Master's Thesis : BitCoin Clustering: a CoinJoin Discarding Heuristic

Auteur : Jacquot, Vincent
Promoteur(s) : Donnet, Benoît
Faculté : Faculté des Sciences appliquées
Diplôme : Master en sciences informatiques, à finalité spécialisée en "computer systems security"
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/11236

UNIVERSITY OF LIÈGE

MASTER THESIS

Bitcoin Clustering
A CoinJoin discarding heuristic.

Author:
Vincent JACQUOT

Supervisor:
Pr. Benoit DONNET

A thesis submitted in fulfillment of the requirements
for the degree of Master in Computer Science
Professional focus on Computer Systems and Security

in the

University of Liège
Faculty of Applied Science

January 4, 2021

Abstract

The Bitcoin, which was an attempt to propose an alternative to the centralized currencies, quickly attracted all sorts of trafficking activities. The pseudo anonymity of the transactions and the absence of regulatory authority is a windfall for criminals.

Therefore, understanding and being able to extract exploitable information from the Bitcoin network is the key to deal with these illicit activities. There exist many techniques to extract exploitable information from the public transaction data. Unfortunately, many techniques have been deployed to counter them. This Thesis proposes a review of the application layer of the Bitcoin and of the work that has already been achieved in this field. Finally, a new clustering heuristic is proposed. In particular, an algorithm is presented to retrieve the public keys involved in a transaction. We will see how the graph theory and these keys can help to spot and discard the transactions that may result from an obfuscation technique.

Amongst other main contributions, it is shown that i) the popular transaction patterns vary extremely over time, ii) at least 3% of the transactions involve several entities. Finally, iii) most of the clusters are very ephemeral, iv) and the wealth is by far non-uniformly distributed amongst them.

Acknowledgements

Throughout the writing of this thesis, I have received a great deal of support and assistance.

Firstly, I would like to thank my supervisor, Professor Benoit Donnet, whose expertise and feedback were invaluable in the dissertation of this thesis. I would also express my gratitude to him and all the teaching staff of the University of Liège. Their dedication to their work allowed me to learn so much during these 5 years.

I would like to acknowledge a PhD student, Sami Ben Mariem, for his advises. They really helped me to determine what should be my objectives and to avoid dispersing.

In addition, I would like to thank my family and my friends for their moral support during my whole studies.

Contents

List of Figures

List of Tables

List of Abbreviations

BTC	**B**i**TC**oin
UTXO	**U**nspent **T**ransaction **O**utput
Tx	**T**ransaction
P2PK	**P**ay **T**o **P**ublic **K**ey
P2PKH	**P**ay **T**o **P**ublic **K**ey **H**ash
P2MS	**P**ay **T**o **M**ulit **S**ignatures
P2SH	**P**ay **T**o **S**cript **H**ash
P2WPKH	**P**ay **T**o **W**itness **P**ublic **K**ey **H**ash
P2WSH	**P**ay **T**o **W**itness **S**cript **H**ash

Chapter 1

Introduction

Eleven years ago, the Bitcoin [1] was born and has kept on growing from there. It was the first attempt to propose a decentralized cryptocurrency. It does not require any institution to work and allows to exchange currency between peers very easily. For these reasons, it is very popular for international transactions. The absence of regulation also attracted all kinds of illicit activities. It quickly became the most well known cryptocurrencies with the general public, and most likely due to its phenomenal variation in price. For example, a recent survey found that 8% of Americans have invested in cryptocurrency and of that 7.95%, 5.15% have invested in Bitcoin, see Fig. 1.1a.[2] The Bitcoin is also the cryptocurrency in which the American spends the most in average, see Fig. 1.1b.

Despite being a pseudo anonymous currency, all the transaction data are public. Therefore, very soon in its existence, many researchers [3][4][5][6][7][8][9] tried to extract exploitable information from the public data, such as the wealth distribution or flux amongst the users or their identity. Consequently, organizations and persons that had an interest in keeping the anonymity, innovated in order to counter these techniques.

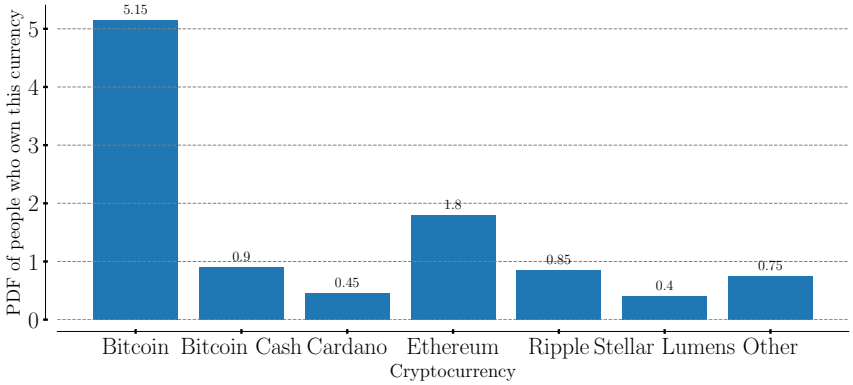
One of the first and most effective technique proposed to link some transactions to real world entities is the multi-input heuristic.[4] The principle is really simple. If several sources of bitcoins are used together in a transaction, the sources are probably controlled by the same entity. This assumption is known as the common-input-ownership assumption.

To defeat this heuristic, many techniques and services allow to mix the funds of several users in a transaction. [10][11][12] One of these techniques is the Coinjoin anonymisation strategy.[13][14] It consists in grouping several independent transactions into only one.

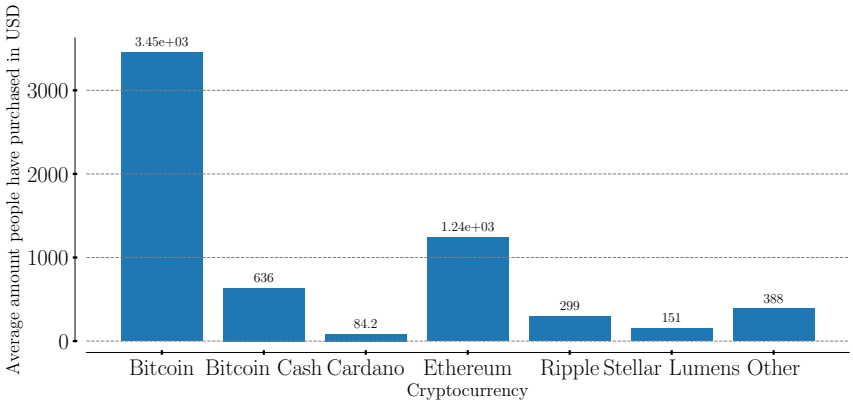
While the main objective of the Bitcoin was to propose an alternative to the centralized currencies, it became quickly popular amongst the criminals and the tax evaders. Being able to extract information from the public data is crucial in criminal investigations. In October 2019, the Financial Crimes Enforcement Network (FinCEN) reported a fine of 60 million dollars against Helix, a Bitcoin mixer. Larry Dean, its founder, was accused of money laundering. [15] Back in the days, Silk Road was the most notorious online criminal marketplace, before being shut by the FBI. Buyers and sellers conducted all transactions with Bitcoin. [16]

1.1 Contributions

The clustering topic has been discussed in many papers. The first heuristics were proposed in [4]. Because these basic heuristics generally fail these days, most of the new refined approaches, such as [5], discard a large portion of the public data that may cause issues.



