text has proven to be a very tedious, time-consuming and challenging process.

If I were to start this thesis again from the beginning, I am not sure if I would make any significant changes. The struggles I had gave me the most valuable lessons. Taking these away would probably detriment the satisfaction I feel when overcoming these struggles and powering through.

To conclude, I feel like this year has been a bumpy ride. However, if I could now talk to my younger self, pondering what the thesis would be like, I think I would tell him not to worry. Things might seem challenging at times, but it is all part of the journey. A bit of determination and a bit of confidence in yourself goes a long way. Enjoy it while it lasts, because it is a tremendous learning experience.

Appendix A

Nederlandse Samenvatting

A.1 Inleiding

De dag van vandaag staan bijna alle digitale toestellen die we gebruiken in verbinding met het internet. Ondernemingen, grootschalige bedrijven alsook individuen zijn dagelijks afhankelijk van digitale toestellen, zij het voor persoonlijk gebruik enerzijds, of voor het werk anderzijds. Deze grote afhankelijkheid van deze toestellen maken hun een gegeerd doelwit voor cyberaanvallen. De meest lucratieve vorm van cyberaanvallen die de afgelopen jaren sterk aan populariteit wint is *ransomware*.

Het woord “ransomware” is een samentrekking van het Engelse *ransom*, wat “losgeld” betekent, en *software*. Met andere woorden, ransomware is een specifieke vorm van cybercriminaliteit waar een toestel of server van een slachtoffer wordt geïnfecteerd met een vorm van *malware* die onopgemerkt data aanwezig op het systeem cryptografisch zal vergrendelen. Het gevolg is dat het slachtoffer dan geen toegang meer heeft tot deze versleutelde data. Om terug toegang tot deze data te krijgen wordt het slachtoffer gevraagd een bepaalde som losgeld te betalen aan de cybercrimineel. In ruil zal het slachtoffer de nodige sleutel of de nodige software ontvangen die gebruikt kan worden om de versleutelde data weer te ontgrendelen. Kort samengevat is ransomware dus een type cybercrinaliteit die eruit bestaat data gijzel te nemen in ruil voor losgeld. Naast het versleutelen van data, kan in principe ook data gestolen worden. In dit geval zal van het slachtoffer een som losgeld gevraagd worden om te beletten dat de aanvaller de gestolen data publiekelijk vrijgeeft. Een voorbeeld van een dergelijke ransomware aanval is degene die recentelijk heeft plaatsgevonden bij NVIDIA in februari 2022. NVIDIA is een bedrijf dat voornamelijk bekend is voor het ontwikkelen van videokaarten voor laptops en PC’s. De aanval was vermoedelijk uitgevoerd door de “Lapsus$” hacker groep. De groep had in deze aanval confidentiele gegevens en broncode buit gemaakt bij NVIDIA en gedreigt deze publiek te maken indien er geen losgeld werd betaald. Uiteindelijk zijn de gestolen gegevens effectief online gelekt [Pan22].

De alsmaar stijgende populariteit van ransomware aanvallen kan toegeweden worden aan 2 oorzaken. Ransomware is zowel een lucratieve als een relatief eenvoudig te organiseren activiteit. Uit onderzoek [Jon22] is gebleken dat in 2021 één op de tien organisaties een som losgeld hebben betaald van 1 miljoen U.S. Dollar. Daarnaast was het laagste gemiddelde bedrag dat als losgeld werd geëist rond de 200.000 U.S. Dollar, voornamelijk in de gezondheidszorg en de overheidssector. Dit toont aan dat ransomware aanvallen organiseren een lucratieve business kan zijn. Daarnaast is het organiseren van een ransomware aanval ook relatief eenvoudig. Vandaag de dag bestaat er een actieve *ransomware-as-a-service (RaaS* sector. Wanneer men een ransomware aanval zou willen organiseren kan men simpelweg hiervoor naar een RaaS operator gaan. Een dergelijke operator voert dan een ransomware aanval op aanvraag uit bij het gevraagde doelwit. De operator doet dit tegen betaling of voor een percentage van de opbrengst van de aanval. Een belangrijk voorbeeld van een aanval die op deze manier heeft plaatsgevonden is de ransomware aanval op de “Colonial Pipeline” in de Verenigde Staten in mei 2021 [Ker21]. De nasleep van de aanval was significant. “Colonial”, het bedrijf achter Colonial Pipeline, had vanwege de aanval voor het eerst in 57 jaar zijn operaties tijdelijk stilgelegd. Het gevolg was lange wachtrijen aan tankstations over het hele land, gevolgd door een algemene prijsstijging van brandstof aan de pomp [TM21]. De aanval was georchestreerd door de DarkSide hacker groep onder een vorm van ransomware-as-a-service. In een analyse uitgevoerd door “Elliptic” [Rob21], cryptocurrency onderzoeksbedrijf, is gebleken dat DarkSide in totaal al 90 miljoen U.S. Dollar heeft ontvangen in de vorm van allerlei ransomware betalingen. 15.5

miljoen dollar hiervan is verdeeld onder DarkSide developers, de rest is verdeeld onder andere aanverwante partijen.

In het hedendaagse ransomware klimaat wordt het losgeld geëist in cryptomunten. De reden hiervoor is dat cryptomunten erom bekend staan anoniem, althans toch zeker pseudoanoniem te zijn. Dit is gunstig voor de ontvanger van het losgeld, omdat zo de identiteit van de aanvaller verborgen kan blijven en ook omdat er geen conventionele financiële instanties mee gemoeid moeten gaan, wat uiteraard gewenst is vanuit het perspectief van de aanvaller. De cryptomunt bij uitstek voor ransomware aanvallen doorheen de afgelopen jaren is Bitcoin, gevolgd door Monero voor een veel kleinere, doch groeiende portie ransomware aanvallen [Fin21].

