Imperial College London

Imperial College London

Department of Computing

Investigating the Source of Wealth of Bitcoin Addresses

Master's Thesis

Author:
Arthur DRIANT

Supervisor:
William KNOTTENBELT

Submitted in partial fulfillment of the requirements for the MSc degree in Computing of Imperial College London

August 2022

Abstract

Bitcoin is a cryptocurrency whose transactions are openly accessible on the blockchain by anyone. Each bitcoin owner has a collection of addresses that serve as pseudonyms on which they can receive and send bitcoins to other users. Thus, the Bitcoin blockchain is generally assumed to provide a high degree of anonymity, which makes identifying users theoretically impossible. However, techniques and heuristics have been embraced to link addresses to their real identity, which can therefore help us identify transaction issuers. In our work, we are interested in knowing where bitcoins of an address come from and thus understand how an address became so rich.

In order to understand, we design and implement BTC tracker, a web application providing insightful visualisations of transaction flow into and out of bitcoin addresses. We do so by developing heuristics to fine-tune the parsing of the blockchain and introducing two novel metrics: the *Ratio to Origin* and the *Closeness to Service*. The former estimates how much of the original bitcoins persist in an earlier transaction, whereas the latter quantifies a relation between an address and the entities from which its bitcoins came, therefore allowing the best interpretation of the results to give an estimation of the source of wealth of an address and to understand how that money was spent. We apply these processes to a Romance Scam and track 50,000 bitcoins through the blockchain layers.

Acknowledgements

First, I would like to express my gratitude to my supervisor Professor William J. Knottenbelt, for his patience and insightful suggestions that led me to these results, and without whom this project would not have been possible.

Secondly, I am thankful for my second marker, Dr Maria Grazia for her valuable feedback at the beginning of this project.

Lastly, I would like to thank my classmates and friends, for their precious advice, late-night feedback sessions, and unrelenting support. Special thoughts to Baptiste.

Contents

Chapter 1

Introduction

1.1 Motivations

With an average of 90,000 bitcoins traded every day representing more than \$4,000,000,000[1], Bitcoin (BTC) is currently one of the most used cryptocurrencies. Indeed, since its introduction in 2008 by Satoshi Nakamoto [1], public interest in Bitcoin has only increased and is now a widespread payment method.
However, what makes Bitcoin so different from any other traditional currency is, among other aspects, its pseudonymity. Indeed, although every transaction is publicly accessible by anyone, it is hard to know who issued what transaction, since they are not directly attached to any real-world identity [1]. Moreover, each user can -and should- create as many "pseudonyms" (also called addresses) as they want so as to protect their privacy [2], which makes the task of identifying them even harder.
Thus, it is not possible to identify the issuer and the recipient of a transaction by only looking at that exact transaction. Nevertheless, by broadening our perspective and examining neighbouring transactions, we can in fact gather a great deal of information that can be used to identify said issuers.

In 2018, 460 million addresses had already been created. Among these, only 172 million are economically relevant (37%), meaning they currently, or used to, own BTC. A further 142 million had already been identified as belonging to services according to Chainalysis[2]. But some addresses are still yet to be identified including many of the richest ones as can be observed in Figure 1.1.

Knowing where bitcoins come from is also a matter of security. Indeed, although Bitcoin was originally created as a fungible token (meaning every bitcoin has the same value), it is no longer believed to be the case since every bitcoin has a different transaction history that can greatly impact its value. Indeed, a freshly mined bitcoin is more valuable than bitcoins that have been involved in several fraudulent transactions on the black market.
Nowadays, exchange platforms are even starting to block certain transactions because they don't want to be affiliated with scammers or money launderers [4].

[1] https://www.blockchain.com/charts/estimated-transaction-volume-usd?timespan=all
[2] https://blog.chainalysis.com/reports/bitcoin-addresses/

	Address	Balance △1w/△1m	% of coins
1	34xp4vRoCGJym3xR7yCVPFHoCNxv4Twseo wallet: Binance-coldwallet	252,597 BTC ($7,704,665,088) +0.00238 BTC / +0.00241 BTC	1.33%
2	bc1qgdjqv0av3q56jvd82tkdjpy7gdp9ut8tlqmgrpmv24sq90ecnvqqjwvw97 wallet: Bitfinex-coldwallet	168,010 BTC ($5,124,603,491)	0.8823%
3	1P5ZEDWTKTFGxQjZphgWPQUpe554WKDfHQ	127,368 BTC ($3,884,936,557) +1251 BTC / +3115 BTC	0.6689%
4	3LYJfcfHPXYJreMsASk2jkn69LWEYKzexb wallet: Binance-BTCB-Reserve	116,601 BTC ($3,556,542,126)	0.6123%
5	bc1qazcm763858nkj2dj986etajv6wquslv8uxwczt	94,643 BTC ($2,886,789,024)	0.4970%
6	37XuVSEpWW4trkfmvWzegTHQt7BdktSKUs wallet: 77604498	94,505 BTC ($2,882,581,075)	0.4963%
7	3M219KR5vEneNb47ewrPfWyb5jQ2DjxRP6 wallet: Binance-coldwallet	90,913 BTC ($2,773,008,255) -19046 BTC / +22183 BTC	0.4774%
8	1FeexV6bAHb8ybZjqQMjJrcCrHGW9sb6uF	79,957 BTC ($2,438,837,410)	0.4199%
9	bc1qa5wkgaew2dkv56kfvj49j0av5nml45x9ek9hz6	69,370 BTC ($2,115,913,811)	0.3643%
10	3JZq4atUahhuA9rLhXLMhhTo133J9rF97j wallet: Bitfinex-coldwallet	63,760 BTC ($1,944,807,131) +18828 BTC / +32017 BTC	0.3348%

Figure 1.1: Top 10 richest addresses and their known owner according to WalletExplorer [3]

Therefore, we analyse what kind of transactions the received bitcoins went through and how they were spent to find out what kind of entity is likely to be controlling them, and thus understand how an address became so rich.

1.2 Objectives

In order to achieve our goal, we build a tool providing insightful visualisations of transaction flow into and out of these addresses. Our objective is to go back through the transaction layers of the blockchain and track bitcoins received on an address until we find relevant information that could help us identify the source.
Likewise, we analyse how received bitcoins were spent by going forward in the transaction layers. This may help us find out more about the source of wealth.
We take advantage of known address classifications, apply address clustering techniques such as transaction history summarisation, and develop our own methods to detect any relevant patterns in addresses or transactions.

It may also be relevant to scrape websites or forums such as Twitter or Reddit to look for any information that might give us hints on the user identity hidden behind the address. Collecting addresses suspected of fraud or scam on BitcoinAbuse[3] may also help us find the origin of the transactions. These types of data are said to be "off-chain" information.

Thus, using these clusters and data found on the web, all that is left to do to determine where BTCs are from is backtrace transactions until we find one potentially already identified address.

We decided to pay special attention to scams and fraudulent addresses. That

[3]https://www.bitcoinabuse.com/

is why our main focus is to determine whether or not a user has received fraudulent transactions or if they are part of a scam by checking if their bitcoins have been involved in transactions linked to the Black Market or known Ponzi schemes, for instance.

1.3 Outcomes

Our project consists of 5 parts:

1. **Block-chain Parsing:** Heuristics created to fine-tune the parsing of the blockchain. We introduce a new metric called **R**atio **T**o **O**rigin (RTO) developed to optimise the parsing time while keeping as much information as possible. We compare them to existing parsing methods and we conclude on their weaknesses and strengths in Chapter 4.
2. **Closeness to service:** Introduction of a new metric called **C**loseness **T**o **S**ervice (CTS), helping determine the proximity between bitcoins of a given address and platforms offering services such as facilitating bitcoin transactions or mining pools. Its effectiveness is discussed and its limits are mentioned in Chapter 5.
3. **Web Scraping:** Details of the processes used to retrieve any potentially useful information that can be found online in order to either identify the given address, or to give us hints on what kind of service is controlling it. Discussion on its relevance in Chapter 6.
4. **BTC Tracker:** Introduction of our web application gathering all the technologies explained previously. We describe how it works and apply it to examples to demonstrate its effectiveness in Chapter 7.
5. **Implementation:** Technical description of our implementation, listing of the technology used, languages chosen and choices made. Discussion about their strengths and weaknesses in Chapter 8.

Chapter 2

The Bitcoin Blockchain

Before diving into the subject, it is essential to understand what a Bitcoin is and know the basic structure and features of the Bitcoin blockchain.

2.1 Key notions

Bitcoin is an electronic currency system introduced by Satoshi Nakamoto in the white paper [1] in 2008, allowing payments between two parties without requiring mutual trust. It is based on a peer-to-peer network instead of a centralised entity controlling the input and output flows. To do so, it relies on digital signatures to prove ownership and a public history of transactions, called blockchain, that prevents double spending among other essential features.
The Bitcoin blockchain is a distributed ledger in the form of a chain of chronologically ordered blocks shared by every user of the blockchain [1]. It is entirely public and is made so that nobody can alter it in any way without anybody else noticing, and the majority of miners must agree for a change to be made, thanks to authentication methods and block hashes. In each block are stored transactions.

Bitcoins are held in wallets that are used to send and/or receive bitcoins. Transactions are agreed upon using a proof-of-work system and are made from one wallet to another, allowing the user to stay pseudonymous, as nothing can link a user to its wallet address(es) and no relevant real-world information is revealed during a transaction.

2.2 Transactions and wallets

In order to make transactions, every user requires a bitcoin wallet. A wallet can be owned by either an individual or an entity and can be seen as a key-chain of public keys that the user will communicate to the person with whom they wish to transact. These public keys are derived from private keys that are only known to the user. They ensure the user is the only person that can dispose of their money. However, there is no restriction on whom a user can send money to, as long as they know the address of the recipient.

Several methods exist to make a wallet, but they can be split into two main categories [5]:

- **Nondeterministic (Random) Wallets:** Several private keys are generated independently from one another at the creation of the wallet, and new ones are added when needed. Each of these is used once and only once to generate a public key, which means there are as many private keys as there are public ones. This can quickly become cumbersome to manage as every private key needs to be remembered in order to keep access to the funds it is holding, especially if the user is avoiding re-using its addresses for privacy purposes [2].
- **Deterministic (Seeded) Wallets:** Private keys are all derived from a common seed, allowing the user to easily export or import its wallet, allowing a wallet migration between different wallet services, for instance. Nowadays, the seed is usually represented as an English word sequence that encodes a random number (the seed itself). This method is called a mnemonic code and is usually composed of 12 to 24 words (2).

Therefore, a transaction T can be defined by:

- n inputs $(A_i, \alpha_i, pT_i)_{i\in[1,n]}$, $(n \geq 1)$ where:
  - A_i is an address sending the bitcoins
  - α_i is the amount of bitcoins sent by that address
  - pT_i is the transaction from which these bitcoins come from, i.e. the transaction they were output of. Here, pT stands for "previous transaction". We will often use this term to designate that transaction.
- m outputs $(B_j, \beta_j, nT_j)_{j\in[1,m]}$, $(m \geq 1)$ where:
  - B_j is an address receiving bitcoins
  - β_j is the amount of bitcoins received by that address
  - nT_j is the transaction in which these bitcoins were spent, i.e. the transaction they are an input of. Here, nT stands for "next transaction". We will often use this term to designate that transaction.
    In the case that these bitcoins have not yet been spent, we simply consider that nT_j is set to $None$.

This notation is summed up in the figure 2.1.

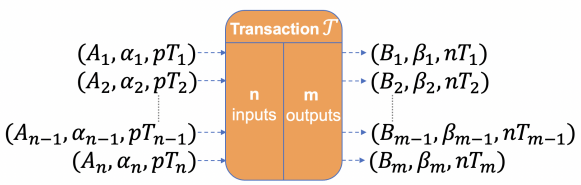


Figure 2.1: Illustration of a transaction using the notation previously introduced.

The Bitcoin blockchain follows the Unspent Transaction Output (UTXO) model, meaning all bitcoins from the sender's address(es) must be sent in the transaction in which they are involved. Thus, the sender must redirect their remaining bitcoins to their wallet. This principle is illustrated in Figure 2.2.

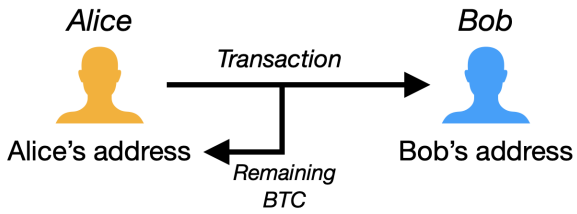


Figure 2.2: Example of a transaction where Alice sends bitcoins to Bob.

More concretely, it means that the sum of the input value must equal the sum of the output values, minus a transaction fee that occurs to incentivise miners to validate the transaction:
Let $\alpha_{i\in[1,n]}$, $\beta_{j\in[1,m]}$ be the values of the n inputs and m outputs respectively, as previously defined. Let ϵ be the transaction fee.
We have then:

$$\sum_{k=1}^{n} \alpha_k + \epsilon = \sum_{i=1}^{m} \beta_i \tag{2.1}$$

2.3 Generating Bitcoins

New bitcoins are created through a decentralised process called "mining". It is a process during which miners validate new transactions and record them on the blockchain by creating a new block of transactions. That new block points toward the hash of the previous one on the blockchain, hence the concept of chain depicted in Figure 2.3.
Transactions that are added to the blockchain are then considered "valid", meaning users involved in these newly added transactions can now use the bitcoins they received.

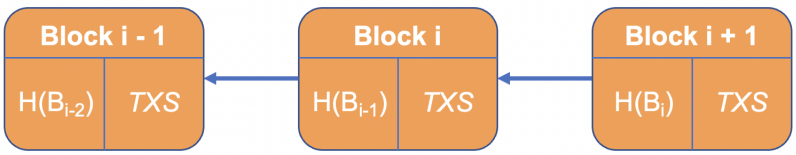


Figure 2.3: Illustration of the blockchain principle. $H(B_k)$ represents the hash of the k^{th} block and TXS is the list of transactions in that block.

In order to add a block to the blockchain, miners compete to solve a resource-intensive mathematical problem based on a cryptographic hash algorithm. Once the solution -called the proof of work- is found, it is included in the new block and acts as proof that the miner used a lot of computational power in order to successfully solve that problem.

This proof takes the form of a transaction called "coin-base" and contains the newly generated bitcoins awarded to the successful miner. It is appended to the list of validated transactions.
In addition to the coin-base bitcoins, the miner also receives transaction fees from all the transactions he mined.

The mining mechanism is crucial for the blockchain as it ensures that the Bitcoin system is safe from fraudulent transactions or transactions spending the same amount of bitcoins more than once (Double spending transactions).
Therefore, miners provide safety to the network in exchange for the opportunity to be rewarded with bitcoins.

2.4 Transaction graph

As explained in 2.3, bitcoins can only be created through coin-base transactions. Therefore, by parsing the blockchain, it is possible to go back to the coin-base transaction of every bitcoin in circulation today.
Moreover, as the Bitcoin blockchain follows the UTXO model, finding this special transaction also implies that we are able to list all the transactions a bitcoin has ever been involved in. That transaction history is what characterises a bitcoin and makes it unique, thus questioning the notion of fungibility.
This can be represented in a graph called the transaction graph. Following a bitcoin transfer is made easy since this is a directed temporal graph connecting transactions by their inputs and outputs [6]. This graph is acyclic thanks to the UTXO model. Therefore, the output of a transaction can only be used once by another transaction. Thus, if we go through this graph, we know we will never loop and the path we decide to take will end at some point.
A simple example of a transaction graph starting from a wallet address can be found below in figure 2.4.