(A) Proportion of people who own this currency



(B) Average amount people have purchased in USD

FIGURE 1.1: Cryptocurrencies's Popularity

This Master thesis addresses the possibility to exploit more efficiently the transactions and still being able to discard insecure Bitcoin usage patterns. A new refined multi-input heuristic is proposed.

In particular, we'll see that the graph theory can be used to formalize the concept of transaction and categorize these latter. This classification is used to distinguish the unsafe patterns and discard them.

In addition, a technique to extract the public keys (i.e. the identities of the entities) from the transactions is described. We also show how these keys can improve the effectiveness of our heuristic.

Finally, this new heuristic is applied to the data generated between the beginning of the Bitcoin and the 19th of October 2019. The analysis of the clusters revealed that most of them are very ephemeral and the wealth is largely non uniformly spread amongst them.

1.2 Thesis Outline

This Thesis starts with an outline of the Bitcoin and its mechanics in Chapter 2. Specifically, the application layer is studied in depth. The Chapter 3 is a tour of the previous work achieved in this field. The existing heuristics and techniques are described.

While being public, the data still require a lot of pre-processing in order to be exploited. Thus, the Chapter 4 is entirely dedicated to the data collection methodology. The first part focuses on the tools that were employed to collect and process the data (Sec. 4.1). The second part proposes a first insight on the data (Sec. 4.2).

Based on the knowledge in the previous chapters, the refined multi-input heuristic is defined in the Chapter 5. The motivations are first discussed in Sec. 5.1 based on the discussion of the other methods (Chapter 3) and the first inspection of the data (Sec. 4.2). The graph theory approach is formalized in the Section 5.2. Finally, the last Section 5.3 is dedicated to the extraction of the public keys involved in a transaction.

Then, the Chapter 6 presents the results of the new heuristic on the transactions up to the 19^{th} of October 2019.

Finally, the Chapter 7 sums up the observations and results obtained throughout this thesis and proposes some future works.

Chapter 2

The mechanics of Bitcoin

This chapter introduces the basic notions required for the reminder of this Master thesis. In particular, we focus on the main principles of the Blockchain (Sec. 2.1), the basics of cryptography (Sec. 2.2) and an introduction to the application layer (Sec. 2.3). This latter section will introduce the concepts of transactions and scripts that are developed in depth later (Sec. 2.4 and Sec. 2.5).

Finally, this chapter ends with an introduction of the mechanisms that exists to increase anonymity: the centralized mixing services (Sec. 2.6.1), the decentralized mixing services (Sec. 2.6.2) and Zerocoin and Zerocash (Sec. 2.6.3).

2.1 The Bitcoin Blockchain

Satoshi, the author of the first publication about Bitcoin, defines the Bitcoin coin as a as a chain of digital signatures.[1] To transfer a coin, the owner of a coin can sign the hash of the previous transaction and the public key of the next owner. A payee can verify the signatures to verify the chain of ownership.

A decentralized solution However, as it is this solution does not prevent the double-spend of the coin, i.e. a coin being used twice by his owner. A common solution is to introduce a trusted central authority, but the entire system depends on the system in this case.

To design a system without a trusted party, the transactions must be publicly broadcasted and the participants must agree on a single history of the order in which the transactions were received. The history of all transactions is called the ledger.

The Blockchain To prevent tampering of the data and keep the order, the solution works by hashing a block containing transactions and the hash of the previous block. This forms what is called the Blockchain (see Fig. 2.1).

As such, any modification in a block will result in an invalid hash in the next block. The system also implements a proof-of-work to prevent an adversary to modify a block and recompute accordingly all the hashes.

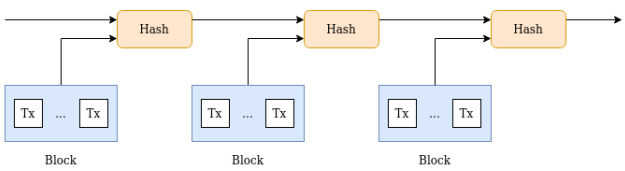


FIGURE 2.1: The Blockchain

Proof-of-Work The proof-of-work involves a value called *nonce* that when hashed with the rest of the data, the hash begins with a number of zero bits. [1]

The valid Blockchain is defined as the one that required the largest amount of work. This design requires that a majority of nodes are honest. If not, some attack such the 51% attack is possible.[17] These nodes are called miners.

To compensate the growth of the miners' computing power and their interest, the difficulty is adjusted to produce 6 blocks per hour. For example, if the number of blocks growths too fast, the difficulty is increased.

Incentive The decentralized architecture also implies there is no central authority to create new coins. They are created to reward the miners when a block is mined. The block reward is halved every 210,000 blocks (or roughly every 4 years). Initially, the reward was 50 BTC.

The mining reward is not the only incentive. The sum of money that flows out of a transaction can be lower than the sum of money that flows in. In this case, the difference is added to the mining reward of the block containing the transaction. This incentive may help to adjust to the inflation of the electricity cost and encourage nodes to stay honest.

2.2 A Public Key Architecture

The Bitcoin network[1] is based on a decentralized public key architecture. It uses public keys as identities and the private keys to perform signatures and prove the ownership. In particular, it uses the Elliptic Curve Digital Signature Algorithm (ECDSA) and the curve "secp256k1". [18, Chapter 1.4] The details on the functioning of this scheme are beyond the scope of this work, but it is interesting to remind the following features.

- A ECDSA private key is 256 bits long.
- A uncompressed ECDSA public key is 512 bits long.
- A compressed ECDSA public key is 257 bits long.
- A message to sign is 256 bits long.
- A ECDSA signature is 512 bits long.