Dit brengt ons tot de focus van deze thesis. Met de groeiende toename aan ransomware aanvallen groeit ook de nood voor analyses van deze aanvallen. Deze analyses kunnen inzichten bieden in de impact en de organisatorische eigenschappen van een ransomware aanval. In de thesis wordt op zoek gegaan naar wat er nodig is om ransomware aanvallen met behulp van een blockchain analyse concreter in kaart te brengen. Er wordt onderzocht welke inzichten er te verkrijgen zijn en wat er niet mogelijk blijkt te zijn. Vervolgens worden de technieken die toegepast worden in Bitcoin gespiegeld aan Monero om te bestuderen hoe haalbaar het proces blijft binnen de context van een privacy coin.

A.2 Ransomware Traceren in het Bitcoin Ecosysteem

Om een ransomware aanval van korterbij te kunnen analyseren en te traceren doorheen het Bitcoin ecosysteem moeten enkele stappen ondernomen worden om de relevante gegevens te verzamelen en te structureren. Deze stappen worden in deze sectie besproken.

A.2.1 Seed Adressen Verzamelen

Het traceren van ransomware aanvallen in het Bitcoin ecosysteem begint steeds bij de *seed adressen*. Een seed adres is een Bitcoin adres dat eigendom is van de ransomware aanvaller waar slachtoffers hun losgeld naartoe kunnen sturen. Er zijn 4 manieren om seed adressen te verzamelen. Men kan op zoek gaan naar gekende slachtoffers van een ransomware aanval en via deze weg meteen een seed adres verzamelen. Anderzijds kan doelbewust een synthetisch slachtoffer worden gemaakt door zelf een ransomware binary te installeren en via deze weg een seed adres te bekomen. Een derde optie is om derde partijen te raadplegen die zich bezighouden met het verzamelen van seed adressen die toebehoren aan ransomware adressen, of datasets uit bestaand onderzoek rond ransomware. Een laatste methode, voorgesteld in deze thesis, is om op zoek te gaan naar seed adressen rechtstreeks op de blockchain door middel van feature matching.

A.2.2 Clustering

Nadat, voor een gegeven ransomware aanval, een verzameling seed adressen is verzameld, kunnen deze gebruikt worden als startpunt voor het traceren en in kaart brengen van de aanval. Echter, de grootte van de verzameling seed adressen die uit de vorige stap resulteren gaat in veel gevallen niet eindig en volledig zijn. Wellicht bestaan er voor dezelfde ransomware aanval nog meer adressen die met voorgaande stappen niet "ontdekt" worden. Om toch nog de initiële verzameling uit te kunnen breiden wordt gebruik gemaakt van de "common-input ownership heuristic", of de "gemeenschappelijke-input eigendom heuristiek" in het Nederlands. Wat deze heuristiek zegt is dat alle inputs van eenzelfde transactie wellicht toebehoren aan dezelfde Bitcoin portemonee, en bij uitbreiding dus ook dezelfde achterliggende entiteit. In onderstaande listing wordt een concreet voorbeeld aangehaald.

```
A (2 BTC) -> X (5 BTC)
B (2 BTC)    Y (1 BTC)
C (2 BTC)
```

Listing A.1: Voorbeeld van een transactie waarop we de "multiple-input ownership heuristic" kunnen toepassen. In dit voorbeeld stellen A, B en C de inputs van transactie voor. X en Y stellen de outputs van de transactie voor. Met behulp van de heuristiek kunnen we, onder voorbehoud, stellen dat A, B en C allen eigendom zijn van dezelfde portemonee, en bij uitbreiding dus ook dezelfde achterliggende entiteit.

Dit principe kan worden toegepast op alle seed adressen die gekend zijn over een gegeven ransomware aanval. Voor elk seed adres kan individueel gekeken worden in welke transacties op de blockchain het adres voorkomt als input. Vervolgens kunnen alle andere inputs uit diezelfde transactie ook worden toegevoegd aan de verzameling seed adressen. Tot slot kunnen dezelfde stappen ook nogmaals worden herhaald op de "nieuwe" adressen. Dit kunnen we blijven doen tot de verzameling seed adressen uiteindelijk niet meer veranderd. Een belangrijke opmerking bij deze aanpak is dat ze gebaseerd is op een heuristiek, en niet een algoritme. Bijgevolg is er altijd een mate van onzekerheid gemoeid met de adressen die met deze clustering techniek gevonden worden.

CoinJoin transacties

Doorgaans genomen zou de multiple-input ownership heuristiek altijd waar moeten zijn zolang Bitcoin transacties worden gebruikt zoals bedoeld bij het ontwerp van Bitcoin. Echter bestaat er een categorie van transacties die niet voldoen aan dit initieel ontwerp, en waarop de heuristiek dan ook niet waar is. Deze categorie zijn de *CoinJoin* transacties.

Een CoinJoin transactie is een bijzonder type transactie dat bedacht is om het traceren van een transactie moeilijker te maken. Bij een "standaard" Bitcoin transactie voorziet één enkele gebruiker alle inputs voor de transactie en kent vervolgens outputs toe aan een andere gebruiker, en soms aan zichzelf (wisselgeld). In dit soort transacties geldt de multiple-input ownership heuristic uiteraard volledig. In een CoinJoin transactie gaan verschillende deelnemers zich gezamenlijk coördineren om een nieuwe transactie te maken waar verschillende deelnemers de inputs voorzien van eenzelfde transactie, en waar ieders individuele ontvanger in de transactie outputs wordt geplaatst. Het gevolg is dat er een soort "groepstransactie" ontstaat waarbij het niet mogelijk is om te achterhalen welke transactie inputs horen bij welke transactie outputs. Mits meerdere gebruikers gezamelijk transactie inputs voorzien geldt hier ook de multiple-input spending heuristiek niet meer. CoinJoin transacties vormen de basis voor "mixer" services. Dit zijn services die bestaan om de traceerbaarheid van transacties te ontnemen. Mixers zijn om deze reden ook in trek binnen het ransomware milieu, omdat dit een extra laag van bescherming biedt voor de aanvallende partij.