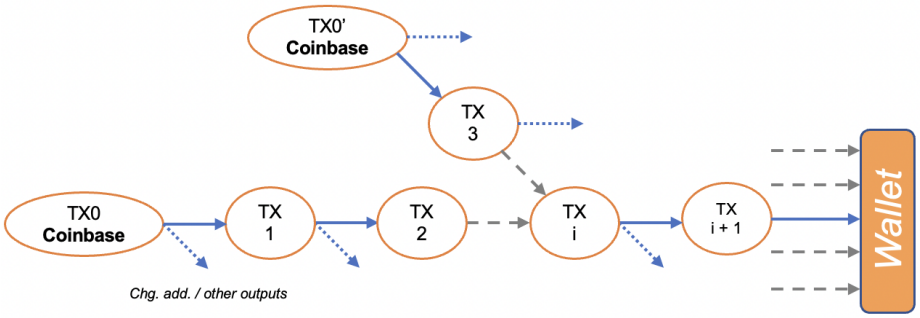


Figure 2.4: Example of a transaction graph of bitcoins stored in a wallet. Blue dotted arrows represent other transactions that were not displayed for the sake of the example.

Chapter 3

Literature Review

3.1 Heuristics to cluster addresses

There exists two well-known heuristics used to cluster BTC addresses [7]. The first one is based on multiple input addresses in a transaction (also called Common Spend (CS)), and the second one is based on one-time change addresses.

3.1.1 Common Spend Transaction heuristic

In the case where a bitcoin owner wants to buy a product listed for 10 BTC and has two different addresses with 8 BTC and 2 BTC respectively, they can use their 2 addresses as inputs to acquire the product. Thus, in most cases, we can assume that multiple addresses used as inputs in the same transactions belong to the same individual, as is depicted in Figure 3.1. It is a rather safe assumption since, as explained in 2.2, in order to use the funds linked to an address and validate the transaction, the user must sign it and therefore must know both private keys.
This is a widely known heuristic and it has been cited in a lot of clustering algorithm papers such as in [7], [6] or [8].

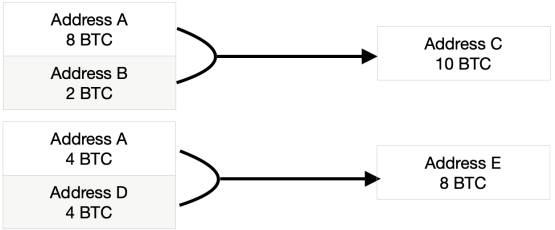


Figure 3.1: Illustration of the Common Spend heuristic. From these 2 transactions, we can infer that addresses A, B and D belong to the same user.

Using the notations previously introduced, this heuristic can be defined as such: Let E be an entity owning an address $\mathscr{A}$, T a transaction, $(A_i, \alpha_i, pT_i)_{i\in[1,n]}$ its inputs and $(B_j, \beta_j, nT_j)_{j\in[1,m]}$ its outputs $(n, m \geq 1)$.
We have the following:

$$\forall i \in \{1, n\}, A_i = \mathscr{A} \implies \forall j \in \{1, n\}, A_j \in E \tag{3.1}$$

However, services such as CoinJoin provide a solution to counter that heuristic by regrouping multiple transactions of the same amount into a single transaction [9]. Therefore, as depicted in Figure 3.2, if A wants to transfer X bitcoins to B and C wants to transfer the same amount to D, CoinJoin will regroup these two transactions with two inputs (A and C) and two outputs (B and D). Thus, it is not possible to know if A paid B or D, and the same applies to C. [10], [11]
These transactions would likely produce false positives with our heuristic.

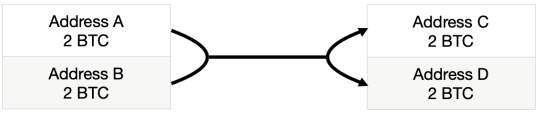


Figure 3.2: Example of a CoinJoin transaction with 2 users sending 2 BTC.

Mixing services such as Bitcoin Fog also blur the transaction history by making it almost impossible to link the input address to the output address(es) [12].
However, we can restrict our heuristic to only consider transactions with only one output, which will eliminate shared transactions issued by such services or users to conceal transaction history [13].

3.1.2 One-time change address heuristic

On the other hand, if a user wants to buy a product listed for 10 BTC and has an address with 15 BTC, he will not use all of his funds to make the transaction. But as the Bitcoin blockchain follows UTXO model, the remains -here 5 BTC- must be redirected to the user's wallet. The user can either use the same address to receive the change (although it is not recommended [2]) or use a new one. This is on what the second heuristic is based.
It is important to note that this heuristic only works if the user resorts to one-time change addresses. These addresses are automatically created by the Bitcoin Client to receive the remains of a transaction and are only used once by the user when the change is spent again (hence the term "one-time") [7].
The hardest part of that procedure is to identify which one in the transaction is the change address. Several definitions of one-time change have been established to limit the rate of false positives [12]. Zhang, Wang and Luo [6] introduce a third heuristic based on address reuse to refine the one-time address change protocol.
However, the one-time change address heuristic is not as used as the first one presented because the former is not as reliable as the latter. In [7], researchers have identified 3.5 million change addresses with a false positive rate estimated at 0.17%, which still represents approximately 6,000 wrongly identified addresses.
Researchers have also shown that clusters can be identified thanks to their transaction behaviours that are characteristics of some hosted services or even of hosted wallet software [8].

3.1.3 Identification of clusters

After all of these different clustering methods have been used to regroup addresses together, the next step is to identify each cluster in order to find to whom it belongs. To do so, we only need to find the identity behind one of the addresses contained in the cluster. Once we find it, we can then categorise the cluster.
These clustering methods include:

Web parsing: Addresses can be found by automatically or manually parsing bitcoin-related forums such as BitcoinTalk [14] or the Bitcoin section of Reddit, where users willingly display their address to receive donations, or to just make a payment. Services such as Binance, for instance, give their customer an address to deposit the money.
Automatic parsing can be achieved through web crawlers, a technology used by D. Ermilov et al. in their paper to collect address tags [13].
But they can also be revealed through data leaks and a discovered identity can therefore be linked with an address cluster. In 2020, the personal information of more than 270,000 Ledgers users was revealed because of a security breach [15]. These services causing leaks store user information off-chain in their database and only broadcast their transactions when it is necessary to reduce the transaction fees.

Transaction History Summarisation This method consists of applying Machine Learning algorithms to transaction information from a wallet in order to recognize patterns and classify addresses [8]. To do so, Kentaroh Toyoda et al.[16] focused on 8 features that can be found in Figure 3.3. Yu-Jing Lin et al [17] later increased the number of features taken into account and added extra statistics about each transaction, but also introduced the notion of transaction moments, a set of metrics corresponding to the time distribution of the transactions.

Feature	Definition
f_{TX}	The frequency of transactions which is defined as the number of all transactions per day.
r_{received}	The ratio of received transactions to all transactions.
r_{coinbase}	The ratio of coinbase transactions to all transactions.
$f_{\mathrm{spent}}(10^i)$	The frequency of digit i in USD appeared in spent transactions, where $i \in (10^{-3}, 10^{-2}, \cdots, 10^{6})$
$f_{\mathrm{received}}(10^i)$	The frequency of digit i in USD appeared in received transactions, where $i \in (10^{-3}, 10^{-2}, \cdots, 10^{6})$
r_{payback}	Payback ratio defined as the ratio of Bitcoin addresses that appear in both inputs and outputs.
$\bar{N}_{\mathrm{inputs}}$	The mean value of the number of inputs in the spent transactions
$\bar{N}_{\mathrm{outputs}}$	The mean value of the number of outputs in the spent transactions

Figure 3.3: Table of extracted features from [16]

Overall, combining multiple sources of information such as forums, websites like Twitter or Reddit, and data leaks allowed the identification of pseudonymous users, and this is what has been used to categorise all the clusters found so far.

3.2 Blockchain Forensics tools

In order to gather information about transactions to or from a wallet address, we need to access the Bitcoin blockchain. As explained in the introduction, every transaction is public and everyone can download the entire blockchain. Downloading it allows to not be constrained by any website limit and gives instant access to block data as well as better reliability, provided enough storage and processing power is available, as the blockchain is currently a little over 400GB[1]. But nowadays, websites allowing to browse and analyse the blockchain exist and tools have been created to ease the reading of the latter.

In this section, we present already existing blockchain forensics tools. We take a closer look at those on which we base our application.

3.2.1 Wallet Explorer

Website

WalletExplorer is a free-to-use website that allows users to easily browse the Bitcoin blockchain. Following a bitcoin transfer is made easy thanks to the possibility of going through the transaction graph. As described in section 2.4, this is a directed temporal graph connecting transactions by their inputs and outputs. This graph is acyclic since the output of a transaction can only be used once by another transaction. Therefore, if we go through this graph, we know we will never loop and the path we decide to take will end at some point.

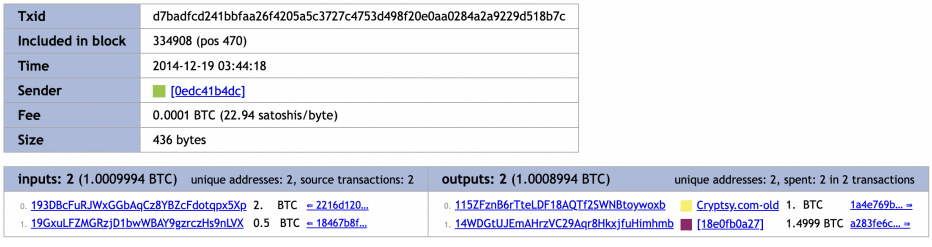


Figure 3.4: Example of transaction displayed by WalletExplorer

On the example of a transaction depicted above can be read several useful information such as:

- **The transaction ID:** Unique 32 bytes (64 characters) hexadecimal number identifying transactions.
- **Its timestamp:** Time when the transaction was broadcast on the network. Not to be mistaken with with the time the block was added to the blockchain.
- **Number of unique addresses:** Since WalletExplorer is based on the transaction graph, a transaction can have the same address in input several times, as bitcoins

[1]https://ycharts.com/indicators/bitcoin_blockchain_size

owned by an address can all come from different transactions (i.e. each correspond to a different UTXO). Therefore, having the number of unique addresses allows to instantly identify if this is a transaction to aggregate bitcoins together.

- **Whether addresses in input have been clustered and/or identified:** WalletExplorer uses multi-input transaction heuristic explained in section 3.1.1 in order to cluster addresses together[2].

When a cluster is found, every address inside is assigned a unique ID that will characterise the wallet it belongs to. Additionally, using different off-chain sources of information not detailed on the website, it also identifies these wallets and tags them with the name of the wallet holder. In that example, we see that "Cryptsy.com-old", an online casino, owns the address "115ZFznB. . . "[3]

Contrary to other blockchain browsing websites, the transaction ID of the inputs can be directly read from the transaction page itself, which makes it easier to browse manually, but also reduces considerably the number of requests to make.

API

WalletExplorer additionally has an semi-public free API that provides the same information than the normal website, but allows a faster parsing and requests of smaller size.


```
{
    "found":true,
    "txid":"d7badfcd241bbfaa26f4205a5c3727c4753d498f20e0aa0284a2a9229d518b7c",
    "is_coinbase":false,
    "wallet_id":"0edc41b4dc93c735",
    "block_height":334908,
    "block_pos":470,
    "time":1418960658,
    "size":436,
    "in":[
        {
            "address":"193DBcFuRJWxGGbAqCz8YBZcFdotqpx5Xp",
            "amount":2,
            "is_standard":true,
            "next_tx":"2216d120a4c28fa5cdd6cbf3a69d2aee06ab75f3e2b334d554f3cbb810846d62"
        },
        { ...
        }
    ],
    "out":[
        {
            "address":"115ZFznB6rTteLDF18AQTf2SWNBtoywoxb",
            "wallet_id":"00000755defaf057",
            "amount":1,"is_standard":true,
            "next_tx":"1a4e769bb6989508019e89e7638f1a64331c788df612b328c1fbdb08f8188e4c",
            "label":"Cryptsy.com-old"
        },
        { ...
    ],
"updated_to_block":749585}
```


Figure 3.5: Response to a request to WalletExplorer's API on the same transaction shown in Figure 3.4. *Result has been shortened for the sake of the example.*

[2]https://www.walletexplorer.com/info
[3]115ZFznB6rTteLDF18AQTf2SWNBtoywoxb

As can be inferred from the response to a request made to WalletExplorer's API depicted in Figure 3.5, the return request states whether this transaction is a standard one, but also provides the type of the input. Therefore, it allows users to directly identify coinbase transactions, for instance. This API however has a limit of 2 requests per second, which needs to be considered regarding the scalability.

However, the wallet name database has not been updated since 2016 and the bitcoin blockchain is only parsed once every 2 days, so this is not the optimal website to use to access the latest transactions. But, as clustering heuristics are still run frequently, this tool still provides insightful information about transactions, as long as the latest transaction data is not required as the blockchain is updated.

Overall, WalletExplorer is a quick and easy low-level solution to browse the blockchain via the transaction graph. We decided to base our application on it to retrieve transaction information.

3.2.2 BlockSci

BlockSci is a fast, secured and open-source software platform that allows multi-blockchain analysis [18]. It consists of a high-performance importer that directly reads the Bitcoin blockchain raw data on disk, an optimised data parser to speed up data browsing thanks to bloom filters, address cache and spent UTXO removal for example, and a user interface allowing extensive analysis tasks of the blockchain. It aims to take advantage of the speed of C++ while providing expressiveness and flexibility of Python via its custom Fluent interface.

BlockSci does not provide address tags. However, if the user provides tags for a subset of addresses, individual clusters can return tags associated with them. Its architecture is described in Figure 3.6.

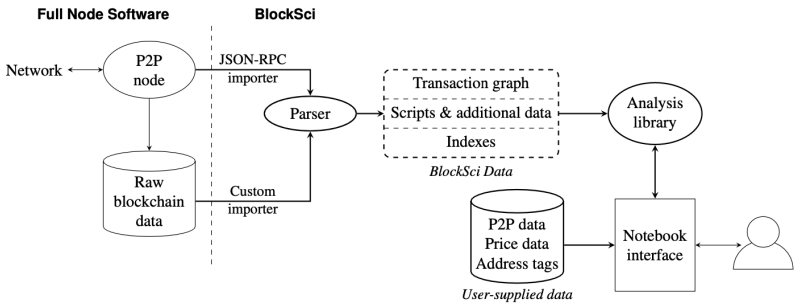


Figure 3.6: Overview of BlockSci's architecture [18]

In summary, BlockSci not only parses the blockchain, but also processes its content to build the transaction graph and provides an optimised user interface to perform extensive analyses.

3.2.3 GraphSense

Analytics platform

GraphSense is an analytics platform used to run customised analytics tasks on data gathered from different blockchains and other off-chain relevant sources. GraphSense is completely open-source and performs similar analytics tasks to BlockSci, on which GraphSense's UTXO model ledgers are based, among other things [19].
The main difference between these two tools is that the former builds address and entity graphs whereas the latter only provides transaction graphs, like WalletExplorer. Indeed, although transaction graphs already unveil a lot of information on addresses [20], [21], we can analyse them further to build address and entity graphs:

- **Address graphs** are bi-directed cyclic graphs in which a node represents an address and an edge represents the set of transactions between two addresses. In other words, it groups transactions according to their pairs of inputs/outputs.
- **Entity graphs** are based on address graphs but go further by clustering addresses that are likely controlled by the same entity (that entity can be a human or an exchange platform, for instance). This process is illustrated in Figure 3.7.

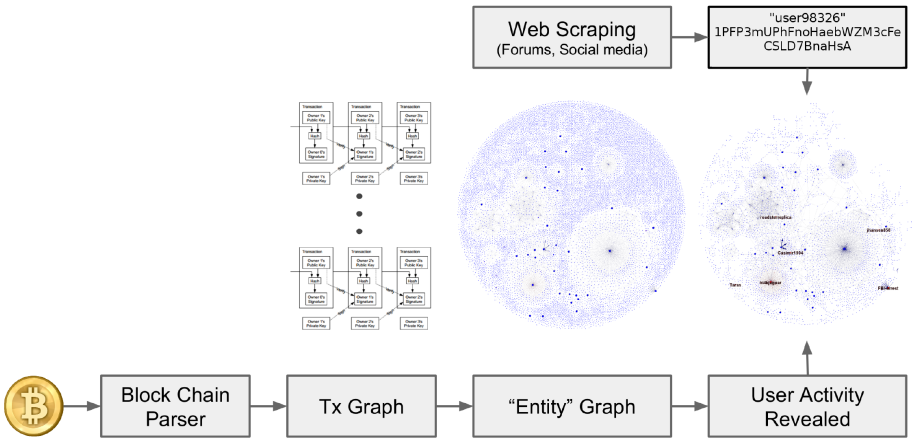


Figure 3.7: Example of the creation of the entity graph issued from [20]

Data can then be accessed via either an API or a visual web interface representing the calculated graphs. Browsing the latter is made easy thanks to a highly interactive interface as illustrated in Figure 3.8 with the same transaction displayed as in the previous examples. Using the entity graph, one can therefore quickly identify the identities of users who sent bitcoins to this address.
A semi-public demo version of that software is available online and has a request limit of 1,000 per hour.