ECDSA can only sign messages that are 256 bits long. This is not a problem, it only requires to hash every piece of data that requires to be signed.

The signature also relies on generating a random value which may lead to vulnerability and attacks if the source of randomness is bad. [19]

2.3 An overview of the Bitcoin Application Layer

The Bitcoin architecture[1] is very different from the banking architecture. This section presents a quick recap of the very basics to know before digging into the details.

Contrary to the traditional banking system, the money is not tied up to one account, but rather on what is called an output. The money on an output cannot be spent until a proof of ownership is presented. This is where the concept of inputs comes. An input is basically the proof that you own a particular output.

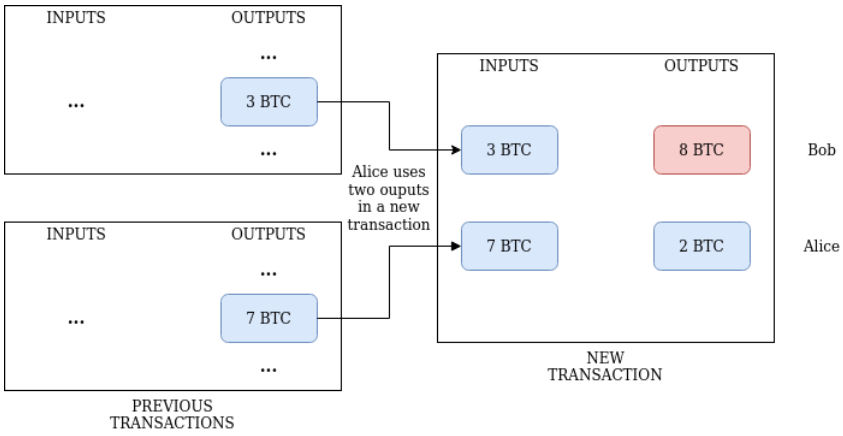


FIGURE 2.2: High level representation of a transaction

A transaction is a collection of inputs and outputs. Let's assume that Alice owns respectively 3 BTC and 7 BTC on two different outputs that lie in different transactions (see Figure 2.2). Now, she wants to transfer 8 BTC to Bob. She will create a new transaction using as inputs the two outputs she owns. It will contain two outputs; one contains 8 BTC and is spendable only by Bob and the second will collect the change (2 BTC) and belongs to Alice.

Once an output is redeemed, it is totally consumed by the transaction. This is why Alice has to create a change output.

A transaction must satisfy the non-exhaustive list of rules to be valid:

1. The sum of the inputs values must be greater or equal to the sum of outputs values.
2. Each input must provide a valid proof of ownership of the output it spends.
3. Each output that is redeemed by an input must exist and has not been already spent.

The transactions are broadcasted on the network and published into blocks by special nodes called miners. These latter have the responsibility to check the validity of the transactions.

To check the condition 3, the miners need to parse the Blockchain backward to check if another transaction already spends the output.

The decentralized architecture also implies there is no central authority to create new coins. They are created to reward the miners when a block is mined. The block reward is halved every 210,000 blocks (or roughly every 4 years). Initially, the reward was 50 BTC.

The complexity of the mining is adjusted such that in average 6 blocks are mined per hour.

2.4 Transaction

At a low level, a transaction is composed of inputs, outputs and some other fields and whose format is presented in the Table 2.1 in the Annex 2.1. This format is well-known thanks to the reference client [20] that was initially maintained by Satoshi which is the pseudonym used by the person or the group of persons that developed the Bitcoin. [1]

TABLE 2.1: Transaction syntax[21]

field		**size in bytes**	**description**
version		4	Transaction data format version
flag		0 or 2	If present, always 0001, and indicates the presence of witness data
input count		variable	Number of inputs
inputs	txid	32	The hash of the referenced transaction.
	vout	4	The index of the specific output in the transaction.
	script size	variable	The length of the signature script
	scriptsig	script size	A script that unlocks the input
	sequence	4	Transaction version as defined by the sender. Intended for "replacement"of transactions when information is updated before inclusion into a block.
output count		variable	Number of outputs
outputs	value	8	Transaction Value
	script size	variable	Length of the pk_script
	scriptpubkey	script size	A script that locks the output.
witnesses		variable	A list of witnesses, one for each input; omitted if flag is omitted above.
locktime		4	The block number or timestamp at which this transaction is unlocked.

If you take a look at the format of an input, it is nothing more than a reference to an output and a script that unlocks this output and allows the user to spend it. There is also a 'version' field that indicates the transaction version of the sender and a 'sequence' that do not matter in our case.

An output is composed of a value and a script that locks it. The value is given in Satoshis (1 Satoshi = $\frac{1}{100000000}$bitcoin).

Initially, each output was bound to one public key or one public key hash. The address was at that time defined as the hash of this public key. As the Bitcoin evolved, some variations appeared: it is possible now to associate several public keys to an output or the hash of a script. Thus, the concept of address associated with an output does not necessarily represent the same information. Consequently, the next section will cover in depth the Bitcoin scripting language.

It is worth to explain the "lock_time" field which offers a powerful mechanism used in some applications. This field is used to lock a transaction until a given time and prevent it to be included in the ledger. There are three distinct cases:

- $lock\_time = 0$: The transaction is unlocked.
- $lock\_time < 500000000$: It represents the block number at which this transaction is unlocked.
- $lock\_time \geq 500000000$: It represents the Unix time at which this transaction is unlocked.

See section 2.5.7 for an example.

2.5 Script

The script language used in the Bitcoin network, soberly named 'Script', is stack-based. Every instruction is executed exactly once in a linear manner. By design, this language is thus not Turing-complete which is by purpose. For example, there is no loop instruction. To check the validity of a spending in the chain, these scripts have to be executed by the miners. It was too risky to run arbitrarily powerful functions submitted by users (e.g. infinite loops). Each instruction is encoded into 1 byte meaning that there are only 256 possible different opcodes. [18, Chapter 3.2]

To check if an input correctly redeems an output, the concatenation of the input script, called 'scriptsig', and the output script, called 'scriptpubkey', is executed. The output is considered as spent by the input if the execution of the resulting script does not raise errors and the top stack item is True (non-zero) when the script exits.