Er is reeds onderzoek gevoerd naar het achterhalen van CoinJoin transacties [Gol+18] [Sto+21] [Wu+20] waaruit verschillende heuristieken zijn voortgekomen die telkens zijn toegespitst op zeer specifiek mixer services. Met deze heuristieken kan een mixer transactie worden opgespoord en kan achterhaald worden welke transactie inputs bedoeld zijn voor welke transactie outputs. Het probleem hiermee is dat dat er geen eenduidige aanpak bestaat die in het algemeen van toepassing kan zijn om CoinJoin transacties te achterhalen. Daarnaast is het landschap van mixer services en CoinJoin services ook zeer breed gezaaid en constant in verandering. Ontwikkelaars van deze services updaten hun technieken continu, gepaard met het verdwijnen en verschijnen van oude en nieuwe gelijkaardige services. Het is niet realistisch om een heuristiek of algoritme te ontwikkelen met de gewenste eigenschappen die universeel toepasbaar is op alle mixer services. Dit vorm voor het traceren van transacties momenteel de grootste blokkade.

A.2.3 Graaf Representatie

Eens een verzameling van seed adressen is bekomen, met of zonder hulp van de multiple-input ownership heuristiek, kan deze verzameling adressen voorgesteld worden in een graaf database. In een dergelijke graafvoorstelling worden de adressen voorgesteld als knopen, en transacties tussen adressen als bogen. Het voordeel van het voorstellen van een transactienetwerk als een graaf is dat er later graafqueries kunnen worden gebruikt om handiger dingen te berekenen.

In deze thesis bestaat de inhoud van een ransomware graaf uit de seed adressen die de initiële knopen voorstellen. Vanuit deze knopen worden bogen getrokken voor alle transacties waar de seed adressen als input voorkomen. Vervolgens worden ook voor alle adressen waar deze transacties "aankomen" knopen getekend. In deze voorstelling noemen we de seed adressen de "eerste laag" van de graaf, en de adressen die betalingen ontvangen van adressen uit de eerste laag noemen we de tweede laag van de graaf. In principe zou de graaf vervolgens kunnen worden uitgebreid met een derde laag, vierde laag, ... n-de laag. Echter is hier weinig reden toe. De tweede laag van een ransomware graaf fungeert typisch als een "collector" laag. In deze laag worden de sommen losgeld van alle losse seed adressen in één of meerdere collector adressen verzameld. Een collector adres kan vervolgens deel uitmaken van een exchange service, waar de verzamelde Bitcoins worden omgezet naar fiat geld, of mogelijk een andere cryptomunt. In dit geval stopt de trace in deze collector adressen, aangezien alles wat nog volgt in de derde laag en daarna met weinig zekerheid nog steeds toebehoort aan de ransomware aanval in kwestie. Een andere optie is dat de collector adressen in de tweede laag fungeren als inputs voor mixer services of CoinJoin transacties om de traceerbaarheid te beperken. In dit geval stopt ook de trace reeds in de tweede laag, aangezien er niets met zekerheid kan worden gezegd over adressen of transacties uit de derde laag of daarna. Dit toont ook aan dat het proces om ransomware betalingen te traceren al snel met een stok in de spaken komt te zitten. Wat in deze stap van het proces nodig zou zijn is een procedure die in staat is om mixer services te herkennen en achterhalen, alsook een procedure die in staat is om exchange services te identificeren door het bestuderen van adressen en transacties in kwestie. Tot op heden bestaan geen van deze technieken. Ergens is dit ook niet onlogisch, aangezien dit ook deel van het ontwerp van Bitcoin is geweest.

A.3 TRaP

Om het traceren van ransomware gerelateerde betalingen naar de praktijk te brengen is een tool ontwikkeld, genaamd *TRaP (Tracer of Ransomware Payments).* De motivatie achter de ontwikkeling van TRaP is dat er een tekort is aan doelspecifieke crypto-analyse tools die relatief eenvoudig in gebruik te nemen zijn en gelijktijdig ook open source zijn. Wanneer men als derde partij aan crypto-analyse wil doen op de Bitcoin blockchain kan men in eerste instantie naar een commerciële dienst toegaan. Diensten als Chainalysis[1] bieden tegen betaling heel wat mogelijkheden aan om aan de slag te gaan.

In de open source ruimte is het aanbod redelijk beperkt. Een wel te overwegen kandidaat hier is *GraphSense*[2]. GraphSense is een open source oplossing die de Bitcoin blockchain indexeert in een graaf, gelijkaardig aan wat in de vorige sectie werd voorgesteld. Vervolgens kan deze graafstructuur gebruikt worden als basis om allerhande analyses uit te voeren. Het voordeel hiervan is dat een breed scala aan probleemstellingen beantwoord kan worden met behulp van GraphSense. Het nadeel is dat het installeren en beheren van een GraphSense instantie ontzettend resource intensief is, en bijgevolg niet triviaal inzetbaar.

Als antwoord hierop is TRaP ontwikkeld. TRaP is een open-source oplossing die ontwikkeld is om op een alledaagse laptop of desktop te kunnen runnen. Dit wordt mogelijk gemaakt door de tool specifiek op de analyse van ransomware aanvallen toe te spitsen. Dit ontneemt heel wat data processing die wel nog steeds vereist is in GraphSense, met als nadeel dat TRaP ook niet algemeen toepasbaar is op eender welke probleemstelling buiten de analyse van ransomware aanvallen om.