GraphSense provides data on major blockchains such as Bitcoin, Bitcoin Cash, LiteCoin or Ethereum, but for the sake of our project, we only focus on Bitcoin data. They also introduce the notion of TagPacks [19] which are information of any kind

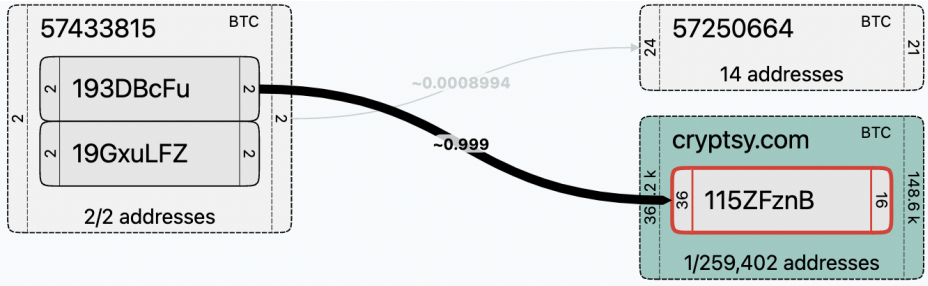


Figure 3.8: Illustration of a transaction on GraphSense. Here, the entity graph is displayed.

about an address, a transaction or a cluster collected and maintained by the community.

TagPacks

TagPacks are a structure for collecting and packaging address attribution tags and stored in YAML files. This is a collaborative database published on GitHub [22] allowing users to add their own tags as long as it has been proven. TagPacks use INTERPOL Darkweb and Cryptoassets Entity Taxonomy[4] to tag the category of the entity that is likely behind the address (e.g Hosted Wallet, Exchange) but also the abuse that it has been reported for (e.g Scam, Ransomware).

```yaml
title: Ransomwhere tag pack
creator: GraphSense Core Team
description: Cryptocurrency addresses associated with ransom
lastmod: 2022-02-15
source: https://ransomwhe.re/
abuse: ransomware
confidence: forensic
tags:
- address: '17TMc2UkVRSga2yYvuxSD9Q1XyB2EPRjTF'
  currency: BTC
  label: Netwalker (Mailto)
```

Figure 3.9: Example of a TagPack

Figure 3.9 depicts an example of a TagPack. We can see this TagPack was created by GraphSense Core Team, and that it contains addresses associated with ransom from Ransomwhere[5]. The first address was received by email. It is attributed to Netwalk, a ransomware that encrypts and exfiltrates all of the data it beaches to then demand a ransom in exchange for the stolen data. Overall, this TagPack contains no less than 7,500 reported addresses.

[4]https://interpol-innovation-centre.github.io/DW-VA-Taxonomy/
[5]https://ransomwhe.re/

In summary, GraphSense gathers data from several sources listed above and processes it to create address and entity graphs that are then displayed via either an application programming or a visual interface.

3.2.4 Other relevant blockchain forensics tools

We presented 3 main tools currently allowing users to browse and analyse blockchain transactions, but many other advanced tools exist. These include, but are not limited to:

- **Chainalysis** [23], a more recent blockchain data platform providing key information about transactions and detection of wallets or wallet identities. This tool is, however, not free to use.
- **BitInfoCharts** [24], a blockchain parser like WalletExplorer, based on the transaction graph. No information on the clustering methods used was found, but the Common Spend heuristic explained in Section 3.1.1 is likely to have been implemented alongside off-chain data gathering. Although no API is available, it is still possible to parse the page directly. However, the transaction IDs from which bitcoins in input come are not available directly from the request.
- **BitIodine** [25], automatically parses the blockchain, clusters and tags addresses, and visualises elaborated information by building of graphs. It is also composed of a web crawler to find insightful off-chain information as explained in 3.1.3.
- **Blockchain Explorer** [26], similar to BitInfoCharts and WalletExplorer, is a multi-blockchain explorer. Thus, it is also based on the transaction graph. An API is available as well[6] and seems to be limited to a request every 10 seconds. However, similarly to BitInfoCharts, additional requests must be made in order to get the previous transaction IDs from which input bitcoins are.

3.3 Conclusion

From the simple chain parser to the advanced analysis tool, several solutions already exist and give out a range of varied and insightful information about Bitcoin blockchain users.
However, to our knowledge, there exists no tool capable of automatically tracking a bitcoin through all the transactions it has been involved in. Our application is thus innovative in that aspect.

GraphSense provides a high-end sophisticated user interface as well as an API to request data. As we want our application to remain easily executable from any device, a solution would be to use their interface version and automate its browsing. However, their interface does not offer enough flexibility for what we want to do. That is why we decided to develop our own visualising tool.

[6] https://www.blockchain.com/api

Thus, in order to retrieve information from the blockchain, starting from scratch would not be judicious as this task is tedious and seems to be already well implemented in every tool described in this section.
Therefore, we decided to base our application on WalletExplorer, since it provides all the information needed to track bitcoins via an optimised API. Indeed, as explained in 3.2.1, being able to retrieve the transaction ID from which the input bitcoins came (i.e. pT, refer to 2.2) is crucial for optimisation and speed purposes, and WalletExplorer is the only blockchain browser to offer that feature. Therefore, we will not use BitInfoCharts or BlockchainExplorer.
Finally, once the transaction data from an address is retrieved, we process it to find out how it got so rich.

3.4 Ethical and Professional Considerations

We aim to identify sources of wealth of Bitcoin addresses. As users are pseudonymous on the Bitcoin network, our work should, in theory, not expose any personal data about a user. However, in the case an address is revealed to be linked to a real identity, the information we determined through our app might affect that person directly.
Therefore, it is crucial to clarify that our results must not be taken as ground truth, that it only estimates someone's wealth and thus may lead to incorrect results.

All the data accessed through our work is publicly available, and should therefore reveal neither sensitive nor private information. We are only interpreting already existing information. However, we decided not to store any data collected via APIs on websites such as Twitter or Reddit. In doing so, we are respecting the users' rights to delete and/or modify anything they have made public in the past.
The only information we decided to store is on-chain transaction information, which is essential for optimisation purposes. Nevertheless, since this data is public as well, and as nothing can be changed on the blockchain without the agreement of the majority of the miners, we consider that this will not cause an issue.

Chapter 4

Fine-tuning the parsing of the blockchain

In this chapter, we first explain how a bitcoin can be tracked on the blockchain. Then, we detail novel heuristics and methods implemented to refine the parsing of the blockchain in order to optimise it, while minimising information loss. Finally, we introduce a new metric to further the transaction analysis: the **R**atio **T**o **O**rigin (RTO)

In the following sections, we use notation introduced in 2.2. It is recalled here: Let $\mathscr{A}$ be the wallet address investigated, $\mathscr{S}$ its total amount of bitcoins received. Unless stated otherwise, we define a transaction T as such:

- n inputs $(A_i, \alpha_i, pT_i)_{i\in[1,n]}$, $n \geq 1$
- m outputs $(B_j, \beta_j, nT_j)_{j\in[1,m]}$, $m \geq 1$

To avoid unnecessary complications in the formulas, we do not take into account the transaction fee in the following paragraphs. Therefore, we assume here that:

$$\sum_{k=1}^{n} \alpha_k = \sum_{i=1}^{m} \beta_i \tag{4.1}$$

This approximation is a reasonable assumption as we will see in Chapter 8 how we tackled that problem in our implementation.

4.1 Tracking a bitcoin

In order to find out where a bitcoin owned by a user comes from, we are going to track back all the transactions it has been involved in.
Starting from wallet address $\mathscr{A}$, only transactions from which bitcoins have been received are analysed. An example of a wallet displayed on WalletExplorer is illustrated in Figure 4.1. Received transactions are indicated in green and sent transactions are indicated in red. Corresponding transaction IDs can be found in the right column.

Bitcoins received by that address are outputs of these received transactions. So, for each of the latter, we have a look at its inputs since they indicate where these

date	received/sent	balance	transaction
2015-11-28 03:03:10	-1.	0.	0e483cd762184c211a5cd8393872a83a03c463ae89084484b01839d85338579b
2015-11-27 18:57:32	+1.	1.	29f06f692da93db88f99a2d4c835463e05172a86c9a567b2ba8a30da4ef51fd1
2015-03-23 02:30:03	-0.07608991	0.	e651359f879a6b345064049e431a014922546b292ba41d883d766f1abea02936
2015-03-18 15:38:01	+0.07608991	0.07608991	99fd988bf60ff67847488ceeb76d08a8fcca7bde80bb0b06be2ef4a0055c3ba7

Figure 4.1: Extract of a wallet as it is displayed on WalletExplorer.

bitcoins are from.
This process is repeated until we find a transaction that has been identified, or when other stop conditions -detailed in the following paragraphs- are triggered. Each bitcoin's transaction history is thus established.
We consider a transaction *identified* if its input addresses have been tagged. In that case, we stop the parsing for that transaction and conclude the initial bitcoins tracked all the way to that one can be associated with that identity.

4.2 Choosing relevant transactions

4.2.1 Challenges faced

The Bitcoin blockchain allows users to have multiple inputs in their transactions (i.e. $n > 1$). The sender can therefore aggregate bitcoins from his wallet addresses to make a payment. According to data gathered by Bitcoin Visuals [27] and represented in Figure 4.2, transactions have 2.49 input addresses on average.

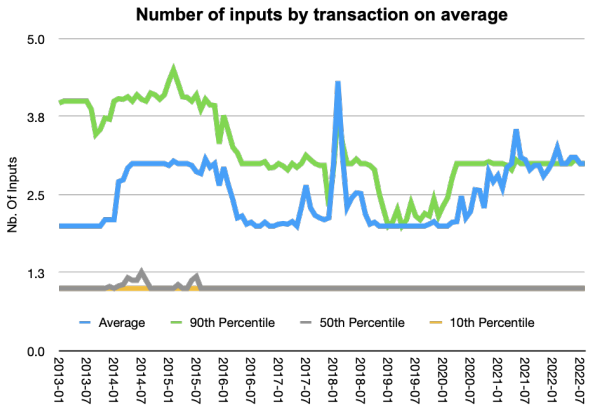


Figure 4.2: Number of transaction inputs on average, by month. 10^{th}, 50^{th} and 90^{th} percentiles are represented as well.

By analysing further Figure 4.2, from the 50^{th} percentile having an average of 1.01, we infer that most transactions only have one input. However, the 90^{th} percentile averaging at 3.21 clearly shows that although most transactions do have only one input, there still exists a significant amount of transactions with a consequent number of inputs. Therefore, if we consider a wallet that received 50 transactions, going through 6 layers would already represent $50 * 2.49^6 = 11,917$ transactions to parse on average.

This raises two main challenges as we go through each transaction:

1. Sometimes, when a transaction has multiple inputs and outputs, it is impossible to figure out for sure which inputs correspond to the output we are tracking. We have to make choices to estimate its origin, which will cause information loss, as the methods used are only approximation processes.
   → **Estimation Challenge**

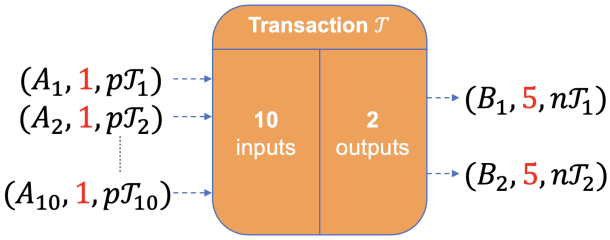


Figure 4.3: Example of an estimation challenge

2. Moreover, some transactions have too many inputs to consider. Since we should theoretically analyse the origin of every bitcoin in input, the number of transactions is likely to grow exponentially, and so is the processing time if we don't apply heuristics to select the most relevant ones.
   We thus need to reduce that execution time by removing transactions. This consequently causes information loss as well.
   → **Pruning Challenge**

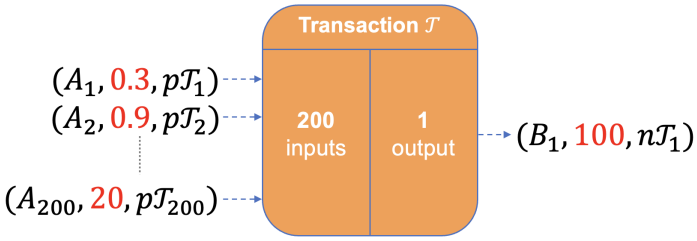


Figure 4.4: Example of a pruning challenge

These two challenges are not mutually exclusive, meaning we can face both in one transaction. In our heuristics, we decide to solve the first challenge in priority, and then the second one if the former can't be solved. In the case both challenges are faced, our heuristics are designed in a way that solving challenge 1 solves challenge 2 as well.

4.2.2 Pruning process

In order to optimise our analysis, we first studied the main types of transactions that can be encountered during the parsing only regarding their number of inputs. We classified them into three sections:

- **Transactions with only one input ($n = 1$):** This is the most classic case. The number of outputs does not matter, since the bitcoins we are interested in can only come from that input. No fine-tuning is necessary as we only need to follow that one.
- **Transactions with a reasonable number of inputs ($n < 20$):** These transactions are likely made by users for personal use to aggregate bitcoins from multiple addresses to make a payment, or to simply regroup them on a unique one. In most cases, all inputs are relevant and could be considered, but we may refine them to only keep the most relevant ones.
- **Transactions with a significant number of inputs ($n > 50$):** Transactions likely made by professionals or entities. These cause an issue as there are too many inputs to continue the parsing with. Therefore, they need to be refined.

It is important to notice that in the last two scenarios, the challenges faced depend on whether transactions have one or multiple outputs. The pruning challenge will be faced regardless of the number of outputs, whereas the Estimation challenge will only be faced if there are more than one output.
To deal with these two scenarios, we decided to create our own heuristic to prune transaction inputs. Figure 4.5 depicts it in the shape of a decision tree.

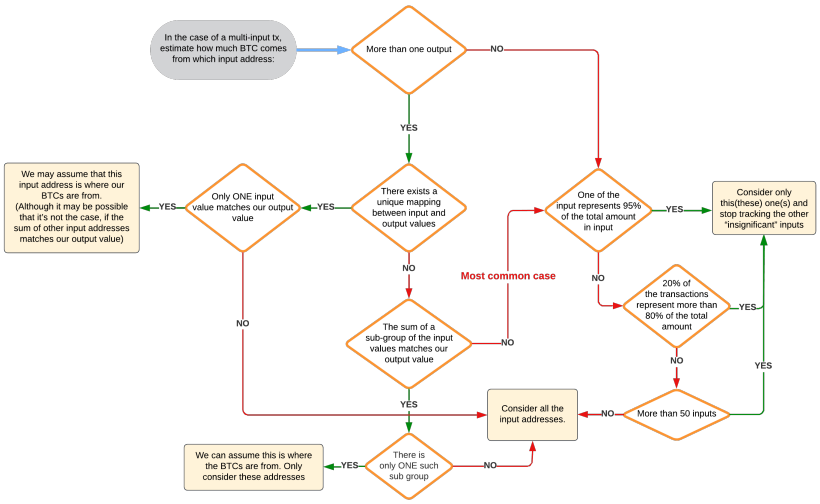


Figure 4.5: Overview of the decision tree implemented to prune transaction inputs. A complete version can be found in Appendix A.1

First, we focus our analysis on **multiple input and multiple output** transactions. Input and output values are interpreted and compared with each other to detect two patterns:

1. We check whether input values completely correlate with output values, meaning there is a unique mapping between the input and output addresses. We thus take into consideration every input and every output:
   Let f be the mapping function such that $\forall (i, j) \in [1, n],\ f(A_i) = B_j \implies \alpha_i = \beta_j$.