While the large majority of scriptpubkey requires a proof of ownership in order to spend money, it is not mandatory. For example, the following locking script:

$$(ADD\ 8\ EQUAL)$$

is perfectly valid as well as the corresponding unlocking script:

$$(6\ 2).$$

The execution is illustrated here Fig.2.3

As such, extracting information from any custom script would require a full semantic analyser. Fortunately, in the first years of the Bitcoin history, standard transactions were defined. The reference client was implemented such that it does not relay non-standard transactions. [22] However, such transactions can still be found

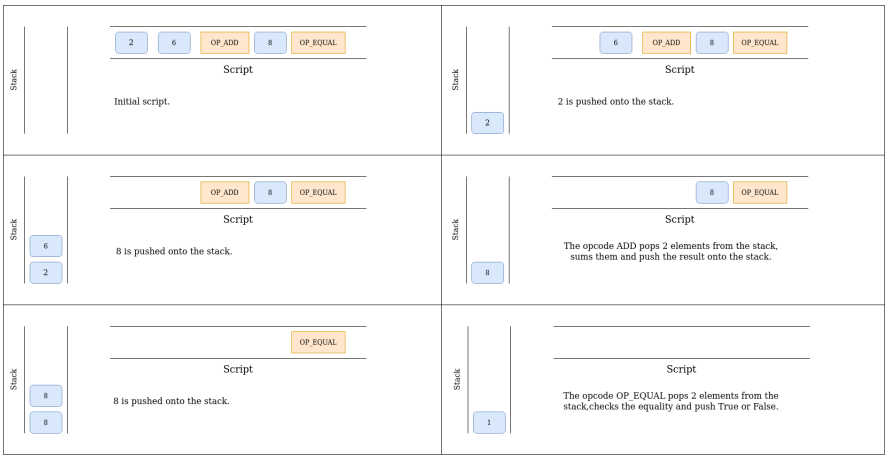


FIGURE 2.3: Execution of the script.

FIGURE 2.4: Template of a P2PK script.

in the Blockchain since nothing prevents some specific implementations to broadcast them and some miners to include them in the block. There are two main motivations to restrict the set of scripts that could be found in the Blockchain.

1. Some scripts might cause harm to the network.

2. Some scripts might make future upgrades harder.

An example of harmful script is presented on the following blog [23]. Concerning the second motivation, new operators could be designed with the OP_NOP1-OP_NOP10 opcodes if a softfork was carefully designed. [24] [25]

On the 13th of August 2019, the non-standard outputs represented only the 0,02% of the total. [3] The main reason being that it is in the interest of the major actors in the Bitcoin industry to take care of the network. Furthermore, miners try to maximize their profit by including transactions that offer the best fee/kB ratio. Interpreting a non standard script is computationally expensive compared to simply include standard scripts in a block.

The list of standard scripts is discussed in subsequent sub-sections.

2.5.1 Pay to Public Key (P2PK)

This script is the first being used in the main network.[1]

The signature contained in the unlocking script and the public key contained in the locking script are pushed onto the stack.

Finally, the opcode CHECK_SIG is executed. It checks the validity of the signature. In particular, the signature and the public key must match and the signature is bound to the transaction data it is in. [26]

The address can be easily extracted from this kind of script since the method is standardized and well known.[27] The method is described in the Annex A.1.

[1] see tx_id $4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b$ in genesis block $000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f$

FIGURE 2.5: Template of a NULL DATA script

FIGURE 2.6: Template of a P2PKH script.

2.5.2 NULL DATA

This script has been added in Bitcoin 0.9.0.[28] As soon as the OP_RETURN is executed, the script is marked as invalid and the rest of the script is discarded.

This functionality originates from the fact that there is no way to prevent users to store data in a transaction. For example, a user could create a null-value output with a P2PK locking script and insert some arbitrary data in place of the public key. Such outputs would pollute the UTXO set. This kind of scripts allows people to include arbitrary data inside transactions while it has the property of being provably unspendable. Thus, such scripts can be discarded from the UTXO set.

Such a practice is discouraged by the community in any case since there are more convenient ways to store data outside the Blockchain.[29]

Some examples of such scripts can be found here[2] and here[3].

2.5.3 Pay to Public Key Hash (P2PKH)

This script is the most common on the Bitcoin network. The input script contains a signature and the associated public key while the locking script contains the hash of the public key.

The reason of its popularity is that it more convenient to send money to an address than to a public key. The main reasons are:

1. The address associated with a public key is composed of a checksum making safer the transactions (see Section A.1).

2. The hash of a public key (20 B) is shorter than the full public key (64B).

An example of a valid script execution is provided in Fig. 2.7 The signature and the public key are pushed onto the stack. Then, a copy of the public key is pushed onto the stack. This copy is hash160'ed, meaning that it is hashed with SHA-256 and the result is hashed with RIPEMD-160 and pushed onto the stack. The public key hash is pushed onto the stack and compared with the previous result. Finally, the opcode CHECK_SIG is executed as for the P2PK script.

The method to obtain the address from a public key hash is presented in the Annex A.1.

2.5.4 Pay to Multi Sig (P2MS)

The multi signatures scripts have been standardized in October 2011 in the BIP 11. [30] The motivation was to support several applications that need more than one signature, such secured wallets, escrow transactions, and other use cases. Some of these are presented in the Section 2.5.7.

[2] txid: 8bae12b5f4c088d940733dcd1455efc6a3a69cf9340e17a981286d3778615684
[3] txid: 6dfb16dd580698242bcfd8e433d557ed8c642272a368894de27292a8844a4e75

FIGURE 2.7: Execution of a valid P2PKH script.

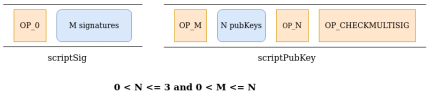


FIGURE 2.8: Template of a P2MS script

The opcode CHECK_MULTISIG checks the validity of m signatures against a pool of n public keys ($n \geq m$). If all signatures are valid, a 1 is returned, otherwise a 0 is returned.

Note due to a bug in the implementation of this operator, an OP_0 is required in the unlocking script because the opcode pops one too many items off the execution stack. This bug has never been fixed so far because it would require the update of all nodes.

This script also contains special operators to indicate the number of signatures (M) and public keys (N). Currently, the maximum value for N is 3. [30]

The method to obtain the address from a public key hash is presented in the Annex A.1.

2.5.5 Pay to Script Hash (P2SH)

The BIP 16 defined a new standard script in January 2012. The benefit is that any script arbitrarily complicated is hashed into a 20-byte hash that is short enough to be scanned from a QR code or easily copied and pasted.[31] Furthermore, it has the benefit to reduce the size of the UTXO DB.

In practice, it also means that the redeemer is responsible to provide the redeem script.