Om de omgeving waarin TRaP opereert volledig lokaal te houden en onafhankelijk van externe diensten, zijn nog wat andere componenten vereist. In eerste instantie is een “Bitcoin Full Node” vereist. Dit is een volwaardige node die in verbinding staat met het Bitcoin netwerk. De node kan gebruikt worden om informatie op te halen uit de Bitcoin blockchain via de ingebouwde JSON-RPC server. In tweede instantie is een *ElectrumX* server vereist. Dit is een server die informatie van de JSON-RPC opvraagt en indexeert in een lokale database voor gestructureerde en snelle toegang. Vervolgens wordt een *ArangoDB* instantie gebruikt als graaf database om de ransomware graaf in op te slaan. ArangoDB ondersteunt ook de *Arango Query Language (AQL)*, die gebruikt kan worden om graafqueries op de graaf uit te voeren.

A.3.1 Resultaten

TRaP kan verschillende inzichten in een ransomware aanval automatisch berekenen. De resultaten van TRaP worden systematisch vergeleken met bestaand onderzoek uitgevoerd door Paquet et al. [PHD19]. De bekomen resultaten vergeleken met resultaten van Paquet et al. zijn te vinden in tabellen A.1, A.2, A.3 en figuur A.1.

Wat besloten kan worden uit deze resultaten is dat TRaP erin slaagt om dezelfde resultaten te produceren als in bestaand onderzoek door Paquet et al. Wat ook opvalt is dat in de meeste gevallen wel een lichte afwijking naar boven wordt waargenomen. Dit valt te verklaren door het tijdsverschil dat zit tussen de berekening van de bestaande resultaten en degene die berekend zijn met TRaP. De bestaande resultaten zijn bekomen doorheen het jaar 2017. Tussen 2017 en 2022 zijn hebben de adressen in kwestie nog activiteit vertoond die toen nog niet meetbaar was. Voor de meeste ransomware families is deze activiteit verwaarloosbaar, op DMAlockerV3 en NoobCrypt na. Zij vertonen beide nog een zeer actieve fase na

[1]https://www.chainalysis.com/
[2]https://graphsense.info/

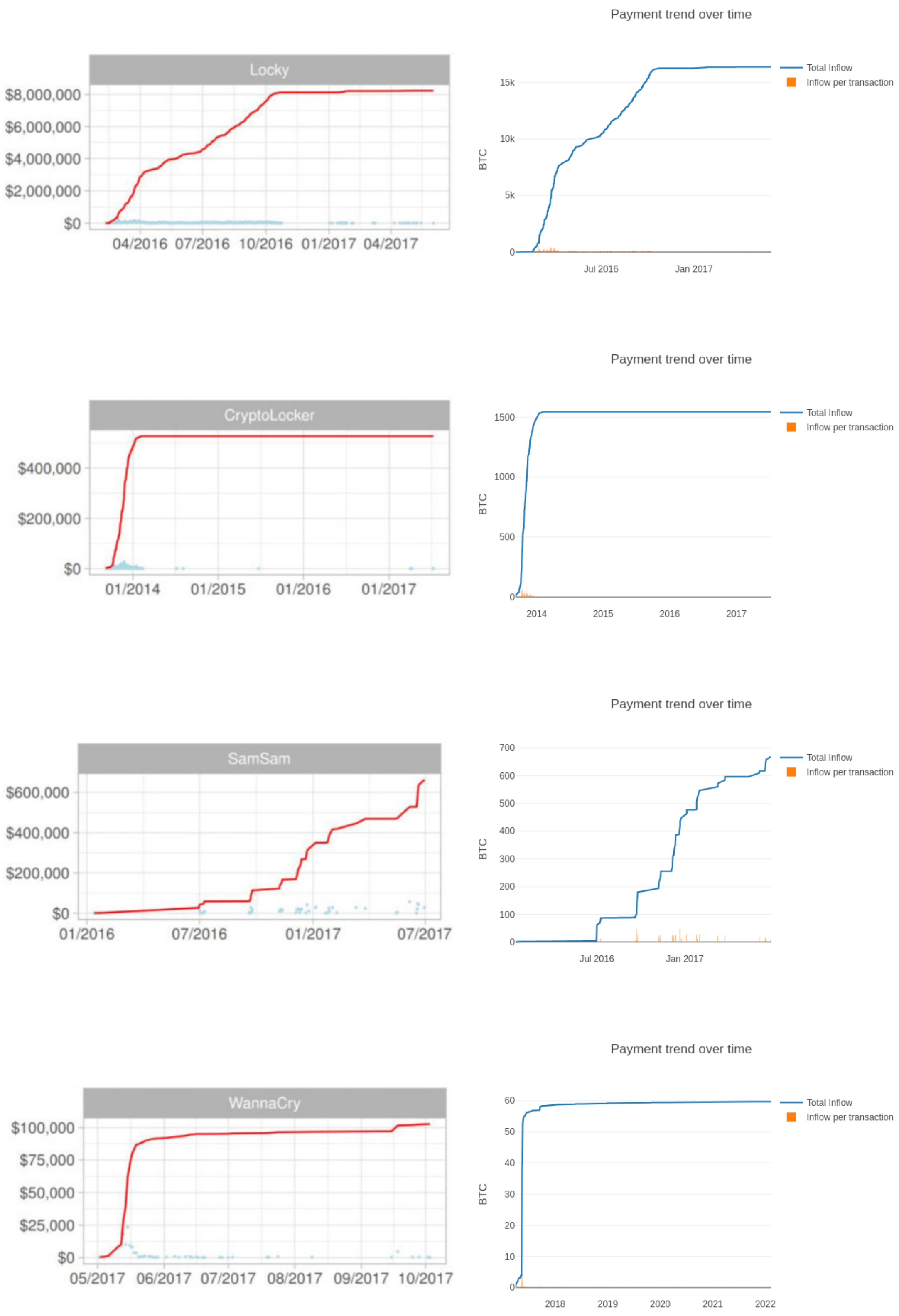


Figure A.1: Voor vier ransomware families de betaaltrends over de tijd. De linkergrafieken stellen telkens de betaaltrends voor zoals beschreven in onderzoek van Paquet et al. De rechterkolom stelt de betaaltrends voor zoals berekend door TRaP. Opmerking: in de grafieken door TRaP wordt het aantal Bitcoin als eenheid gebruikt, terwijl Paquet et al. U.S. Dollar gebruiken. Deze keuze is doelbewust, aangezien de waarde van een Bitcoin in dollar fluctueert over de tijd, zal de bijhorend waarde in dollar uit de grafieken van Paquet et al. nooit helemaal correct zijn maar slechts een benadering, terwijl de waarden uitgedrukt in Bitcoin wel altijd helemaal juist zullen zijn.