We require $n = m$.

$$\begin{aligned} \text{Pattern 1 is verified} &\iff \forall j \in [1,n],\ \exists! i \in [1,n],\ B_j = f(A_i) \\ &\iff f \text{ is bijective} \end{aligned} \tag{4.2}$$

Unfortunately, this pattern may rarely be detected, as it only represents a very small proportion of the transactions. However, if it is confirmed, we can conclude rather safely that the input corresponding to our bitcoins in output is the one we need to continue the parsing with.

2. To further our analysis, we introduce a second pattern recognition: if the first pattern recognition was unsuccessful, we determine whether the sum of a subgroup of the input address values matches our bitcoins in output. As this algorithm does not depend on other transaction outputs, we can define the output we are tracking as (B, β).

$$\begin{aligned} \text{Pattern 2 is verified} \iff \exists! l \in \mathbb{N}^*,\ \exists! (i_1, i_2, ..., i_l) \in [1,n]^m, \\ i_1 \neq i_2 \neq ... \neq i_l \ \ and \ \ \sum_{k=1}^{l} \alpha_{i_k} = \beta \end{aligned} \tag{4.3}$$

In order to avoid too many false positives, we insist on the subgroup being unique. If not, we decide not to apply this pruning method.

Although there is no guarantee that our patterns are flawless and precisely identify the inputs from which our output bitcoins come, we consider these two approximation methods to be quick and efficient in most cases.
However, if none of the patterns explained above are present in the transaction, we still try to reduce the number of inputs to consider by implementing one last rule. This rule may be applied whether the transaction has one or more outputs, as it is not dependent on the latter.

3. By browsing several transactions, we noticed a recurring pattern illustrated in Figure 4.6. Indeed, often, inputs are composed of a value far superior to the others. We can infer that the former may be more relevant than the latter. We can define the problem in the following way:

$$\text{Pattern 3 is verified} \iff \exists i \in [1,n],\ \alpha_i > 0.95 * \sum_{k=1}^{n} \alpha_k \tag{4.4}$$

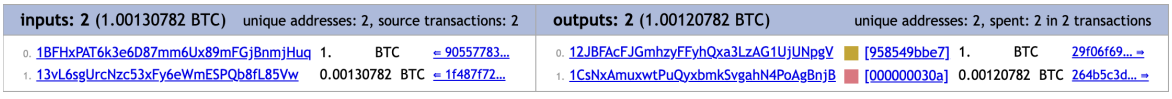


Figure 4.6: Example of a transaction verifying the pattern 3.

Finally, to overcome the challenges described in Chapter 8 regarding request limits, two rules had to be added:

4. If 20% of the input transactions represent more than 80% of the total amount, we only consider these.

5. If a transaction has more than 50 inputs, we only keep the 50 highest.

Unfortunately, these two new rules cause an important loss of information as we see in Section 4.4. However, we believe this is an acceptable compromise between speed and accuracy.

Overall, in this section we introduced rules reducing the number of transactions taken into consideration in each layer while keeping as much information as possible, thus optimising the parsing vertically.
However, as we are progressively going deeper into the layers, information is inherently lost due to choices that were made to speed up the browsing. Therefore, it may lead to irrelevant transactions being kept, as each transaction is only optimised according to that transaction itself, and does not depend on any transactions previously encountered. Moreover, so far, we have no way of estimating the proportion of our initial bitcoins each transaction represents.
That is why we decided to implement a new metric: the RTO score.

4.3 Ratio To Origin Score

The **R**atio **T**o **O**rigin (RTO) score is a metric we created to estimate how much of the initial bitcoins are present in a previous transaction. Each transaction we encounter during the parsing is attributed a RTO score (represented either in bitcoin or in the percentage of the initial total amount $\mathscr{S}$), corresponding to the estimated proportion of initial bitcoins that transaction contains.
We implemented a very simple approach to calculate the RTO score of each transaction:

1. First, we apply heuristics explained in Section 4.2 to select inputs of a transaction T we want to keep parsing. To simplify the notation, we consider that the k first inputs are selected ($1 \leq k \leq n$) (Otherwise, we change the input order to fall back on that notation). Likewise, we define $\alpha = \sum_{j=1}^{k} \alpha_{i_j}$

2. The previous transaction pT_i of each input kept $(A_i, \alpha_i, pT_i)_{(1 \leq i \leq k)}$ receives then a ratio of the RTO score of T weighted by its own value α_i:

$$RTO(A_i) = RTO(T) * \frac{\alpha_i}{\alpha} \tag{4.5}$$

Regarding the transactions received by the wallet address, we set their RTO score to the amount of bitcoins that address received from that transaction.

A simplified example is illustrated in Figure 4.7.
First, we set the RTO score of the transaction T_1: As 20 bitcoins were received in total

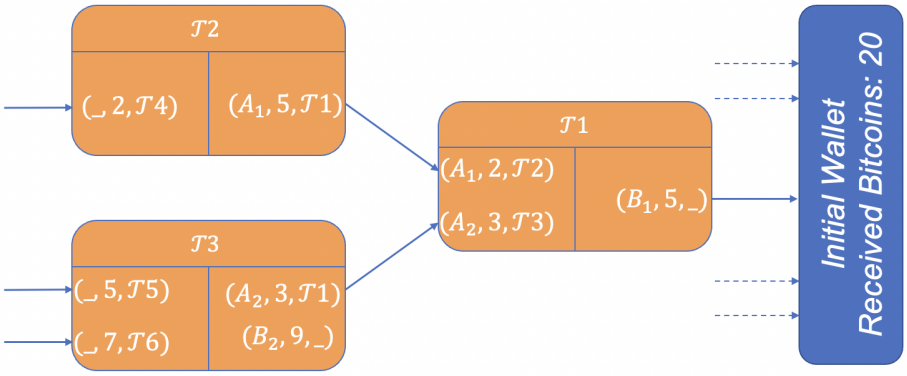


Figure 4.7: Simplified example of the calculation process of the RTO score.

and T_1 is responsible for 5 of them:

$$\begin{aligned} RTO(T_1) &= 5 \text{ bitcoins} \\ &= 25\% \end{aligned} \tag{4.6}$$

Then, we apply the rules explained above to calculate the RTO scores of T_2 and T_3:

$$\begin{aligned} RTO(T_2) = \frac{2}{5} * RTO(T_1) &= 2 \text{ bitcoins} \\ &= 10\% \end{aligned} \quad \Big| \quad \begin{aligned} RTO(T_3) = \frac{3}{5} * RTO(T_1) &= 3 \text{ bitcoins} \\ &= 15\% \end{aligned} \tag{4.7}$$

Finally, we can do the same calculations for T_4, T_5 and T_6:

$$\begin{aligned} RTO(T_4) &= 2/2 * RTO(T_2) = 2 \\ RTO(T_5) &= 5/12 * RTO(T_3) = 1.25 \\ RTO(T_4) &= 7/12 * RTO(T_3) = 1.75 \end{aligned} \tag{4.8}$$

Our RTO metric is built so that for each transaction layer, the sum of all the RTO scores of these transactions is equal to 100% (or $\mathscr{S}$), provided every parsed transaction has been kept.

Now that the RTO score allows us to keep track of the proportion of the initial bitcoins in each transaction, we can set a threshold so that if the RTO of a transaction gets under that limit, we stop prematurely its parsing.

4.4 Evaluation

To evaluate our heuristics, we have a look at an address[1] involved in a romance scam. To this date, this address has received more than 50,000 bitcoins in approximately 16,000 transactions, so this puts the robustness as well as the efficiency of our methods to the test.

[1] 3JMjHDTJjKPnrvS7DycPAgYcA6HrHRk8UG

We performed an analysis on that address, keeping track of the amount of RTO kept in each layer and the number of transactions pruned either because of our heuristics or because the RTO amount dropped below the threshold.
Results are gathered in table 4.1.

	Tot. TX		**TX removed**		**Tot. RTO**	
Layer	*0.1%*	*0.01%*	*0.1%*	*0.01%*	*0.1%*	*0.01%*
0	16,319	16,319	0 (16,030)	0 (14,578)	45.8 %	80.6%
1	393	4,551	49 (66)	1,766 (1,102)	36.9%	68.2%
2	1,312	7,231	812 (400)	4,114 (2,048)	22.6%	49.3%
3	514	7,724	291 (163)	4,415 (2,548)	19%	43%

Table 4.1: Number of transactions parsed, removed and total amount of RTO by layer, according to two different RTO thresholds, 0.1% and 0.01% on "3JMjHDT...".
First number in "TX removed" column represents the amount of transactions not selected, and the second one the amount of transactions whose RTO were too low.

As expected, the last two rules introduced in Section 4.2 for performance reasons reduce a lot the RTO kept from one layer to another.
However, if we look at an analysis on an address with a more reasonable number of transactions depicted in Table 4.2, the total RTO decreases much more slowly.

Layer	**Tot. TX**	**Tot. RTO**
0	15	100%
1	22	100%
2	31	99.74%
3	28	99.73%
9	363	94.4%

Table 4.2: Number of transactions parsed and total amount of RTO by layer, with a RTO threshold set to 0.1% for the address "115ZFznB...".

4.5 Conclusion

As the number of transactions to browse grows exponentially the deeper the layer is, we implemented heuristics to optimise its parsing by studying the types of transactions made on the Bitcoin blockchain.
Once we were able to identify recurring patterns and group them into different types, we designed rules adapted to each transaction group identified to accelerate the parsing while minimising the amount of information lost.
The evaluation of these heuristics has proven to be promising when considering an address with a reasonable number of transactions. However, when the address becomes too big, information dissipates through the transaction layers much more quickly in favour of the processing speed. In that case, estimating the origin of bitcoins might become less accurate.

Now that we are able to go through the layers and select relevant transactions, it is crucial to interpret the data we found, since our goal is to identify the origin of an address' wealth. That is why we introduced another novel metric: the Closeness to Service.

Chapter 5

Closeness to Service

Once initial bitcoins have been backtracked all the way to an identified transaction, we then consider that these bitcoins originate from that entity.
However, depending on how many transactions there are between the user's wallet and that entity, we can't always conclude with the same degree of certainty on the link between the former and the latter. Indeed, the situation shown in Figure 5.1 can be interpreted in two different ways:

1. The wallet owner directly received money from a BlackMarket transaction in Transaction T_2 on an address not linked to the investigated address, and then transferred it to its main wallet during T_1. They are therefore directly involved with the entity discovered.

2. The wallet owner sold an item online in bitcoins to another user whose bitcoins came from a BlackMarket transaction. In that case, the owner is the victim as he did not know the origin of these bitcoins. Therefore, he should not be linked to that entity.

This situation clearly shows that it is impossible to confidently know the source of wealth of a user. This ambiguity must be addressed by introducing a degree of closeness between a transaction emitted by a service and the user's wallet: the Closeness to Service.

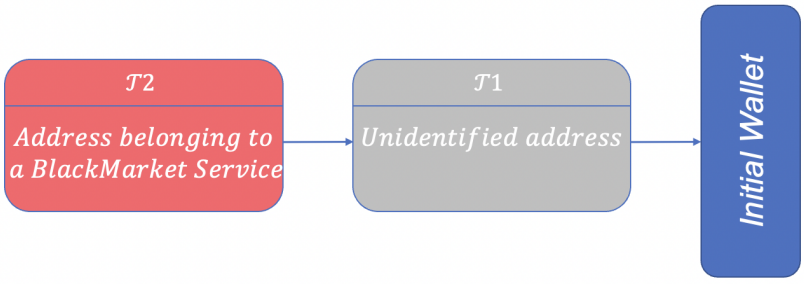


Figure 5.1: Example of an ambiguous situation

In this chapter, we first explain what services are and which are considered. We then introduce the concept of closeness to a transaction and explain why this is a crucial concept to estimate a bitcoin's origin. Finally, we introduce our new metric

called **C**loseness **T**o **S**ervice (CTS) which allows the interpretation of the data we collected while parsing the blockchain.

5.1 Services

According to Interpol, a service "refers to some software functionality or a set of software functionalities that can be used by different actors for different purposes" [28]. In the case of the Bitcoin blockchain, we can define a service as an entity controlling addresses to send/receive bitcoins to/from users.
For the sake of our project, we decided to use WalletExplorer's list of identified services, as this is the blockchain browser we use. However, it is important to remember that the service name database has not been updated since 2016, so it does not contain newer services or newer wallets of existing services.

We can cite a few service types identified by WalletExplorer:

- **Wallet Hosting Services:** Platform on which users can store their bitcoins as well as send/receive transactions (e.g., Binance, Cryptsy).
- **Mining Services:** Platform allowing users to form a pool to mine blocks. If successfully mined, freshly generated bitcoins are then sent to the users in proportion to the effort made (e.g., SlushPool, BTCCPool).
- **Gambling Services:** Platform offering users to gamble their bitcoins to potentially earn rewards such as online casinos (e.g., SatoshiDice, CryptoGames).
- **DarkMarket Services:** Platform providing users with a way to sell/buy illegal items. As with any other platforms listed above, it needs addresses to send/receive bitcoins to/from users (e.g., SilkRoad2Market, AgoraMarket).

Service types are not limited to this list, but this already gives us an idea of what kind of services there are and what their purpose is.

5.2 Transaction Closeness

We can define transaction closeness as the proximity between that transaction and the investigated address. We decided to quantify it as the smallest number of transactions between the former and the latter:

- Let $\mathscr{A}$ be the address investigated.
- Let T_k be the transaction we want to determine its closeness with.
- Let $T_1, ..., T_i$ be normal transactions different from T_k. $(i \geq 1)$
- Let $\mathscr{O}(T_j)$ designate the next transactions[1] of T_j. To simplify the notation, we write $\mathscr{A}$ in $\mathscr{O}(T_j)$, meaning $\mathscr{A}$ is an output address in the transaction T_j.

[1] Refer to the definition of a transaction in Section 2.2

We can thus define the Transaction Closeness TC between $\mathscr{A}$ and T_k as:

$$\mathrm{TC}(T_k, \mathscr{A}) = \begin{cases} 0 & \text{if } \mathscr{A} \text{ in } \mathscr{O}(T_k) \\ 1 + min(\mathrm{TC}(T_j,\ \mathscr{A}))_{T_j \in \mathscr{O}(T_k)} & \text{otherwise} \end{cases} \tag{5.1}$$

Remark: In our implementation, we only calculate the closeness of a transaction to which a path has been found from the wallet address. Therefore, we don't consider any cases where such path does not exist.

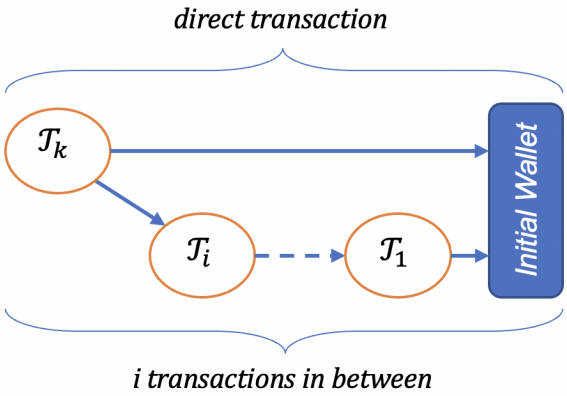


Figure 5.2: Transaction Closeness Example

In the example pictured in Figure 5.2, we see there are two possible paths to go from the initial wallet to the desired transaction T_k, a direct path and a path with i transactions in between. The transaction closeness is therefore **0**.

5.3 Closeness to Service

Now that we have introduced transaction closeness, we just need to restrict it to identified transactions in order to define the **C**loseness **T**o **S**ervice (CTO).
Therefore, we are now able to quantify the relation between an address and Services from which their bitcoins came from.