Some rules to check the validity of such script are also added:

FIGURE 2.9: Template of a P2SH script

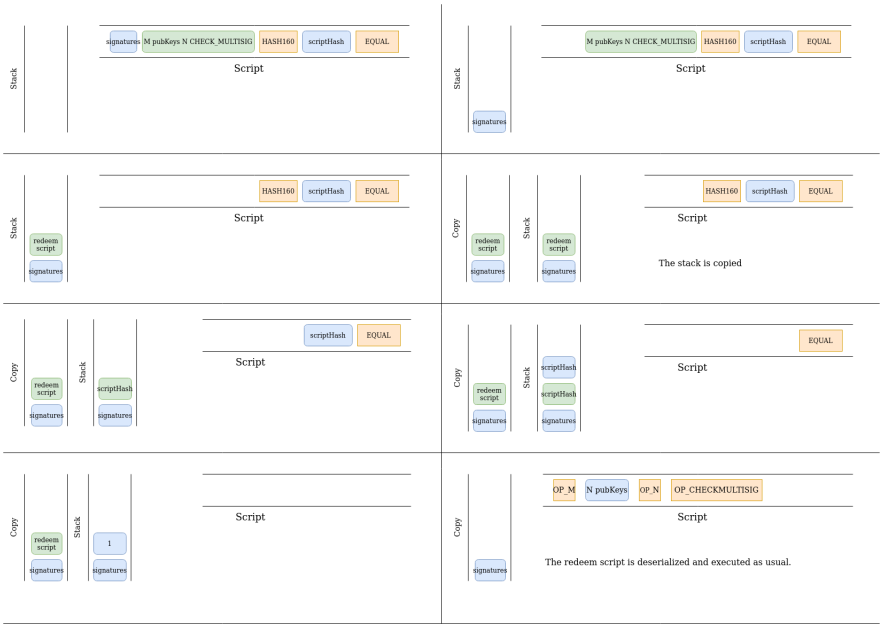


FIGURE 2.10: Execution of a P2SH script containing a P2MS script. (Note: the dummy 0 value due to the CHECKMULTISIG bug has been omitted for sake of simplicity.

1. Validation fails if there are any operations other than "push data" operations in the scriptSig.

2. Normal validation is done: an initial stack is created from the signatures and serialized script. The hash of the script is computed and validation fails immediately if it does not match the hash in the outpoint.

3. serialized script is popped off the initial stack, and the transaction is validated again using the popped stack and the deserialized script as the scriptPubKey.

In addition, the serialized script (or redeem script) has be itself standard. An example of P2SH script whose redeem script is a P2MS script is presented at Fig.2.10

The address is computed on the hash of the redeem script and the method is presented in the Annex A.1.

2.5.6 Pay to Witness Public Key Hash (P2WPKH) and Pay to Witness Script Hash (P2WSH)

In December 2015, two new standard scripts are added: the P2WPKH and the P2WSH scripts. [32] The main benefit of using Segregated Witness (or Segwit) is to fix one the malleability issues well known in Bitcoin.[33] Third-party malleability is the possibility for some third-party to change the transaction without modifying its semantic.

In details, if one or more signers of the transaction revise their signatures, then the transaction remains valid and pays the same amounts to the same addresses, but the txid changes completely because it incorporates the signatures. [34]

Segwit prevent scriptSig malleability by moving and segregating the malleable parts of the transaction into the transaction witness such that this witness does not account in the calculation of the transaction id.

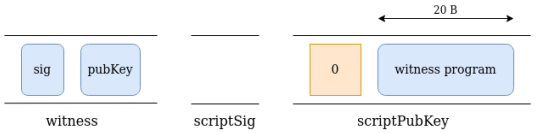


FIGURE 2.11: Template of a P2WPKH script

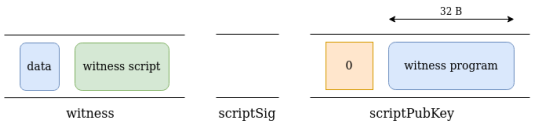


FIGURE 2.12: Template of a P2WSH script

Every witness program starts with a 0 which stands for the version of the witness program. Then the size of the witness program indicates its type: 20 bytes for P2WPKH and 32 bytes for P2WSH.

In the case of the P2WPKH, the program is the hash160 of a public key. The public key from the witness is thus hash160'ed and compared to it. Finally, the node verifies that the signature is a valid one for the public key and the transaction.

In the case of a P2WSH, the witness script" is popped off, hashed with SHA256, compared against the 32-byte-hash in scriptPubKey, and deserialized. The deserialized script is then executed with the remaining data from the witness.

Furthermore, it is interesting to highlight that these modifications have been deployed through a soft fork, older nodes can continue to operate without any modification. Non-upgraded nodes will just interpret such scripts as True and spendable by anyone. Indeed, such scripts are composed of a 0 followed by a non-zero value which is interpreted as True.

Finally, it must be noticed that the segwit addresses have their own format as described in BIP 173.[35] The motivations for this change are the following:

- Base58 needs a lot of space in QR codes, as it cannot use the alphanumeric mode.
- The mixed case in base58 makes it inconvenient to reliably write down, type on mobile keyboards, or read out loud.
- The double SHA256 checksum is slow and has no error-detection guarantees.
- Most of the research on error-detecting codes only applies to character-set sizes that are a prime power, which 58 is not.
- Base58 decoding is complicate and relatively slow.

This format is described in depth in the Annex A.2.

2.5.7 Applications of scripts

While having a full scripting language may seem more complicated than a single architecture using only public keys, it also offers unique opportunities to design applications. This section will present some applications implementable with the Bitcoin script language. [18, Chapter 3.3]

Escrow transactions Suppose that Alice wants to purchase a product from Bob but she doesn't want to send any money before getting the product. In the other side, Bob won't send the product until he receives the money. A simple way to solve this is to use a P2MS script with the assistance of a third-party arbitrator Judy.

Alice will send the money to an output locked by a script requiring 2 valid signatures and include the public keys of Alice, Bob and Judy. At some point in time, the transaction will be in the Blockchain and Bob will judge as safe enough to send the goods.

In the normal situation, Alice will receive the goods and both she and Bob will sign the input spending the money held in escrow and send it to Bob. In the case, where some conflict emerges between Alice and Bob, Judy will decide who merits to receive the money. She will sign along with one of them the input to redeem this money.

Efficient micro-payments Let's assume that Bob offers a service that requires micro payments, i.e. x Satoshis per unit of time. The naive solution to create a transaction per payment unit is costly considered the sum of the fees. Ideally, we would like to have a unique transaction that sums up all the micro payments.

An elegant solution is the following: Alice will send some money that corresponds to the maximum she can spend with the service to an escrow address locked by a MULTISIG script. This script will require the signatures of both Alice and Bob. As soon as this transaction appears in the Blockchain with enough confirmations (typically 6), Bob will consider as safe to initiate the service.