Ransomware family	Paquet et al.		TRaP	
	Clusters	Exp. Addr.	Clusters	Exp. Addr.
Locky	1	6827	1	7094
SamSam	11	41	11	72
JigSaw	4	17	4	23
WannaCry	1	6	1	6
DMALockerV3	3	147	3	177
CryptXXX	1	1304	1	1742
NoobCrypt	1	28	1	28

Table A.1: Vergelijking van de resultaten bekomen met behulp van TRaP vergeleken met resultaten bekomen in onderzoek door Paquet et al. De kolom “clusters” geeft het aantal alleenstaande clusters weer, de kolom “Exp. Addr.” geeft weer hoeveel seed adressen er gekend zijn na clustering.

Ransomware Family	Paquet et al.	TRaP
Locky	571	571
SamSam	37	38
JigSaw	17	26
WannaCry	$\leq$ 6	1
DMAlockerV3	71	79
CryptXXX	926	926
NoobCrypt	54	71

Table A.2: Vergelijking van de resultaten bekomen met behulp van TRaP vergeleken met resultaten bekomen in onderzoek door Paquet et al. De numerieke waarden stellen voor elke ransomware familie voor hoeveel collector adressen er ontdekt zijn.

Ransomware Family	Paquet et al.	TRaP
Locky	15 399.01	16 355.16
SamSam	632.01	668,72
JigSaw	$\leq$ 10.75	5.81
WannaCry	55.34	59.66
DMAlockerV3	1 505.78	2 185.30
CryptXXX	3 339.68	3 427.22
NoobCrypt	54.34	1142.44

Table A.3: Vergelijking van de ondergrens van de financiële impact van een ransomware familie berekend met TRaP, vergeleken met resultaten uit Paquet et al. De gegeven waarden zijn telkens uitgerdrukt in Bitcoin.

publicatie van het onderzoek door Paquet et al., wat een sterke opwaartse deviate in de resultaten verklaart.

Naast het produceren van deze resultaten zijn ook nog enkele pogingen gedaan om andere aspecten te implementeren in TRaP, echter zijn deze niet sluitend positief geëvalueerd. Een eerste additioneel aspect dat geïmplementeerd is, is het kunnen opsporen van seed adressen rechtstreeks op de blockchain door gebruik te maken van gekende eigenschappen van een ransomware aanval. Gekende eigenschappen kunnen bijvoorbeeld zijn dat geweten is in welke tijdsperiode de aanval plaatsvond, wat de gevraagde som losgeld is, wat voor type adressen er gebruikt zijn, etc. Deze functionaliteit is geïmplementeerd, maar kon niet voldoende onderbouwd worden om te besluiten dat deze accuraat genoeg was. Daarnaast is ook nog een feature ingebouwd die tracht om voor een gegeven adres te achterhalen of het deel uitmaakt van een exchange service of mixer service. Deze feature bleek uiteindelijk moeilijk te implementeren wegens afhankelijkheid van informatie van derde partijen. De partijen die hierbij zinvol zouden zijn zijn meestal tegen betaling, terwijl de gratis en open-source oplossingen vaak niet up-to-date waren en geen zinvolle informatie konden opleveren over de adressen horende bij de eerder besproken ransomware families.

A.4 Monero

Uit een rapport dat jaarlijks wordt geproduceerd door het *Financial Crimes Enforcement Network (FinCEN)* blijkt dat Bitcoin the cryptomunt bij uitstek is als betaalmiddel voor ransomware aanvallen. Een andere bron vermeldt dat 98% van ransomware aanvallen gebruik maakt van Bitcoin als betaalmiddel. Het FinCEN rapport vermeldt ook dat het in 17 gevallen heeft waargenomen dat betalingen hoofdzakelijk of gedeeltelijk worden geëist in *Monero*. Monero staat garant voor "maar" 37 miljoen U.S. Dollar in ransomware betalingen, vergeleken met 5.1 miljard U.S. Dollar in Bitcoin op dezelfde tijdspanne in 2021. Monero is aanzienlijk minder aanwezig, maar maakt wel een sterke opmars. Om deze reden is het zinvol om het analyseproces van Bitcoin ook trachten uit te breiden naar Monero. Monero wordt hoofdzakelijk verkozen door ransomware aanvallers omdat het een *privacy coin* is, waar Bitcoin een *transparante* cryptomunt is. Het principe van Monero is dat geen enkele gebruiker of transactie te traceren moet zijn, wat volledige anonimiteit garandeert. Er wordt in de thesis om deze reden een kijk genomen naar de maatregelen die Monero neemt bovenop een transparante munt, en welke impact dit heeft op zijn traceerbaarheid.

A.4.1 Eigenschappen

Monero neemt 3 additionele maatregelen om een betere anonimiteit te creëren. In eerste instantie gebruikt Monero *stealth adressen*. Een stealth adres is een adres dat de zender van een transactie afleid van het Monero adres van de ontvanger op het moment dat de zender een transactie aanmaakt. Enkel de zender en ontvangen van de transactie zijn staat om te achterhalen wie het stealth adres toebehoort. Op deze manier blijft de privacy van een transactie-ontvanger gegarandeerd.