To illustrate its use, we analysed the origin of bitcoins owned by the wallet address "115znB6..."[2]. This address is owned by Cryptsy, an old online cryptocurrency exchange platform [29] whose CEO ran away with more than 3 million dollars in 2015 [30]. By studying the pattern of the transactions, we infer that this address was used by users to deposit their bitcoins.
So, here, the question at stake is not to find out who owns that address, but rather what kind of bitcoins were deposited on that platform, and thus what kind of bitcoins users were likely to withdraw.
By applying parsing heuristics previously defined in Chapter 4, we went through 6 layers of transactions, representing 86 transactions parsed, and we successfully identified the origin of an estimated amount of 5.5 bitcoins out of the 10.37 received.

[2]https://www.walletexplorer.com/address/115ZFznB6rTteLDF18AQTf2SWNBtoywoxb

The CTS summary of these bitcoins can be found in Figure 5.3. *AgoraMarket* and *SilkRoad2Market* are both BlackMarket Services, having a RTO score of 14.45% and 1.21% respectively. Thus, we could conclude that users tried to exchange their "dirty" bitcoins on that platform. However, as the *SilkRoadMarket* CTS is equal to 5 (i.e. 5 transactions between our address and that transaction), one has to be careful before making that conclusion.

Received BTCs	BTC from the root add.	Closeness* to these tags
Cex.io	1.52 BTC (*14.62%*)	0 - 12.12%% | 1 - 2.5%
SlushPool.com	1.0 BTC (*9.64%*)	3 - 9.64%
Bitstamp.net	1 BTC (*9.64%*)	3 - 9.64%
AgoraMarket	1.5 BTC (*14.45%*)	3 - 14.45%
SilkRoad2Market	0.13 BTC (*1.21%*)	5 - 1.21%

Figure 5.3: CTS recap

The number of transactions parsed by layer as well as the evolution of the tagged bitcoins can be found in Figures 5.4 and 5.5 respectively. The transaction graph generated can be found in Appendix B.1.

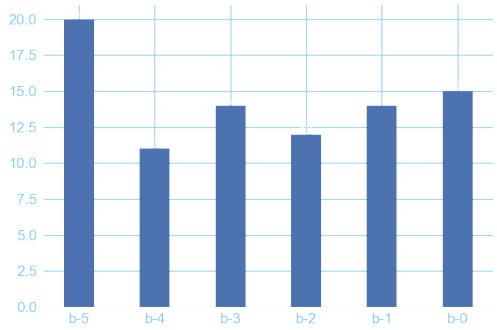


Figure 5.4: Number of transactions by layer.

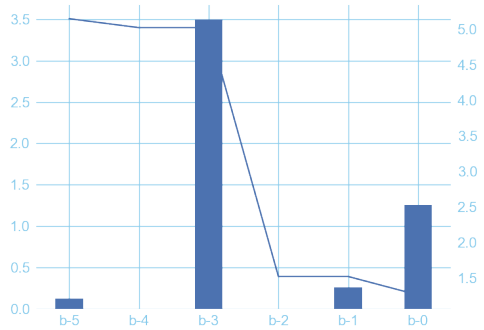


Figure 5.5: Number of tagged BTC by layer (bars) and total tagged BTC (curve)

5.4 Evaluation

As shown in this example, the closeness to service provides a solution to quantify a relation between an address and the entities from which its bitcoin came from. However, we are limited in the way that WalletExplorer does not provide the most up-to-date address identification data, as names have not been updated since 2016. Moreover, WalletExplorer has been tricked into thinking some addresses belonged to the same wallet because of a CoinJoin transaction [31]. The wallet has been ironically called "CoinJoinMess" and gathers more than 10 million transactions [32]. This shows the limits of such clustering heuristics and lays emphasis again on how difficult analysing blockchain is.

However, to date, we believe that there is no other way to measure this. It is therefore complicated to compare it to other metrics.

5.5 Conclusion

In this chapter, we demonstrated the importance of quantifying a relation between an entity and an address. Indeed, the more transactions there are between a service and the address, the less significant that relation is. That is why we decided to implement the Closeness to Service metric.

However, browsing the blockchain and compiling statistics is not the only way to find information about the origin of the wealth of an address. In the next chapter, we describe how we look for any useful information off-chain, on the web.

Chapter 6

Scraping information on the Web

Efficient web crawlers have already been implemented to find as much information as possible on addresses to identify them by scraping forums and websites. This is what D. Ermilov et al. presented in their paper to collect address tags [13].
However, we decided not to use any already existing web crawlers, as our project does not mainly focus on identifying addresses, but rather on tracking transactions.
Therefore, in this part, we only discuss already existing sources of information, and we detail how we used them.

In our work, we decided to focus our attention on finding any hints on whether or not an address has been involved in scams. To do so, we used four main sources of information: Twitter, Reddit, GoogleSearch and BitcoinAbuse. We detail the the design of our web parser in this chapter.
We evaluate its relevance by analysing a Romance Scam address "3JMjHD..."[1]. This address has been involved in scams operated on dating apps and other social platforms.

6.1 Detecting Scammers with BitcoinAbuse

6.1.1 Theory

BitcoinAbuse is a public database of bitcoin addresses used by hackers and criminals, gathering close to 300,000 reports. Anybody victim of ransomware, blackmailers or fraudsters can file a report to shed light on a new scam or confirm the existence of an already established one.
This represents an important source of information that is not taken into account if we only consider on-chain data. Indeed, finding out that the wallet address investigated has been reported as a scammer proves to be really helpful in understanding how an address became so rich.
However, allowing anybody to file a report has its drawbacks as shown in Figure 6.1. Indeed, ironically, it seems that it has been used by malicious users to advertise their own scams.

The first report praises the effectiveness of a website that could *allegedly* help recover your stolen bitcoins. This is definitely not a genuine report.

[1] 3JMjHDTJjKPnrvS7DycPAgYcA6HrHRk8UG

Date	Abuse Type	Description
Aug 2, 2022	ransomware	Good work deserves recommendation... i lost over 2.3 BTC on Instagram bitcoin scam.. Right about 2 weeks after my ordeal with them I tried using the recommendation from someone on one of the comment section www.coinabuser.com I was able to get all my money back in less than 48 hours.. Contact www.coinabuser.com to recover all your stolen bitcoins free of charge
Aug 2, 2022	blackmail scam	The address is linked to a romance trading scam operated in South East Asia. The money received on these addresses belongs to victims and will be used by criminals for money laundering. Do not accept any transaction from this address. We ask crypto exchanges to refuse operations from these addresses or freeze the wallets. The scam brokers are www.nysetdfvq.cc

Figure 6.1: Reports made about a romance scam address on *www.bitcoinabuse.com*.

The second report, on the other hand, warns that this address is linked to a romance trading scam operated in South East Asia. This is likely a genuine report.

6.1.2 Design

In order to discern genuine reports from fake ones, we first thought of resorting to a **N**atural **L**anguage **P**rocessing (NLP) classifier. This is a Machine Learning process categorizing text sentences into different groups, or classes. In our case, only two classes are needed: 1) genuine report and 2) fake report.
However, it turned out that using regular expressions for the classification task was already precise enough for what we wanted to do. As it was faster and simpler, we opted for that option instead.
To correctly label reports, we analysed fake ones to extract a pattern. It turned out that they are mainly distinguished from genuine reports by the presence of specific keywords that were usually not used in the latter (e.g., "recover"). We then established a list of keywords that would automatically classify a report as fake.
Although some reports might still be misclassified, we believe this won't be an issue considering that the user can still access all reports and examine them themselves. Moreover, we leave the user the possibility to edit this list, as we believe it is important that one can adapt it according to their needs.

However, as can be inferred from the report information on the romance scam address in Figure 6.2, only 9 reports have been filled for more than 16,000 transactions made. This shows that only few victims report a scam address. Therefore, it may seem appropriate to look for information on other websites.

6.2 Finding hints on Twitter, Reddit and GoogleSearch

As explained in the previous section, being able to look for information everywhere on the web can be a game changer to find out where the wealth of an address comes from.
Therefore, we decided to focus on scraping three websites:

- **Reddit:** We believe Reddit has become a popular place to talk about cryptocurrencies in general, but more specifically to discuss abnormalities detected on the blockchain.

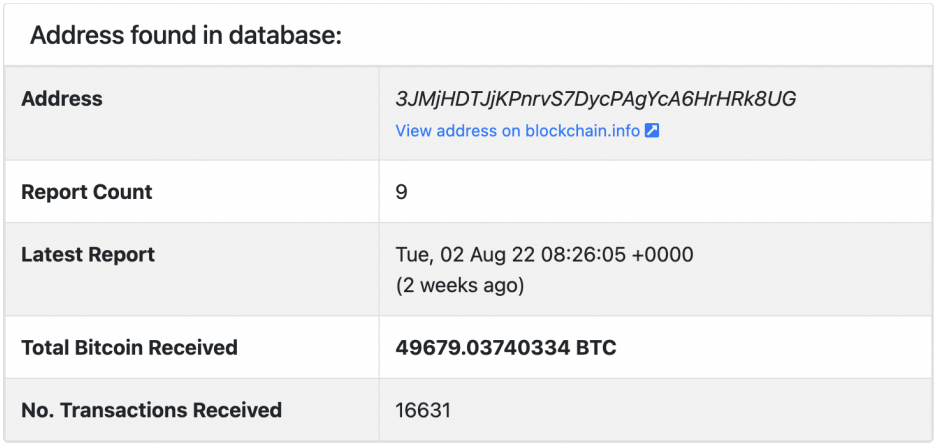


Figure 6.2: Recap of reports made on a romance scam address on *www.bitcoinabuse.com*.

- **Twitter:** Similarly to Reddit, Twitter is a social network wildly used to talk about bitcoins, and thus anything Bitcoin related, such as the identification of addresses or the report of a scam. To make results user-friendly, we decided to limit the results to only Tweets written in English and that are neither quotes of a tweet nor a retweet.
- **GoogleSearch:** Finally, we decided to implement a basic GoogleSearch as it allows the user to make sure no crucial information has been missed on the address. Indeed, we set it up so that the results shown should be the ones a user expects if they perform a normal Google research.

In order to display relevant Google search results only, we decided to implement a filter based on the lexical field of cryptocurrencies.
However, similarly to BitcoinAbuse, we made these keywords easy to edit, so that users can adjust them to focus their search on what they are looking for. This allows for more flexibility, which is a key criterion in research.

6.3 Evaluation on a romance scam

6.3.1 BitcoinAbuse

To evaluate our web scraper, we look for information on the romance scam address cited in the introduction of this chapter.
In Figure 6.2, we already saw that 9 reports had been filed for that address. By applying our filtering method, the algorithm concluded that of these 9 reports, only 6 are likely to be genuine ones. A closer look at the reports reveals that 7 reports actually look genuine, meaning that they were likely not filed by scammers.
Although these results are good enough for their use, users can fine-tune the filtering method by changing regular expressions used or adding new keywords if they want to target particular reports.

Finally, although fraudulent reports can be distinguished from the likely genuine

ones, it is impossible to know whether a genuine report has actually been written faithfully, or it was an act of revenge from someone who did not like the address owner and decided to write a fake complaint, for instance.
However, we believe that this concern can be addressed by looking at the number of reports made and their date of publication. Indeed, if an address has only been reported once or twice on the same day at the same time, we might not be able to conclude anything about its fraudulent activity. On the other hand, if a report has been reported 10 times over a period of several days, and the reports look genuine, we may deduce that the address belongs to a scam.

6.3.2 Twitter, Reddit and GoogleSearch

When querying our romance scam address on these three websites, information is found on Reddit, but also on Google as shown in Figure 6.3.

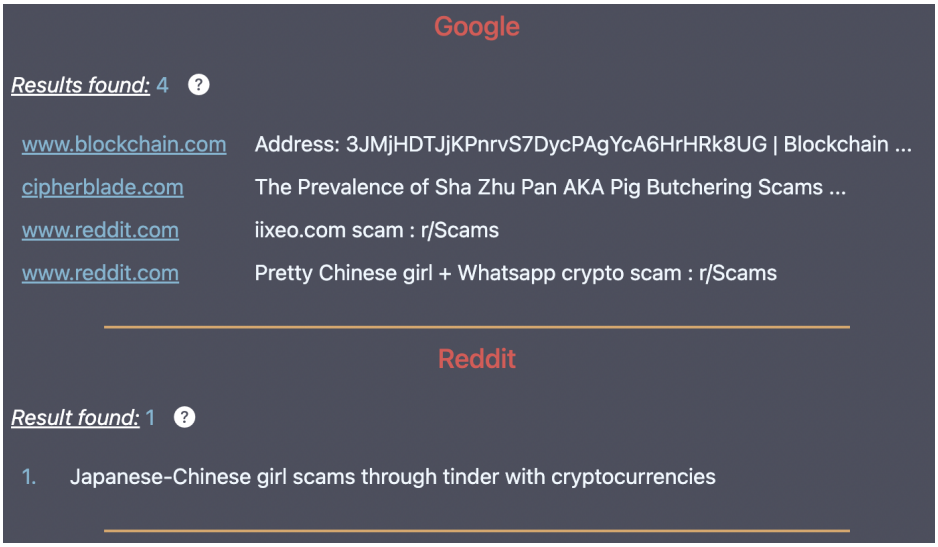


Figure 6.3: Results of a query on the romance scam address.

We immediately notice that Google returns two Reddit results that are not displayed via the Reddit search. This is because the latter only displays results according to post titles, and not their comments. This proves that it is crucial to compare several sources of information in order not to miss any.

On a side note, parsing the blockchain for that address revealed it received more than 16,000 transactions, and that _% of its bitcoins come from Exchange platforms such as Binance and Huobi, which could lead us to think that these transactions come from individuals, so most likely people who have been scammed.

Overall, we implemented our own web parser to focus on the sources we thought were relevant for our project. Although the internet does not provide accurate data all the time, we believe that, by providing the user with multiple sources of information,

they can assess the veracity of this information on their own, and in our case, conclude that this address is really part of a romance scam.
Moreover, this information can't be found on-chain, so we believe it is worth providing it, regardless of the number of hints revealed.

Chapter 7

BTC Tracker

In the previous chapters, we presented novel metrics and heuristics to improve the parsing of the blockchain to track bitcoins and find information about the targeted address
In this chapter, we put it all together to form BTC Tracker, a tool to analyse an address and thus try to understand how it became so rich.

We break down our tool into three parts:

1. **Blockchain Parsing:** Performs an address analysis by browsing the blockchain thanks to heuristics and novel metrics described in Chapters 4 and 5
2. **Off-chain Information finding:** Gathers information found off-chain, analyses them and displays only relevant results through processes described in Chapter 6
3. **Summary of results and display of statistics:** Takes into account both results from off-chain and on-chain parsing to compile statistics.

7.1 Blockchain Parsing

7.1.1 Backward Parsing

In the left part of the interface depicted in Figure 7.1, the user enters an address they want to investigate, the number of layers they want to go through and the RTO threshold wanted.
Once the analysis has started applying heuristics described in Chapter 4, the transaction graph is progressively built on the right part of the interface, displaying critical information such as the tag of the transaction, its RTO, or the link to the WalletExplorer transaction to allow further manual investigation.

Moreover, to provide even more flexibility, a manual mode has been implemented. It consists of an incremental parsing pausing after each layer and letting the user select which transactions they want to continue the parsing with, according to what they think is relevant.
Finally, the transaction graph can be saved and downloaded by the user to analyse it later if needed.