At each unit of time t, Alice will sign a new transaction sending $t \cdot x$ to Bob and sending the remaining to her. So far, Bob hasn't need to sign these transactions because they won't all end up in the ledger. When Alice is done with the service, she notifies Bob, who signs the last transaction and broadcasts it.

In addition, to provide the certainty that Bob won't keep the money in escrow forever if he refuses to sign the last transaction, a transaction sending back the money to Alice after a certain time is signed by Alice and Bob at the beginning. This script uses the mechanism implemented by the 'lock_time' field in the Bitcoin on the BTC network. transactions.

2.6 Anonymity and Bitcoin

At the very beginning, some users were concerned about anonymity because the transactions are public and everyone can try to extract and infer information from them. This section presents the main methods that were developed to increase the anonymity of the Bitcoin users.

2.6.1 Centralized mixing services

A dedicated mixing service, also called mixer, promises to mix your funds with the funds of other users to obfuscate some linkability. The principle is quite simple, you send your money to an address provided by the service and an address on which you want to get back your money minus the service fees. Your money is then mixed with the money of other users in a same transaction, see Figure 2.13.

To effectively mix money and provide unlinkability, a series of guidelines should be observed. [10]

Use a series of mixers. A user should use a series of mixers such that there is no need to trust one in particular. One of the risks is that some of them could keep some records. The illustration of such series is presented at Figure 2.14. It is enough to

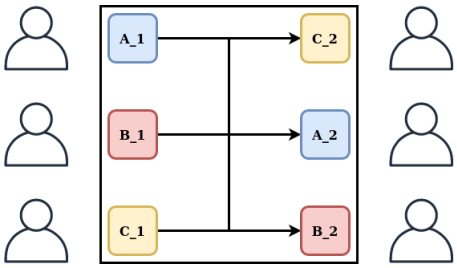


FIGURE 2.13: Mixing transaction

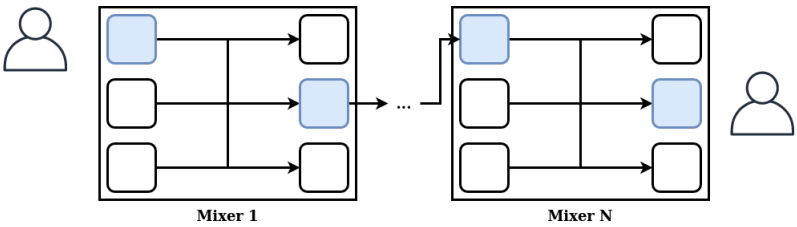


FIGURE 2.14: A series of mixers.

have one trustful mixer in the path to guarantee the safety. Such design is quite common and is similar to the design of the Tor system which uses a series of 3 routers.

Uniform transactions. Obviously, the transaction's inputs should be uniform within a same transaction and across the different mixers. The complexity is to find the right uniform value size, called chunk size. A chunk size too large would discourage "small" users while a small one would require too many splits from the "big" users.

Client side should be automated. The possibility of side-channels attacks exist with mixing. The possibility of attacks analogous to packet counting attack in communication mixes and intersection attack in the mixing literature exists. Some measures can be taken to counter them, but are impractical to be used by a user itself. Theses measures should be included into a wallet software implementation.

Fees should be all-or-nothing. Mixers expect to be paid and usually take a part of the money in each input, but it introduces a new problem. If they proceed in this way, the output chunks can no longer be in the standard size. Then, if Alice tries to split and merges these chunks to get uniform chunks, it would link these chunks.

The solution is to retain an entire chunk as a fee with a probability p. However, such solution requires the trust of the user and the reliability of the mixers.

2.6.2 Decentralized mixing services

A major disadvantage of centralized services is that you have to trust them to not keep information about you. Some theft cases have also been reported in the past. [7] In 2013, two topics have been posted on the official bitcoin forum by the user gmaxwell. [13] [14] These posts described what is called now CoinJoin transactions.

The basic principle is the following:

1. Several users that are interested to mix funds contact each other.

2. They exchange input/output addresses.
3. They build together a transaction.
4. They sign after verifying their output is present.
5. The transaction is broadcast.

2.6.3 Zerocoin and Zerocash

While the mixing methods essentially increase privacy at the application layer of Bitcoin, Zerocoin[11] and its successor Zerocash[12] incorporate anonymity at the protocol level. The anonymity properties come with cryptographic guarantees.

However, these solutions suffer from compatibility issues with the vanilla Bitcoin. Zerocoin authors pretend it could be added to the Bitcoin protocol through a soft fork as it was the case for the BIP16.[31] In the case of Zerocash, a soft fork is not even possible.

Chapter 3

Clustering Overview

This chapter is dedicated to the clustering techniques. We will start with the definition of clustering. The clustering techniques try to defeat the unlinkability property of the BTC currency, i.e. to be able to detect and link the activity of a same entity. In particular, they try to link the different addresses belonging to a same entity.

While being pseudo-anonymous, the Bitcoin transaction data are public. In particular, the information contained in the scripts allows to extract the addresses used in these transactions. There is no ground-truth in this work field, so the techniques are essentially heuristics. It is also worth to mention that an entity signing transactions is not necessarily the owner. Many services, such as online wallets or Bitcoin exchanges, work as intermediary for the users and take in charge the technical details. There is no guarantee on the way these services are implemented and some addresses could contain the money of several users. [18, Chapter 4]

This chapter starts with an introduction to the multi-input heuristic (Sec. 3.1), the change address heuristic (Sec. 3.2) and a method to identify users by transaction behavior (Sec. 3.3).

3.1 Multi-input Transactions

One of the first heuristic (in 2013), [4] consists in grouping the input addresses used in a transaction. The motivation is that shared spending should infer shared control. If a bunch of addresses are spent in common, i.e. in a same transaction, we can generally assume that they are controlled by the same entity. By recursively applying this technique to other transactions, one can find clusters to merge together. An example is provided at Fig. 3.1.

However, this heuristic can fail in presence of mixing transactions. If the inputs of a transaction belong to several entities, the clusters associated to these entities will be merged.