In tweede instantie maakt Monero gebruik van *ring signatures*. Een ring signature fungeert als input van een transactie. In een transparante cryptomunt zal een transactie input slechts uit 1 adres bestaan. Hierdoor is meteen duidelijk dat het desbetreffende adres de input van de transactie voorziet. In een ring signature wordt een transactie input vervangen door een willekeurige "ring" van andere adressen op de Monero blockchain die dienen als mixin, gecombineerd met het effectieve input adres. Vervolgens wordt de input ring zodanig ondertekend dat enkel de echte transactie input geconsumeerd wordt, maar dat het voor derde partijen ook niet te achterhalen is welk van de adressen in de ring fungeert als de echte transactie input, en welke als mixin. Deze maatregelen dient om de identiteit van de zender te garanderen.

Tot slot maakt Monero ook gebruik van een uitbreiding op ring signatures, genaamd "Ring Confidential Transactions (RingCT)". Dit is een techniek die wordt gebruikt om de inhoud van een transactie te verbergen voor derde partijen.

A.4.2 Aanvallen

De additionale maatregelen die Monero neem ten opzichte van transparante cryptomunten maken het moeilijk tot onmogelijk om de technieken die ontwikkeld zijn om ransomware aanvallen in kaart te brengen in Bitcoin over te dragen naar Monero. Tot op heden zijn er drie aanvallen bekend op de Monero blockchain die elk trachten afbraak te doen aan de anonimiteit die Monero garandeert.

Zero mixin removal

De eerste van deze aanvallen is de *zero mixin removal* aanval. De aanval steunt op het feit dat in de beginjaren van Monero Ring Signatures nog niet verplicht waren in nieuwe transacties. Als gevolg ontstonden er toen veel transacties die per transactie input slechts één adres hadden. Wanneer dit soort adressen in latere transacties voorkomt als mixin in een ring signature kan al worden uitgesloten dat dit niet de echte transactie input zal zijn, maar slechts een mixin. Dit zelfde principe kan worden uitgebreid naar alle rings en adressen doorheen de Monero blockchain. In onderzoek van Kumar et al. [Kum+17] is gebleken dat de aanval, uitgevoerd in 2016, in staat was om the correcte transactie input van een gegeven ring te bepalen in 87.9% van alle gevallen op de Monero blockchain. Naar aanleiding hiervan is een algehele update doorgevoerd in het Monero ecosysteem die ring signatures verplicht maakte, met een vaste minimale ring grootte. Het gevolg is dat de aanval in 2022 nog slechts een effectiviteit heeft grenzend aan de 0%. Dit maakt de aanval niet meer bruikbaar in een hedendaagse context.

Output Merging Heuristic

Een tweede aanval is de *output merging heuristic.* Deze aanval bouwt op de veronderstelling dat wanneer een nieuwe transactie wordt aangemaakt de probabiliteit zeer laag is dat twee mixins die onderdeel zijn van dezelfde ring signature samen output zijn geweest van eenzelfde oudere transactie. De heuristiek zegt dat wanneer dit toch gebeurt deze twee mixins de echte transactie inputs in de ring zijn. Kumar et al. hebben ook deze aanval in de praktijk gezeten behalen een true positive ratio van 87%. Uiteraard is het ook relatief zeldzaam dat een dergelijke situatie zich voordoet.

Temporal Analysis

Tot slot wordt door Kumar et al. nog een derde aanval voorgesteld, de *Temporal Analysis* aanval. Deze aanval steunt op het principe dat wanneer er random mixins worden gekozen voor een transactie input ring, de oudere mixins wellicht niet de echte inputs gaan zijn, en de jongste mixins wellicht het meeste kans hebben om de echte transactie input te zijn. Concreet toegepast zegt deze heuristiek dat de jongste input in een ring de echte transactie input is, de andere zijn dan mixin. De leeftijd van een mixin wordt gemeten door het verschil te meten in tijd tussen het tijdstip dat hij voorkomt als mixin in een ring, en het tijdstip dat hij is “ontstaan” als transactie output in een oudere transactie. De aanval is relatief eenvoudig, en blijkt verwonderenswaardig ook zeer effectief te zijn. Uit resultaten van Kumar et al. blijkt dat de aanval in 98.1% van de gevallen de juiste transactie input identificeert uit een ring met mixins. Als gevolg hiervan is opnieuw een update uitgevoerd in het Monero ecosysteem die de distributie waarmee mixins werden geselecteerd is aangepast, zodanig dat de temporal analysis aanval ook kort tegen de 0% effectiviteit aanleunt anno 2022.

A.5 Conclusie

Na afloop van deze thesis kunnen we enkele zaken concluderen. De ontwikkeling van TRaP leert ons dat enkele van de meest waardevolle inzichten die we kunnen creëren afkomstig moeten zijn van de collectie seed adressen die verzameld wordt. Om deze reden is het belangrijk een zo groot mogelijke en zo representief mogelijke verzameling seed adressen samen te stellen. De multiple-input spending heuristiek speelt hier een elementaire rol in. Een belangrijk inzicht die we *niet* kunnen realiseren is het achterhalen welke adressen mogelijk bij een exchange service of mixer service horen. Indien we konden identificeren bij welke exchange services een adres hoort kon dit waardevolle informatie zijn in de bestrijding van ransomware aanvallen door de exchange services zelf te betrekken in de identificatie van slachtoffers. Echter is er geen sluitende manier die ons in staat stelt deze exchange service identificatie te realiseren. Daarnaast zijn we ook niet in staat om mixers en CoinJoin transacties te achterhalen. Dit stelt het moeilijke dilemma dat elk betrokken adres en transactie in principe in twijfel moet worden getrokken, wat ervoor zorgt dat we enkel helemaal zeker kunnen zijn over informatie die we vergaren direct uit de seed adressen, en mogelijks ook uit collector adressen.