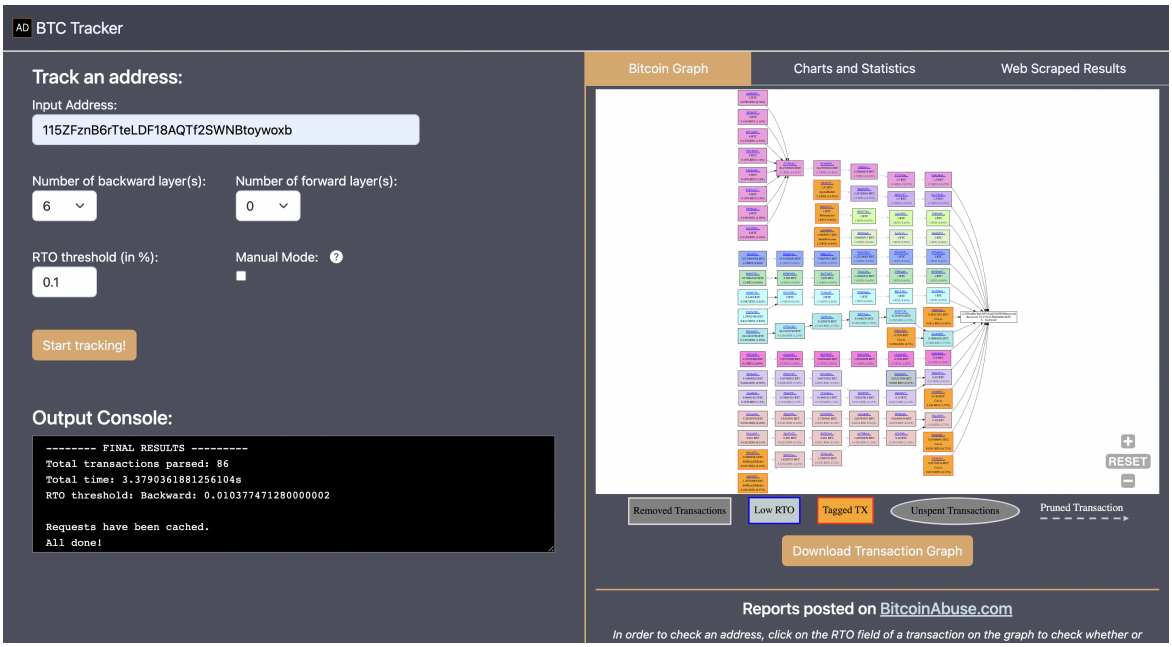


Figure 7.1: BTC Tracker parsing interface

7.1.2 Forward Parsing

In addition to the backward parsing, we realised that analysing where bitcoins of the investigated address were sent also represents a valuable source of information. Indeed, discovering that all bitcoins of an address are directly sent to a mixing service could raise doubts about the legality of the transactions.
Therefore, we also implemented a feature to go forward in the transaction graph. It is very similar to backward parsing except that bitcoins are not necessarily spent. So, the parsing has to stop when bitcoins don't have any next transaction IDs in a transaction output (refer to section 2.2 for the vocabulary). An example of forward parsing graph is shown in Figure 7.2.

This forward tree reveals that this address' bitcoins have been used on the Blackmarket with a Closeness to Service of 3, which sheds light on the type of transactions in which the user was involved.

In both cases, the output console provides essential information about the current step and the number of requests left to make, but also useful results such as the number of parsed transactions or the RTO threshold in bitcoins.

7.2 Off-chain information gathering

The top right tab on the right side of the interface gathers all information found off-chain on Reddit, Twitter, GoogleSearch and BitcoinAbuse whose search designs have been described in chapter 6. A full web parsing result example can be found in Appendix C.1.

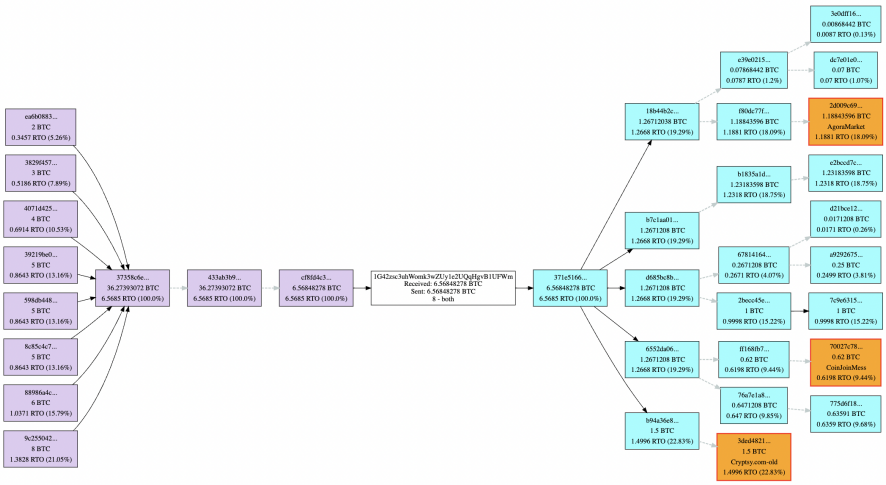


Figure 7.2: Forward parsing example on an address found on Cryptsy's transaction graph shown in Appendix B.1

.

Moreover, we believe that looking for hints only on the target address is not enough. Indeed, it may happen that, at first glance, this address is as pure as the driven snow. But if we can somehow prove that it has received transactions from a scamming scheme, we could thus deduce its wealth likewise comes from a scam.
That is why, for each transaction displayed on the graph, users can request reports of the input addresses on BitcoinAbuse by simply clicking on the RTO line. If any, results will be displayed below the graph.

7.3 Display of results and statistics

Finally, the middle tab on the left side of the interface gathers all the results regarding the on-chain parsing such as the number of transactions by layer and number of tagged bitcoins by layer represented in Figures 5.4 and 5.5, but also the recap of Closeness to Service for both backward and forward layers. An example can be found in appendices D.1 and D.2.

Chapter 8

Implementation

In this chapter, we discuss our implementation choices regarding the different tools and the application we developed.

8.1 Chain parser

The chain parser script was entirely coded in Python. We chose this language because it provides several libraries and is flexible enough in terms of types. Moreover, we can afford to lose speed compared to C++ for instance because performance is limited by the API request limit.
Indeed, as explained in section 3.3, we decided to use WalletExplorer to parse the blockchain. We requested access to their API and had to pay close attention to their request limit rate of 2 requests per second. Doing more requests would lead to a ban for a few minutes and would then slow down the application. Therefore, it had to be strictly respected. We set a rate of one request every 0.6 second, to make sure we don't exceed the limit.

8.1.1 Storing Data

As we go through the blockchain, transactions encountered have to be formatted and saved to be later accessible and build the transaction graph. To do so, two points had to be taken care of:

- We needed a way to store transactions during the parsing, but we especially needed it to be easily manageable and quickly accessible, as calls on each object are made continuously to first write information, and then access it. Thus, we decided to create our own class called `Transaction`.
  This allowed our code to be flexible, which is very important to add new features in the future.
  Finally, these transactions are stored in a dictionary where keys are the layer numbers and values are a list of transactions.
- We did not want the user to have to do the same requests twice if they wanted to re-generate the transaction graph of an address. To solve this issue, we used `requests-cache`, a library based on `requests` allowing us to store every request

made in a database and thus avoid doing them multiple times. We chose SQLite for the database.
Transactions themselves could have been stored in a database instead of the requests, but the size of one transaction as both a request and an object was exactly the same (48 Bytes). Thus, we decided to use `requests-cache`'s built-in database feature.
Since transactions are immutable, requesting the same transaction ID will always return the same content. Therefore, we don't have to worry about data expiring in the database.

8.1.2 Speeding up the parsing

At first, we treated cached requests like any other requests, meaning requests were made normally, and `request-cache` took care of checking the database whether or not the transaction had already been done before. However, this has proved ineffective as a query to the database was made for each request, which was very time-consuming. Instead, we decided to query all the already-cached requests and store them in a Python list. Then, cached requests can be easily separated from not-cached requests. This considerably increased the speed, as can be shown in Figure 8.1.

Moreover, cached requests don't require to respect the rate limit. So not only can the processing time be greatly sped up by skipping the 0.6 sec wait between two requests, but it can be accelerated even further by implementing parallel computing. We used the `ThreadPoolExecutor` Python native class to implement a thread pool of 8 workers capable of processing 1,213 cached requests per second on average.
Figure 8.1 shows a speed comparison of the total analysis time between different ways of making requests. The comparison was based on the analysis of the first two layers of the romance scam address[1] studied in Chapter 6.

Implementation choice	Time taken	Tx. / sec
No cache, respecting the API limit rate	1,849.8 sec	1.67
Cached checked every time for each request	375 sec	8.22
Customised Cache checker without thread pool	7.04 sec	438.93
Customised Cache checker + 8 workers:	4.7 sec	655.96

Table 8.1: Total analysis time of four different ways of managing cached requests. Test made on 3,083 requests (i.e. first two layers of the romance scam address[1])

All workers share a queue of requests to make. To ensure data consistency, we wait for the queue to be emptied by the workers to continue the parsing process.
The number of 8 workers was chosen after comparing the execution time of pools from 1 to 32 threads. With more than 8 threads, little or no improvement in execution time was detected. This is due to Python's Global Interpreter Lock (GIL) that protects

[1] 3JMjHDTJjKPnrvS7DycPAgYcA6HrHRk8UG

access to Python objects, preventing multiple threads from accessing the same data and thus corrupting it [33]. We therefore considered it unnecessary to choose a higher number of threads, thus avoiding overloading the CPU.
A speed comparison of the time taken to make 3083 requests on average according to the number of threads allocated is shown in Table 8.2.

Number of workers	1	2	4	8	16	32
Time taken (sec)	4.42	3.09	2.69	2.54	2.60	2.5

Table 8.2: Time taken on average to make 3,083 requests according to the number of workers allocated.

The request process could have been sped up even more by implementing a multi processing pool instead, but the request process time has already become very low compared to the total execution time, and this would have required further changes on the data structure, so we decided it was not necessary.

8.1.3 Solving the transaction fee problem

Regarding the implementation of the decision tree described in Chapter 4, we first had to deal with the transaction fee problem raised previously. Indeed, in reality, the fee is subtracted from the total amount of bitcoins in input and the approximation stated in Equation 4.1 does not hold anymore.
Therefore, to be able to test our patterns, we iterated through each output and added the fee to it, considering the latter is not split over several inputs, and we ran the pattern recognition algorithm.
This increased our algorithm complexity by a factor of m, where m is the number of outputs in the transaction being analysed. The pseudo-code of the first pattern recognition can be found in Algorithm 1.

Algorithm 1 Integration of the transaction fee to check pattern #1

```
Require: n ≥ 1
  out_values are the output values of the tx. considered
  in_values are the input values of the tx. considered
  for out_value ∈ out_values do
      out_value += tx_fee
      if out_values = in_values then
          if mapping is unique then
              return input(s) corresponding to our output(s)
          end if
      end if
      Try pattern #2
      ...
  end for
```

8.1.4 Addressing parsing implementation challenges

While implementing the parsing mechanisms described previously, we encountered several challenges that needed to be solved to optimise the parsing.

First, we were unable to implement pattern recognition #2, as its complexity would have been of $O(n * 2^n)$, where n is the number of inputs, since we need to evaluate all the input subsets.
Indeed, although this is the complexity in the worst case, the chances of this pattern being recognised were too low for the computing time it required. We thus decided not to keep it.

Then, while parsing the blockchain, a transaction can be encountered several times on different layers. Indeed, as transactions can have more than one output, one output can appear on a layer and other ones might appear later during the parsing. Thus, we had to be careful not to add it multiple time and most of all, its RTO value as well as relations between transactions had to be correctly updated.
This made parsing much more complex as backtracking was required and new checks had to be implemented.

8.1.5 Building the transaction graph

Finally, we used GraphViz to build the transaction graph as it is compatible with Python [34]. It is an open-source graph visualization software allowing full customisation of the nodes and edges, including the insertion of links on the different labels, which was essential in our case to allow the user to quickly access the transaction on WalletExplorer.
Moreover, the graph can be output in the SVG format, which can be theoretically zoomed in infinitely and therefore perfectly adapted when dealing with deep and complex graphs.

For better portability and readability, the chain parser has been implemented as a class, called `ChainParser`.

8.2 Web parser

Similarly to the chain parser, the web scraper was also entirely coded in Python and is based on the use of APIs.
Thus, we used the `requests` library as well to perform requests on the APIs. However, as explained in chapter 3.4, we decided not to save the content of requests made on the internet to respect users' rights to delete and/or modify anything they have made public in the past.

We used 4 APIs to retrieve information from the web:

- **Reddit:** Public API requiring an authentication token to be used that has to be requested every two hours or so. Request limit of 60 per minute.

- **BitcoinAbuse:** Public API requiring an API key to be used. Limit of 30 requests per minute.
- **Twitter:** Public API, no key or token required. 300 tweet searches every 15 min.
- **GoogleSearch:** Combination of Google Custom Search Engine and API as they do not allow direct Google searches with their API. Use of the API directly on that custom search engine whose limit is 10,000 searches a day. However, the API has a limit of 100 requests a day, which greatly limits the web parser.

As for WalletExplorer, we had to be careful to respect each API's request limit. Therefore, although it would have been beneficial to look up every address encountered during the parsing, it was not possible due to low limits on some APIs.

Credentials are stored in a JSON file allowing the user to connect their own account, but most of all personalise the Custom Search to what they think is more adapted to their use. The addition of new APIs is also made easier as everything is centralised in the same place.

Overall, the web scraper implementation mostly consists of requesting information online through APIs, selecting relevant information via regular expression and formatting the results.

8.3 BTC Tracker

In this section, we describe the implementation of BTC Tracker and the integration of the tools previously presented.

8.3.1 Technology used

We decided to build our application as a website. This seemed to be the best choice since we wanted it to be fully compatible with all operating systems but also highly customisable, easy to use, and lightweight.
To do so, we chose Django to manage both the back-end and the front-end of our application. Django is an open-source high-level Python web framework providing tools and features to ease the creation of database-driven websites [35].

Therefore, the back-end of our application is only written in Python. We used Django channels to communicate with the web page through WebSockets, and its template features to load them.
We combined HTML, CSS and Javascript with Django to build the front-end of our application. We leveraged the following libraries:

- **JQuery:** Javascript library simplifying HTML traversal, event handling and AJAX requests.
- **BootStrap:** CSS framework providing shortcuts and tools to quickly create a web page design.

- **SvgPanZoom:** Javascript library that enables panning and zooming of an SVG in an HTML document, with mouse events or custom JavaScript hooks [36]. It is used in our project for the display of the transaction graph.
- **FileSaver:** Javascript library that enables the saving of a file client-side in a specific file format without having to request it from the server [37]. It is used in our project to make the transaction graph downloadable.

The application diagram is shown in Figure 8.1.

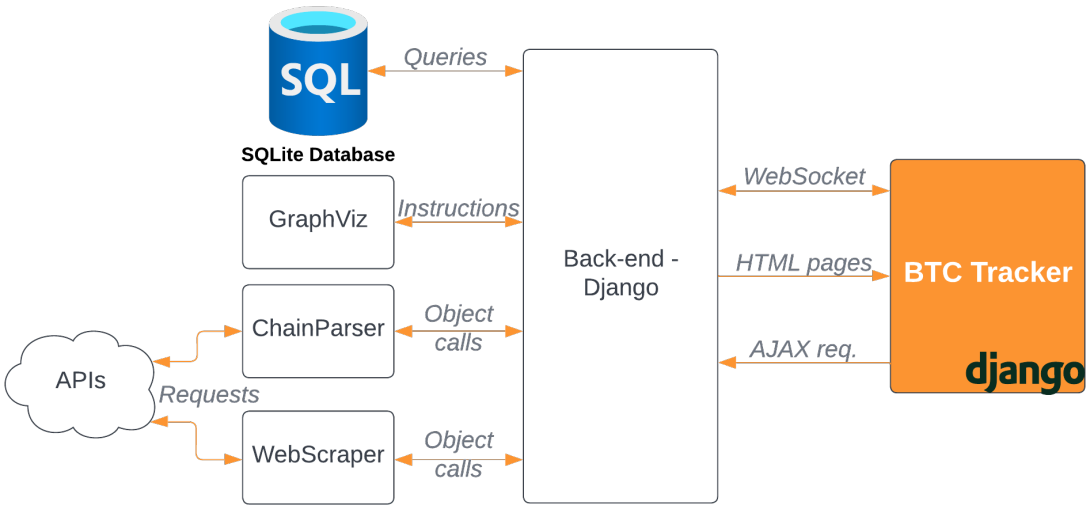


Figure 8.1: BTC Tracker Diagram

8.3.2 Analysis process

The following enumeration describes what happens when a user enters an address along with the number of layers to parse:

1. First, information is sent to the back-end. A ChainParser instance as well as a WebScraper instance are initiated.
2. Twitter, Reddit, GoogleSearch and BitcoinAbuse APIs are all queried to look for information about the address on the web. Information found is sent back to the front end via WebSockets to be displayed in the Web Scraped Results tab.
3. Then, the parsing of the blockchain starts and builds the transaction graph layer after layer. During the parsing, continuous data is sent to the web page via the WebSocket to inform the user on the progress of the analysis.
4. After each layer is done, a temporary transaction graph is built and sent to the front-end.
5. Once the parsing is finished, the final graph is displayed as well as the calculated statistics.