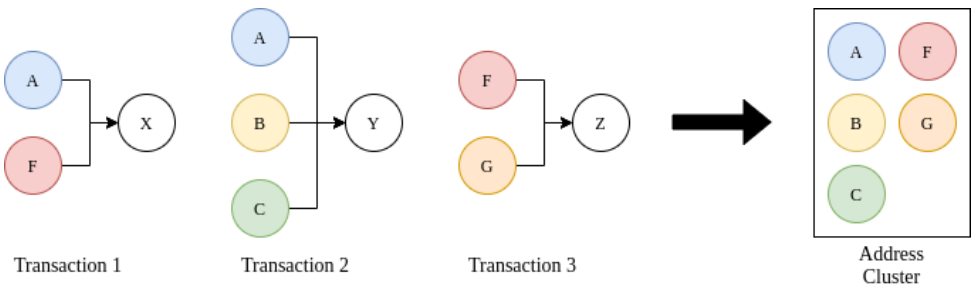


FIGURE 3.1: Example of multi-input heuristic applied on 3 transactions. The cluster obtained is composed of 5 addresses.

In 2017, Ermilov et al. [5], proposed a refined heuristic to get rid of these transactions by discarding transactions with more than one output. Such restriction reduces the false positives at the detriment of the amount of exploitable data.

3.2 Address Change

This heuristic aims at improving the utility of transactions data. Assuming that Alice owns 2 addresses with respectively 4 and 6 BTC and wants to send 7 BTC to Bob. To recover her change, a second output will be used to redirect 3 BTC to an address she owns. Such an address is called a change address. In particular, if this address is also in the inputs, it is called a self-change address. In 2013, Meiklejohn et al. [4] presented a idiom of use that was proved as being true for most Bitcoin clients implementations. Thanks to this idiom of use, an heuristic has been proposed to determine which address collected the change. The formal definition of such address is the following:

Definition 3.2.1 (One-time change address)**.** A public key pk is a one-time change address for a transaction t if the following conditions are met:

1. $d^{+}_{addr}(pk) = 1$; i.e., this is the first appearance of pk.
2. The transaction t is not a coin generation.
3. There is no $pk' \in outputs(t)$ such that $pk' \in inputs(t)$; i.e., there is no self-change address.
4. There is no $pk' \in outputs(t)$ such that $pk' \neq pk$ but $d^{+}_{addr}(pk') = 1$; i.e., for all the outputs in the transaction, condition 1 is met for only pk.

However, this heuristic produces false positive that may yield to link addresses which are not controlled by the same entity.

Some variations exist in the literature. For example, another definition has been proposed by Ermilov et al.[5] It consists in the addition of new constraints in order to eliminate some of the false linkages. Only the transactions with exactly 2 outputs and a number of inputs different of 2 will be considered. Furthermore, the decimal value of the change output must have more than 4 digits after the dot, while the other address must have appeared previously in the Blockchain but never as a change address.

3.3 Identifying Bitcoin users by transaction behavior

In a conference paper of 2015, V. Monaco [6] claims that Bitcoin transaction behavior is largely non random and somewhat nonlinear. It appears that it is possible to extract the behavioural patterns over time from some users. Monaco proposed to use this method to assist the other existing heuristics.

3.4 Defeating Mixing Services

As the number of services that proposes to obfuscate the transaction history increases, some techniques to detect these latter have been developed.

Möser et al. [7] were amongst the first to study these services. They used reverse-engineering methods to understand the mode of operation and discussed a series of

potential attacks. Most of them have already been discussed in a previous section, see Sec. 2.6.1.

In 2016, Yanovich et al, [9] focused their effort on a new model based on the graph theory to detect CoinJoin transactions.

In 2018, Younggee et al, developed a method that could identify the relationships between the input and output addresses of the Helix mixing service with a 99.14% accuracy rate. Helix was at that time one of the most popular mixing service.

Chapter 4

Data Collection Methodology

This chapter presents the main tools used during this thesis (Sec. 4.1). Bitcoin Core, the reference Bitcoin client, was the source of data, Sec. 4.1.1. This data required some work before being exploitable, the process is presented at Sec. 4.1.3. We have developed several tools for data processing, such as a raw block (Sec. 4.1.2) and a script parser (Sec. 4.1.4). A big picture of the data flow and its transformation is proposed at Fig. 4.1. Finally, a simple GUI to study some particular transactions or the effect of my methods on them has been created, Sec. 4.1.5. All this code is available on gitlab.uliege.be.

As the heuristics often work on a subset of the transaction data, it was important to provide a basic analysis of some metrics at the end of this chapter (Sec. 4.2).

For example, the most basic multi-input heuristic works on transaction with at least 2 inputs. Therefore, the distribution of the number of inputs is interesting to study, (see Sec. 4.2.1). An analysis of the script types share is also proposed (Sec. 4.2.2).

4.1 Data Collection

4.1.1 Bitcoin Core

The first step was to obtain the transactions from the Blockchain.

At first, we worked on the data we get from several web APIs (blockchain.info, sochain.com). While being a very convenient way to get a few blocks, it was not possible to download the full Blockchain (approx 280 GB) with these latter.

The solution we chose was to run a full node on my computer with the c++ reference bitcoin client.[1]. By this mean, we obtained the binary data of the whole Blockchain in a relatively small amount of time (approximately 24 hours).

This client provides a convenient CLI that allows one to retrieve easily the blocks and their transactions from the Blockchain. For example, the daemon is able to provide the blocks from height 400000 to height 450000 in JSON format in about an hour (332 GB).

4.1.2 A Raw Block Parser

While being a very convenient format to work with, the json format also consumes more space on disk. The whole Blockchain takes already 300 GB when it is in binary format, see Fig. 4.1. Considering the 436 GB of transactions discussed in the previous section, it was not possible for me to keep the whole Blockchain in json format.

In order to parse these binary blocks, a small program has been written. It handles all the Bips, so it handles correctly the witness data.

[1] https://bitcoin.org/en/full-node

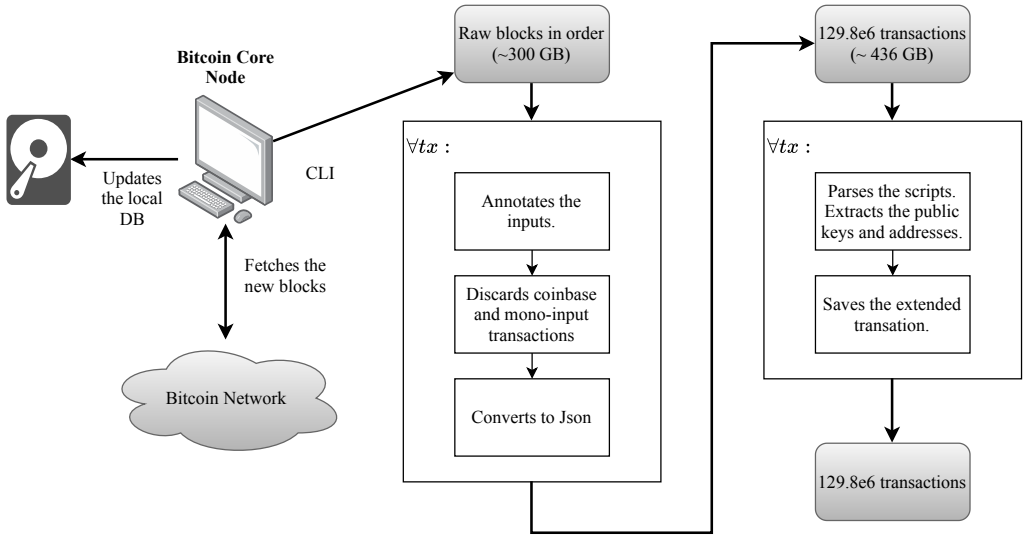


FIGURE 4.1: Data flow.