Vervolgens kunnen we ook stellen dat Monero, zijnde een privacy coin, zijn naam zeker waardig is. De maatregelen die Monero neemt bovenop een transparante cryptomunt als Bitcoin bewijzen effectief te zijn in het beschermen van de identiteiten van zowel transactie zenders als ontvangers. Bij uitbreiding is zelfs de inhoud van elke transactie beschermd tegen pottenkijkers. De weinige aanvallen die worden voorgesteld op de Monero blockchain blijken al snel achterhaald te worden, wat ervoor zorgt dat het trachten te traceren van betalingen doorheen de Monero blockchain een weinig effectieve job blijkt te worden. Dit speelt jammer genoeg in het voordeel van de aanvallers, wiens identiteit in de toekomst almaar beter beschermt zal blijven. Dit leidt tot het besluit dat ransomware aanvallen in het algemeen niet snel een halt zal kunnen worden toegeroepen.

Bibliography

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* (1978).

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms.* Third. Reading, Mass.: Addison-Wesley, 1997. ISBN: 0201896834 9780201896831.

[GJA06] Babu Nath Giri, Nitin Jyoti, and M Avert. “The emergence of ransomware”. In: *AVAR, Auckland* (2006).

[Nak08] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: *Decentralized Business Review* (2008), p. 21260.

[Moi09] Robert Moir. *Defining malware: FAQ.* Apr. 2009. URL: `https://docs.microsoft.com/en-us/previous-versions/tn-archive/dd632948(v=technet.10)?redirectedfrom=MSDN`.

[Bra13] Danny Bradbury. “The problem with Bitcoin”. In: *Computer Fraud Security* 2013.11 (2013), pp. 5–8. ISSN: 1361-3723. DOI: `https://doi.org/10.1016/S1361-3723(13)70101-5`. URL: `https://www.sciencedirect.com/science/article/pii/S1361372313701015`.

[13] *Cryptolocker: How to avoid getting infected and what to do if you are.* Oct. 2013. URL: `https://www.computerworld.com/nl/s/article/9243537/Cryptolocker_How_to_avoid_getting_infected_and_what_to_do_if_you_are_`.

[DW13] Christian Decker and Roger Wattenhofer. “Information propagation in the Bitcoin network”. In: *IEEE P2P 2013 Proceedings.* 2013, pp. 1–10. DOI: `10.1109/P2P.2013.6688704`.

[Nar13] Ryan Naraine. *Cryptolocker infections on the rise; US-Cert Issues Warning.* Nov. 2013. URL: `https://www.securityweek.com/cryptolocker-infections-rise-us-cert-issues-warning`.

[Van13] Nicolas Van Saberhagen. “CryptoNote v 2.0”. In: (2013).

[Del15] University of Delaware. *How is malware distributed?* May 2015. URL: `https://sites.udel.edu/infosecnews/2015/05/18/how-is-malware-distributed/`.

[Sea16] 2016 10:36 pm UTC Sean Gallagher - Feb 17. *”locky” crypto-ransomware rides in on malicious word document macro.* Feb. 2016. URL: `https://arstechnica.com/information-technology/2016/02/locky-crypto-ransomware-rides-in-on-malicious-word-document-macro/`.

[17] In: *Cyber-attack: Europol says it was unprecedented in scale* (May 2017). URL: `https://www.bbc.com/news/world-europe-39907965`.

[Ant17] A.M. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain.* O’Reilly Media, 2017. ISBN: 9781491954362. URL: `https://books.google.be/books?id=MpwnDwAAQBAJ`.

[Kan17] Michael Kan. *Paying the Wannacry Ransom will probably get you nothing. here’s why.* May 2017. URL: `https://www.pcworld.com/article/406793/paying-the-wannacry-ransom-will-probably-get-you-nothing-heres-why.html`.

[Kum+17] Amrit Kumar et al. “A traceability analysis of monero’s blockchain”. In: *European Symposium on Research in Computer Security.* Springer. 2017, pp. 153–173.

[Mös+17] Malte Möser et al. “An empirical analysis of traceability in the monero blockchain”. In: *arXiv preprint arXiv:1704.04299* (2017).

[BPS18] S. Bistarelli, Matteo Parroccini, and Francesco Santini. “Visualizing Bitcoin Flows of Ransomware: WannaCry One Week Later”. In: *ITASEC.* 2018.

[BMS18] Stefano Bistarelli, Ivan Mercanti, and Francesco Santini. “An Analysis of Non-standard Bitcoin Transactions”. In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT).* 2018, pp. 93–96. DOI: `10.1109/CVCBT.2018.00016`.

[FB18] Diogo Fernandes and Jorge Bernardino. “Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB.” In: *Data.* 2018, pp. 373–380.

[Gol+18] Steven Goldfeder et al. "When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies". In: *Proceedings on Privacy Enhancing Technologies* 2018.4 (2018), pp. 179–199.

[HH18] Abraham Hinteregger and Bernhard Haslhofer. "An empirical analysis of monero cross-chain traceability". In: *arXiv preprint arXiv:1812.02808* (2018).

[Hua+18] Danny Yuxing Huang et al. "Tracking Ransomware End-to-end". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 618–631. DOI: 10.1109/SP.2018.00047.

[18] *Mastering Monero: The Future of Private Transactions*. Independently Published, 2018. ISBN: 9781731079961. URL: https://books.google.be/books?id=Vc8YwAEACAAJ.

[Del+19] Sergi Delgado-Segura et al. "Analysis of the Bitcoin UTXO Set". In: *Financial Cryptography and Data Security*. Ed. by Aviv Zohar et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 78–91. ISBN: 978-3-662-58820-8.

[PHD19] Masarah Paquet-Clouston, Bernhard Haslhofer, and Benoit Dupont. "Ransomware payments in the bitcoin ecosystem". In: *Journal of Cybersecurity* 5.1 (2019), tyz003.

[Arg20] Ionut Arghire. *Try2Cry ransomware spreads via USB drives*. July 2020. URL: https://www.securityweek.com/try2cry-ransomware-spreads-usb-drives.

[Llc20] Lifars Llc. *Cryptocurrency mixers and their use in ransomware*. Oct. 2020. URL: https://www.lifars.com/2020/08/cryptocurrency-mixers-and-their-use-in-ransomware/.