8.4 Implementation Limits

Although our application has overcome every challenge we faced so far, it does show certain limits.
Indeed, one of its biggest limits is the request limit rate which has to be respected. Not only does it restrict the speed of execution, but it also hinders the user from running multiple analyses at the same time on the same machine, as the request limit applies by IP address.
Moreover, although the limit rate of 2 requests per second is respected, we still somehow reach a limit after approximately 500 requests made in a row. It appears that pausing the process for a few seconds (10 seconds to a minute) does not solve that issue, the analysis must be manually restarted by the user. This is not a major problem since requests are cached, so the analysis will quickly pick up where it left off, but it still reduces the tool's autonomy.
As WalletExplorer's Terms of Use need to be followed, the use of proxies or VPNs to circumvent the limit is prohibited.

Additionally, we saw that the graph becomes difficult to interact with when hundreds of transactions are displayed on the same layer. This is because of the SVG format chosen. Indeed, although it offers a lot of flexibility to personalise the graph, we are nevertheless limited with the display.
Every attempt of partitioning the graph into multiple smaller ones to ease its interaction was mitigated as they degraded the user experience. That is why we decided to keep it as a whole. Moreover, this problem only occurs when layers contain several hundreds of transactions, and does not appear when hundreds of transactions are dispatched over several layers.

These limits are addressed in Section 9.2 - Future Work.

Chapter 9

Conclusion

9.1 Achievements

Overall, we created a tool providing visualisations of transaction flow, capable of tracking bitcoins from an address to try to determine how it became so rich, but also allows us to inspect how that money has been spent.
Moreover, we showed that this application has proven to be very useful to track bitcoins in general, and we illustrated it by analysing bitcoins transacted in and out of a romance scam address.
This tool is entirely written in Python and takes the form of a lightweight web application, which makes it easy to understand, easy to improve, and allows everybody to host it on their personal computer and thus use it.

Most of all, this project shed light on how difficult tracking a bitcoin is. Indeed, the blockchain has been designed in such a way that it is impossible to know exactly where a bitcoin comes from. However, we showed that it is still possible to estimate its provenance by developing heuristics to fine-tune the parsing and introducing two novel metrics: the *closeness to service* and the *ratio to origin*.

Additionally, we implemented a web parser to retrieve insightful information not only about the investigated address but also about addresses encountered during the parsing. We believe that finding out who that address has been transacting with can be a game-changer in discovering its source of wealth.

Using these tools, we successfully backtracked bitcoins from the exchange platform Cryptsy to BlackMarket transactions and studied a romance scam by gathering information found on the web and recognizing specific transaction patterns.

9.2 Future work

However, even if the tool is functional and rather efficient as it stands, improvements can be made:

- **Blockchain browsing:** Relying on an external semi-public API to browse transactions is not ideal, especially since it is no longer maintained. Moreover, the request limit greatly impacts the processing speed of our application. Building

our own blockchain browser will solve these issues, and will make our tool more reliable.

- **Gather more address tags:** As we saw with the CoinJoinMess example [32], WalletExplorer's identified clusters might not be fully reliable, and definitely not up to date. Thus, pooling several sources of information about address clusters such as TagPacks would not only allow the identification of more sources, but would also help refine our analysis.
- **Building the transaction graph in the front-end:** Using React for the front-end will enable the use of libraries such as React-flow, a graph-building library. This will likely solve scalability issues and make our graph even more interactive for the user and will allow the addition of more features inside the graph, such as interactive parsing.
- **Switch to a NoSQL database:** As our database is likely to grow, a relational database management system may no longer be suitable for our needs. Instead, as there are no relations between the data, and the former is stored as a dictionary, we can replace it with a NoSQL database such as MongoDB.
  Moreover, in the advent that we build our own blockchain browsing tool, NoSQL is likely adapted to store the 1.6 billion transactions created so far.

Bibliography

[1] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, Oct. 2008. [Online]. Available: www.bitcoin.org.

[2] Bitcoin.org, *Protect your privacy*, Accessed: 2022-08-30. [Online]. Available: https://bitcoin.org/en/protect-your-privacy.

[3] BitInfoCharts. "Top 100 richest bitcoin addresses." Accessed: 2022-05-16. (), [Online]. Available: https://bitinfocharts.com/top-100-richest-bitcoin-addresses.html.

[4] C. Harkin. "Wasabi wallet's coinjoin coordinator to blacklist certain bitcoin transactions." Accessed: 2022-05-16. (Mar. 2022), [Online]. Available: https://www.coindesk.com/tech/2022/03/14/wasabi-wallets-coinjoin-coordinator-to-blacklist-certain-bitcoin-transactions/.

[5] A. M.Antonopoulos, *Mastering Bitcoin*. Dec. 2014, ISBN: 9781449374044.

[6] Y. Zhang, J. Wang, and J. Luo, "Heuristic-based address clustering in bitcoin," *IEEE Access*, vol. 8, 2020. DOI: 10.1109/ACCESS.2020.3039570.

[7] S. Meiklejohn, M. Pomarole, G. Jordan, *et al.*, *A fistful of bitcoins characterizing payments among men with no names*, Dec. 2013. [Online]. Available: www.usenix.org.

[8] M. A. Harlev, H. S. Yin, K. C. Langenheldt, R. R. Mukkamala, and R. Vatrapu, *Breaking Bad: De-Anonymising Entity Types on the Bitcoin Blockchain Using Supervised Machine Learning*. 2018, ISBN: 9780998133119.

[9] Y. Yanovich, P. Mischenko, and A. Ostrovskiy, *Shared send untangling in bitcoin white paper*, 2016.

[10] A. Hayes. "Coinjoin." Accessed: 2022-05-16. (Oct. 2021), [Online]. Available: https://www.investopedia.com/terms/c/coinjoin.asp#.

[11] G. Maxwell, *Coinjoin: Bitcoin privacy for the real world*, 2013. [Online]. Available: https://bitcointalk.org/index.php?topic=279249.0.

[12] M. Möser, *Anonymity of bitcoin transactions an analysis of mixing services*, Jul. 2013. [Online]. Available: https://www.torproject.org/.

[13] D. Ermilov, M. Panov, and Y. Yanovich, "Automatic bitcoin address clustering," vol. 2017-December, Institute of Electrical and Electronics Engineers Inc., 2017, pp. 461–466. DOI: 10.1109/ICMLA.2017.0-118.

[14] BitcoinTalk, *Bitcoin forum*, Accessed: 2022-08-06. [Online]. Available: https://bitcointalk.org/.

[15] B. Bambrough. "Massive hack exposes bitcoin's greatest weakness." Accessed: 2022-05-16. (Dec. 2020), [Online]. Available: https://www.forbes.com/sites/

billybambrough/2020/12/23/massive-hack-exposes-bitcoins-greatest-weakness/.

[16] K. Toyoda, T. Ohtsuki, and P. T. Mathiopoulos, "Multi-class bitcoin-enabled service identification based on transaction history summarization," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1153–1160. DOI: 10.1109/Cybermatics_2018.2018.00208.

[17] Y.-J. Lin, P.-W. Wu, C.-H. Hsu, I.-P. Tu, and S.-w. Liao, "An evaluation of bitcoin address classification based on transaction history summarization," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Mar. 2019, pp. 302–310. DOI: 10.1109/BLOC.2019.8751410.

[18] H. Kalodner, M. Möser, K. Lee, *et al.*, *Blocksci: Design and applications of a blockchain analysis platform*, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/kalodner.

[19] B. Haslhofer, R. Stütz, M. Romiti, and R. King, "Graphsense: A general-purpose cryptoasset analytics platform," Feb. 2021. [Online]. Available: http://arxiv.org/abs/2102.13613.

[20] M. Fleder, M. S. Kester, and S. Pillai, *Bitcoin transaction graph analysis*, Feb. 2015. [Online]. Available: http://arxiv.org/abs/1502.01657.

[21] D. Ron and A. Shamir, *Quantitative analysis of the full bitcoin transaction graph.*

[22] Graphsense, *Github: Graphsense tagpack management tool*, Accessed: 2022-08-30. [Online]. Available: https://github.com/graphsense/graphsense-tagpack-tool.

[23] C. M. Intel, *Blockchain analysis for cryptocurrency markets*, Accessed: 2022-08-16. [Online]. Available: https://markets.chainalysis.com/.

[24] BitInfoCharts, *Bitcoin explorer*, Accessed: 2022-08-16. [Online]. Available: https://bitinfocharts.com/bitcoin/explorer/.

[25] M. Spagnuolo, F. Maggi, and S. Zanero, "Bitiodine: Extracting intelligence from the bitcoin network," in *Financial Cryptography and Data Security*, N. Christin and R. Safavi-Naini, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 457–468, ISBN: 978-3-662-45472-5.

[26] Blockchain.com, *Blockchain explorer*, Accessed: 2022-08-16. [Online]. Available: https://www.blockchain.com/explorer.

[27] B. Visuals, *Extensive charts and statistics*, Accessed: 2022-08-17. [Online]. Available: https://bitcoinvisuals.com/.

[28] Interpol, *Interpol dark web and virtual assets taxonomy.* [Online]. Available: https://interpol-innovation-centre.github.io/DW-VA-Taxonomy/taxonomies.

[29] Cryptsy, *Crypto casinos available online*, Accessed: 2022-08-17. [Online]. Available: https://www.cryptsy.com/.

[30] *Ceo of major online cryptocurrency exchange company indicted for defrauding company's customers, destroying evidence, and tax evasion*, Accessed: 2022-08-20, Jan. 2022. [Online]. Available: https://www.justice.gov/usao-sdfl/

pr / ceo - major - online - cryptocurrency - exchange - company - indicted - defrauding-company-s.

[31] Reddit, *R/bitcoin - a popular coin tracking website tricked by coinjoin.* Accessed: 2022-08-26. [Online]. Available: https://www.reddit.com/r/Bitcoin/comments/bwat48/a_popular_coin_tracking_website_is_tricked_by/.

[32] WalletExplorer, *Coinjoinmess wallet*, Accessed: 2022-08-26. [Online]. Available: https://www.walletexplorer.com/wallet/CoinJoinMess.

[33] P. Wiki, *Global interpreter lock*, Accessed: 2022-08-29. [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock.

[34] Graphviz, *Graphviz, an open source graph visualization software*, Accessed: 2022-08-24. [Online]. Available: https://graphviz.org/.

[35] Django, *Django, an open source python framework*, Accessed: 2022-08-24. [Online]. Available: https://www.djangoproject.com/.

[36] Bumbu, *Bumbu/svg-pan-zoom: Javascript library that enables panning and zooming of an svg in an html document, with mouse events or custom javascript hooks*, Version: v3.6.1 - 21 Sep. 2019. [Online]. Available: https://github.com/bumbu/svg-pan-zoom.

[37] Eligrey, *Eligrey/filesaver.js: An html5 saveas() filesaver implementation*, Version: v2.0.4 - 6 Aug. 2020. [Online]. Available: https://github.com/eligrey/FileSaver.js/.

Appendix A

Decision Tree to Prune Transaction Inputs

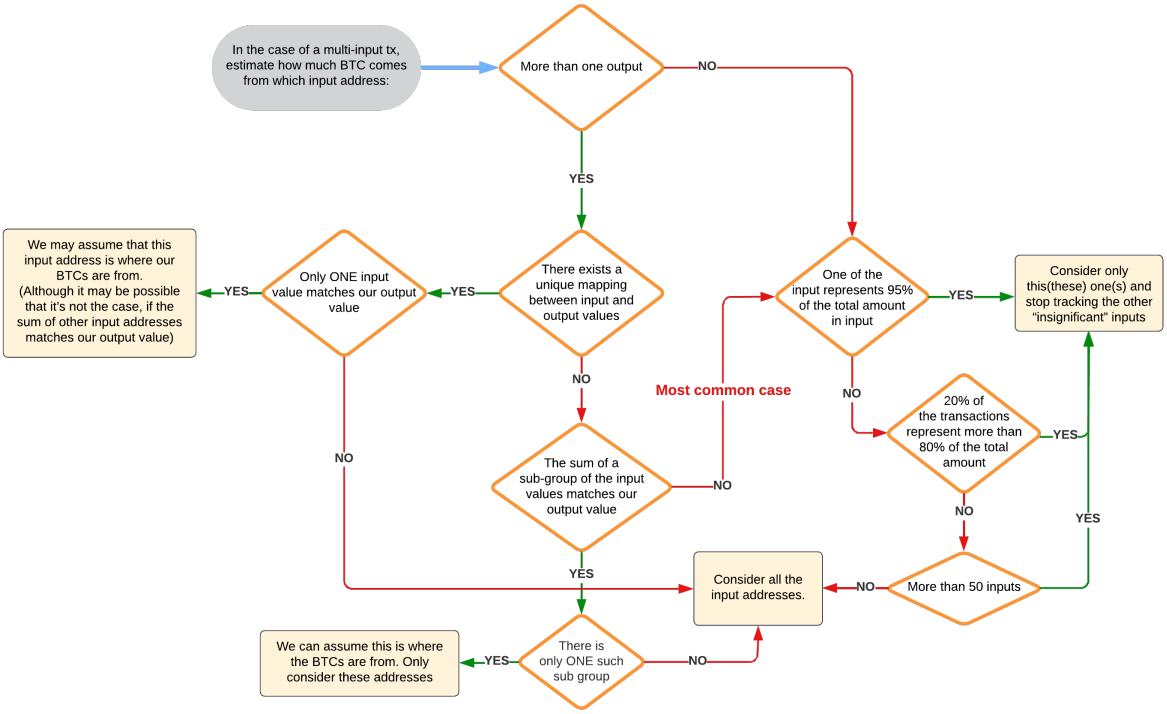


Figure A.1: Decision tree implemented to prune transaction inputs

Appendix B

Transaction graph of the address "115ZFznB..."