[Wu+20] Jiajing Wu et al. "Detecting Mixing Services via Mining Bitcoin Transaction Network with Hybrid Motifs". In: *CoRR* abs/2001.05233 (2020). arXiv: 2001.05233. URL: https://arxiv.org/abs/2001.05233.

[Zag+20] Ehab Zaghloul et al. "Bitcoin and Blockchain: Security and Privacy". In: *IEEE Internet of Things Journal* 7.10 (2020), pp. 10288–10313. DOI: 10.1109/JIOT.2020.3004273.

[Che21] James Chen. *Financial Crimes Enforcement Network (fincen)*. Sept. 2021. URL: https://www.investopedia.com/terms/f/fincen.asp.

[Fin21] FinCEN. *ransomware trends in bank secrecy act data between January 2021 and June 2021*. Oct. 2021. URL: https://www.fincen.gov/sites/default/files/2021-10/Financial%20Trend%20Analysis_Ransomware%20508%20FINAL.pdf.

[Has+21] Bernhard Haslhofer et al. "GraphSense: A General-Purpose Cryptoasset Analytics Platform". In: *CoRR* abs/2102.13613 (2021). arXiv: 2102.13613. URL: https://arxiv.org/abs/2102.13613.

[Ker21] Sean Michael Kerner. *What is ransomware as a service (raas)?* July 2021. URL: https://www.techtarget.com/whatis/definition/ransomware-as-a-service-RaaS.

[Lab21] Malwarebytes Labs. *Explained: Spora ransomware*. July 2021. URL: https://blog.malwarebytes.com/threat-analysis/2017/03/spora-ransomware/.

[Pal21] Danny Palmer. *Ransomware is the biggest cyber threat to business. but most firms still aren't ready for it*. Oct. 2021. URL: https://www.zdnet.com/article/ransomware-is-now-the-most-urgent-cyber-threat-to-business-but-most-firms-arent-ready-for-it/.

[Par21] Radhika Parashar. *Bitcoin's whitepaper turns 13, Netizens wish 'Happy Birthday' to the cryptocurrency*. Nov. 2021. URL: https://gadgets.ndtv.com/cryptocurrency/news/bitcoin-whitepaper-cryptocurrency-satoshi-nakamoto-turns-13-2595475.

[Rob21] Dr. Tom Robinson. *Darkside ransomware has netted over $90 million in Bitcoin*. May 2021. URL: https://www.elliptic.co/blog/darkside-ransomware-has-netted-over-90-million-in-bitcoin.

[Sto+21] Johann Stockinger et al. "Pinpointing and Measuring Wasabi and Samourai CoinJoins in the Bitcoin Ecosystem". In: *CoRR* abs/2109.10229 (2021). arXiv: 2109.10229. URL: https://arxiv.org/abs/2109.10229.

[TM21] William Turton and Kartikay Mehrota. *Hackers Breached Colonial Pipeline Using Compromised Password*. June 2021. URL: https://www.bloomberg.com/news/articles/2021-06-04/hackers-breached-colonial-pipeline-using-compromised-password.

[22a] 2022. URL: https://academy.binance.com/en/articles/coin-mixing-and-coinjoins-explained.

[Bit22] Inc BitPay. *Bitcore*. 2022. URL: https://github.com/bitpay/bitcore.

[22b] *CoinJoin*. 2022. URL: https://en.bitcoin.it/wiki/CoinJoin.

[22c] *Common-input-ownership heuristic*. 2022. URL: https://en.bitcoin.it/wiki/Common-input-ownership_heuristic.

[Fac22] Inc Facebook. *RocksDB*. 2022. URL: https://github.com/facebook/rocksdb.

[Fra22] Jake Frankenfield. *What is a stealth address?* Feb. 2022. URL: `https://www.investopedia.com/terms/s/stealth-address-cryptocurrency.asp`.

[Jon22] David Jones. *Ransomware attacks, payouts soared worldwide in 2021: Report.* Apr. 2022. URL: `https://www.cybersecuritydive.com/news/ransomware-attacks-payouts-2021/622784/`.

[NJ22] Johann Bauer Neil Booth and John Jegutanis. *ElectrumX.* 2022. URL: `https://github.com/kyuupichan/electrumx`.

[OSu22] Fergus O'Sullivan. *How anonymous is bitcoin?* Jan. 2022. URL: `https://www.howtogeek.com/741484/how-anonymous-is-bitcoin/`.

[Pan22] Joe Panetierri. *Lapsus$ cyberattack victim list: Globant, Microsoft, Nvidia, Okta, Samsung, T-Mobile.* Apr. 2022. URL: `https://www.msspalert.com/cybersecurity-breaches-and-attacks/ransomware/alleged-lapsus-cyberattack-victim-list-grows-microsoft-nvidia-okta-samsung-more/`.

[Tou22] Bill Toulas. *Deadbolt ransomware now targets ASUSTOR devices, asks 50 BTC for master key.* Feb. 2022. URL: `https://www.bleepingcomputer.com/news/security/deadbolt-ransomware-now-targets-asustor-devices-asks-50-btc-for-master-key/`.

[bit] bitcoins.net. *Bitcoins.* URL: `https://bitcoins.net/faq/bitcoin-confirmations`.

[Deva] Bitcoin Developers. *Protect your privacy.* URL: `https://bitcoin.org/en/protect-your-privacy`.

[Devb] Bitcoin Developers. *Transactions.* URL: `https://developer.bitcoin.org/devguide/transactions.html`.

[Devc] Monero Developers. *Moneropedia: Ring CT.* URL: `https://www.getmonero.org/resources/moneropedia/ringCT.html`.

[Mar] Marsch. *Ransomware: Paying Cyber Extortion Demands in Cryptocurrency.* URL: `https://www.marsh.com/us/services/cyber-risk/insights/ransomware-paying-cyber-extortion-demands-in-cryptocurrency.html`.