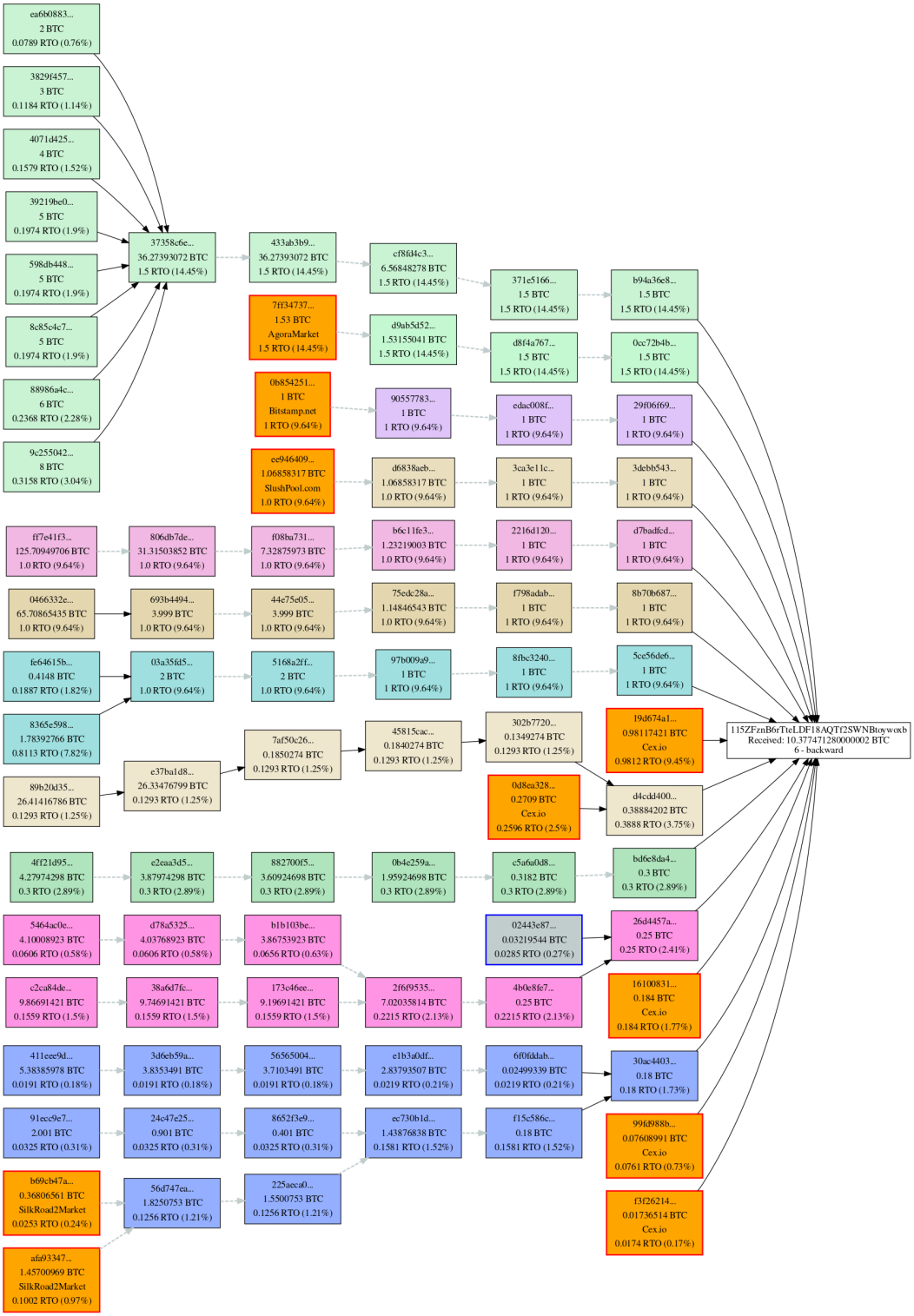


Figure B.1: Transaction graph generated for the address "115ZFznB..." thanks to implemented heuristics and analysis methods. 6 backward layers were browsed. Tagged transactions are represented in orange.

Appendix C

Web parsing result page

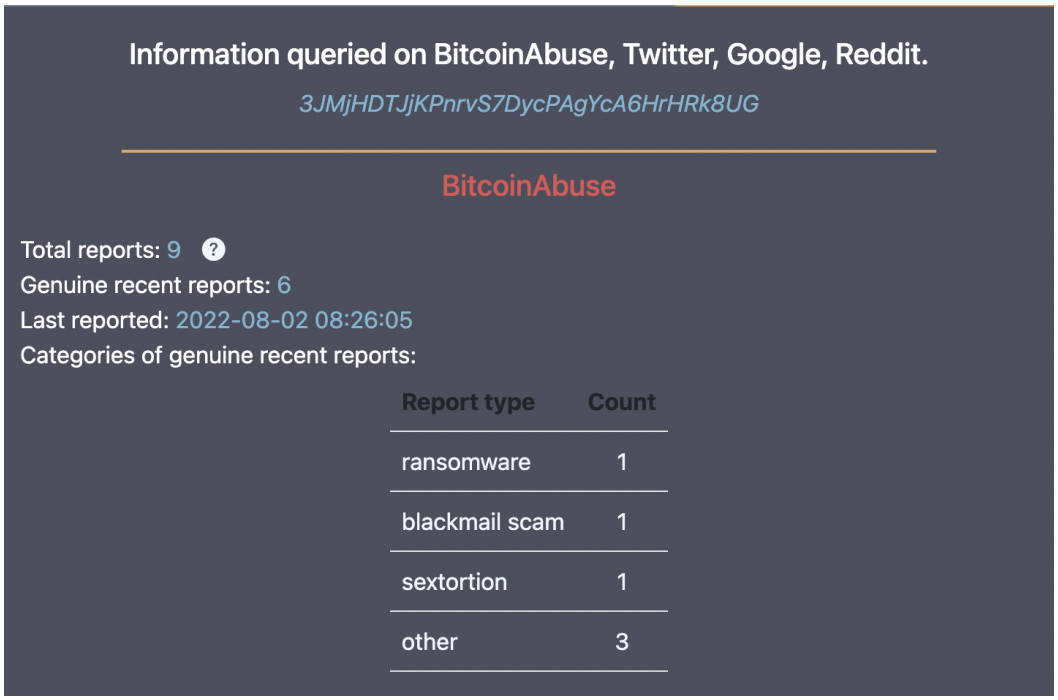


Figure C.1: First part of the off-chain scraping results for the romance scam address.

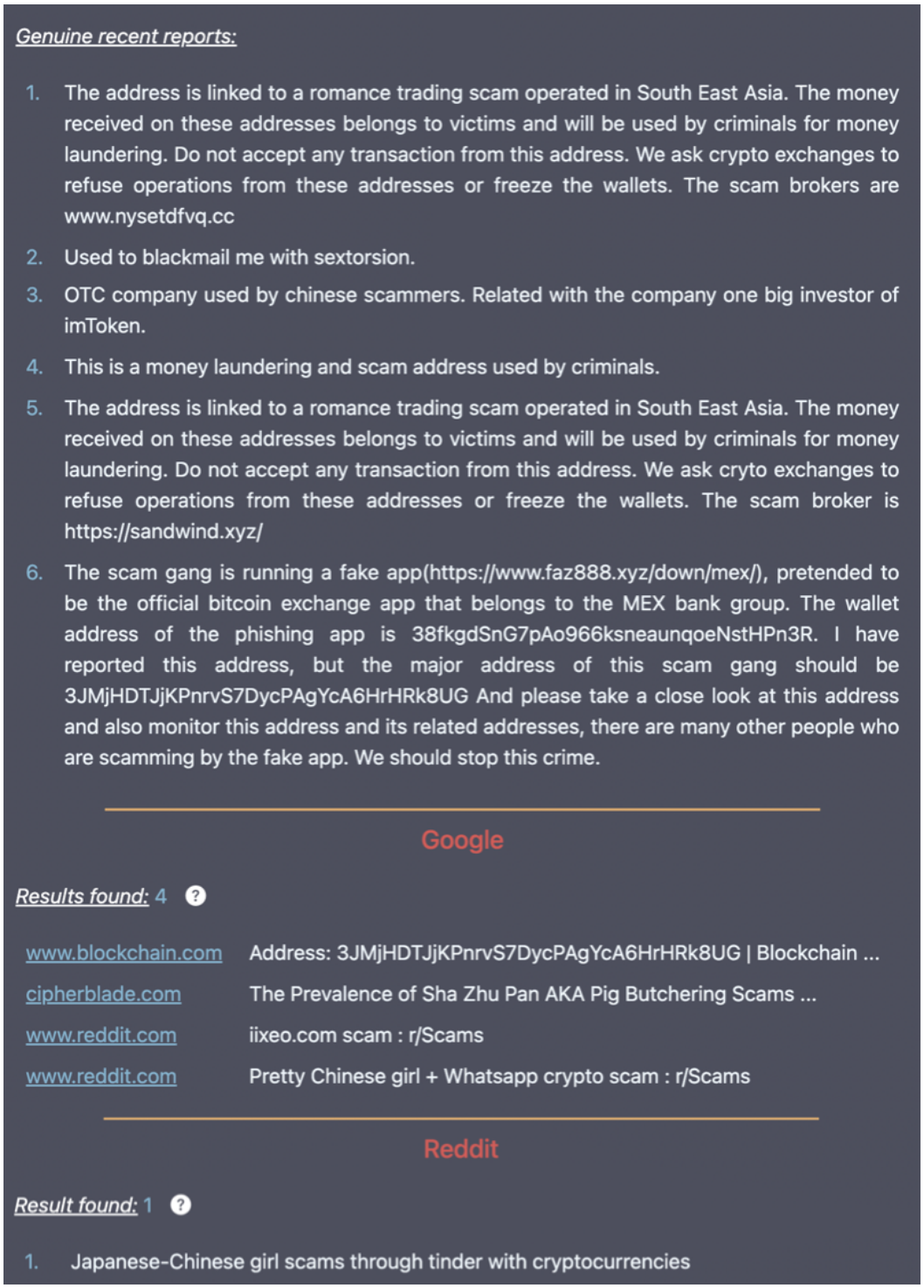
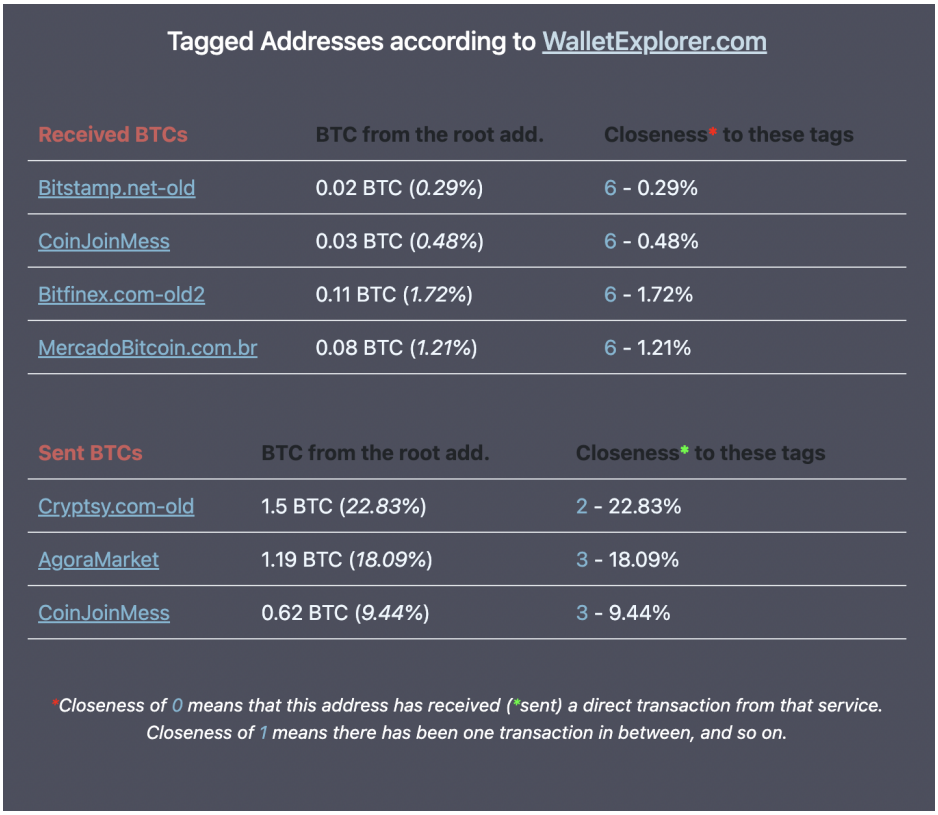


Figure C.2: Second part of the off-chain scraping results for the romance scam address.

Appendix D

CTS and statistics



Tagged Addresses according to WalletExplorer.com

Received BTCs	BTC from the root add.	Closeness* to these tags
Bitstamp.net-old	0.02 BTC (*0.29%*)	6 - 0.29%
CoinJoinMess	0.03 BTC (*0.48%*)	6 - 0.48%
Bitfinex.com-old2	0.11 BTC (*1.72%*)	6 - 1.72%
MercadoBitcoin.com.br	0.08 BTC (*1.21%*)	6 - 1.21%

Sent BTCs	BTC from the root add.	Closeness* to these tags
Cryptsy.com-old	1.5 BTC (*22.83%*)	2 - 22.83%
AgoraMarket	1.19 BTC (*18.09%*)	3 - 18.09%
CoinJoinMess	0.62 BTC (*9.44%*)	3 - 9.44%

*Closeness of 0 means that this address has received (*sent) a direct transaction from that service.
Closeness of 1 means there has been one transaction in between, and so on.

Figure D.1: First part of the CTS and statistics page.

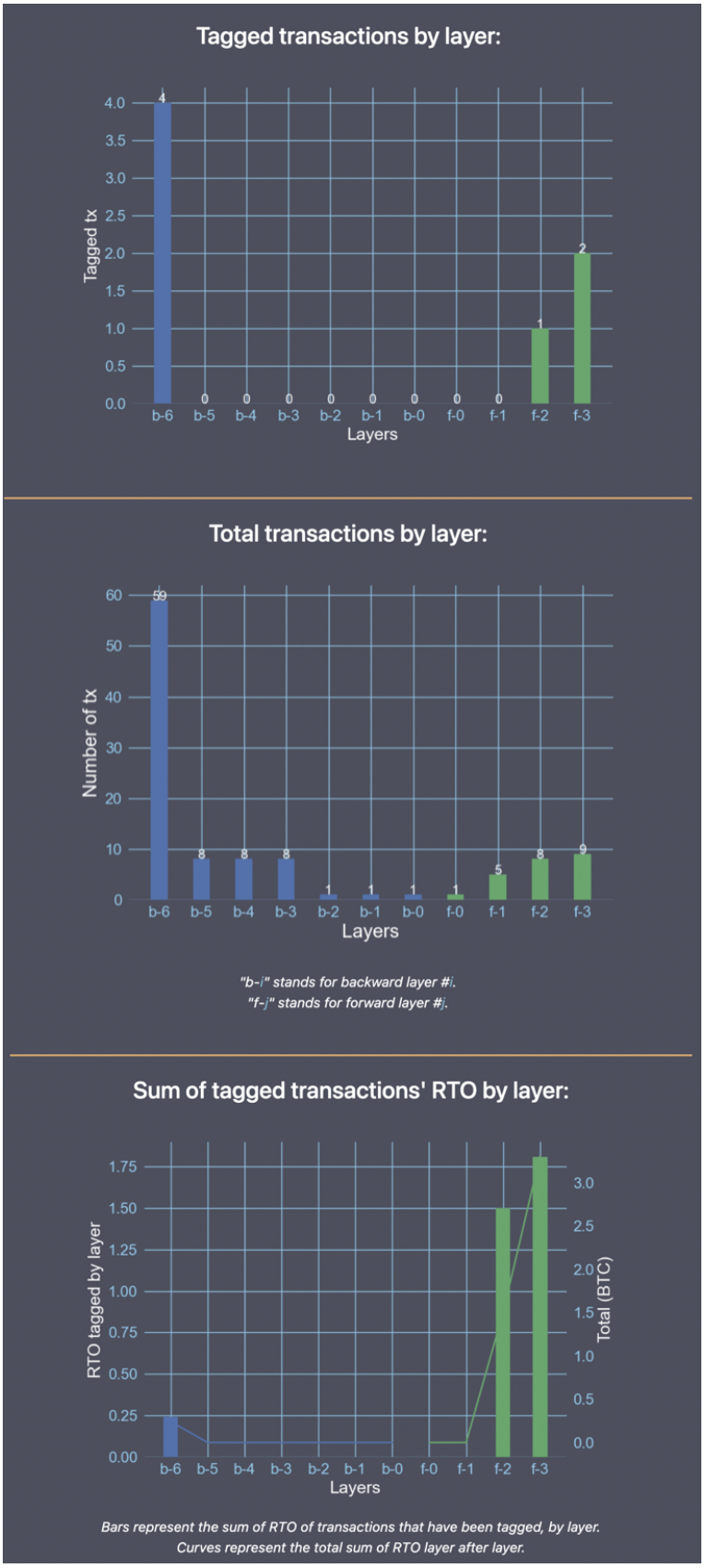


Figure D.2: Second part of the CTS and statistics page.