The CLI of Bitcoin Core was used to test the correctness of the implementation.

4.1.3 Annotating the inputs

As it, the inputs of these transactions were not exploitable. Indeed, the inputs only contain the txid and the index of the output they redeem and the script to unlock the output while a multi-input heuristic requires some addresses knowledge. We developed a python program to retrieve the outputs that the inputs redeem and annotates them with the locking script and the value, see Fig. 4.1.

It works in several passes. The first one parses the Blockchain linearly and every transaction such that:

- The outputs are stored in an UTXO set which is loaded into RAM.
- The inputs retrieve the information from this set, are annotated and remove the output they redeem from the UTXO set.

Unfortunately, the size of UTXO set rapidly growths as illustrated in the Figure 4.2[2] and it was not feasible to keep in memory all the UTXOs. The UTXO set is limited to a few millions of outputs. When the set is full, the oldest UTXOs are evicted from memory. It leads into 2 cases:

- A transaction finds all the outputs it redeems in the set and thus is 'complete' and is saved on disk.
- A transaction didn't all the outputs it redeems in the set and thus is 'incomplete' and is saved on a different file.

This first pass allowed to annotate completely approximately 80% of all transactions in one pass over the Blockchain.

The next steps collected the missing outputs keys, transaction id + index of output, from the incomplete transactions and parsed the Blockchain to retrieve them. Since the outputs keys are light, 32 bytes (the transaction id) + 1 byte (the index),

[2] https://charts.bitcoin.com/bch/chart/utxo-set-size#5ma4

FIGURE 4.2: Evolution of the UTXO set.

it was possible to retrieve and dump to disk all the missing UTXOs in only several parsing of the Blockchain. Finally, these outputs were used to complete the transactions.

To reduce the disk usage, we only kept the transactions that were relevant for my clustering technique, that is to say the transactions with at least 2 inputs. We ended up with 436 GB of transactions in json format, see Fig. 4.1.

4.1.4 A Script Parser

While being very convenient, it seems that Bitcoin Core does correctly parse and provide the addresses of P2PK scripts. Moreover, some old P2MS scripts are also badly parsed. [3] None of Bitcoin Core or the online APIs (blockchain.info, sochain.com) we tested were able to provide the addresses for this script.

Some routines were developed in order to extract the addresses from a script. Furthermore, these routines also allow to retrieve the public keys that relate to these addresses, see Fig. 4.1.

4.1.5 An investigation GUI

As we were in the beginning, we developed a basic GUI (Fig. 4.3) with Qt and networkxx to display some transactions. The motivation was to ease the manual investigation of some transactions. It is a convenient way to display a list of transactions and their simplified form as described in Section 5.2.1. A very basic filtering system allows to select some transactions.

4.2 A First Look on Data

The Blockchain has been analyzed from the genesis block (block 0), up to the 600,000th block. This block has been published on the 19th of October 2019.

This section starts with the analysis of the number of inputs and outputs per transaction (Sec. 4.2.1). Then, the script types proportion is presented (Sec. 4.2.2.

The methodology employed to get the average metric values is to split the Blockchain into chunks of blocks and average the values in each of this chunk. The first 200,000 blocks were split into chunks of 10,000 blocks and the remaining into chunks of 1000

[3] E.g. txid: 78b28d3c2324da8c2f01840021addbcabb68f7ce1d4da870cabe5e9df6afe63d

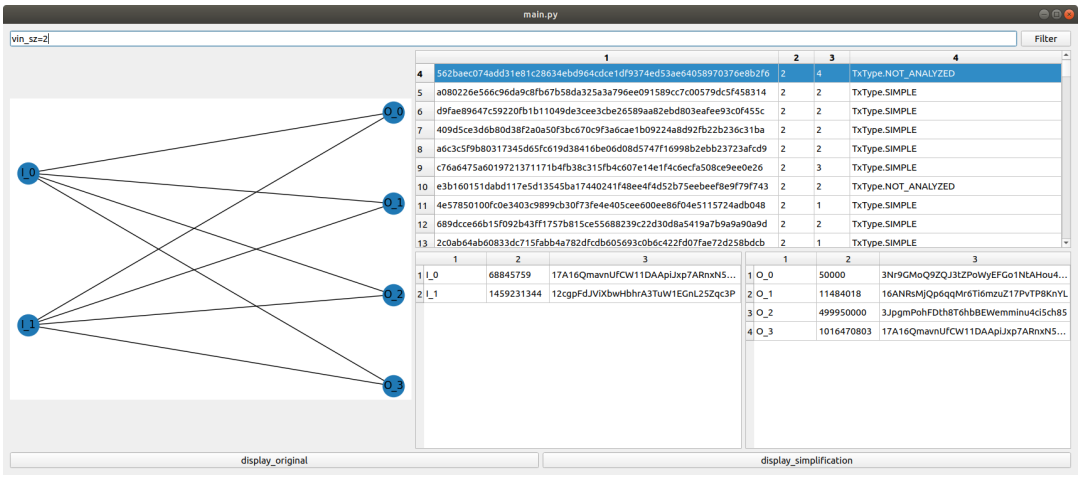


FIGURE 4.3: My investigation GUI. A representation of the selected transaction is displayed at the left.

blocks. The reason for these different chunk size is that the first 200,000 blocks account only for approximately 1% of the data (in size) and splitting them into chunk of 1000 blocks would result in very small chunks of data.

4.2.1 Inputs and Outputs Number

There is no limitation on the number of outputs and inputs that compose a transaction. The maximum number of inputs, resp. of outputs, found in a transaction was 20000[4], resp. 13107[5].

Over the 262 million possibilities, 56273 of them have been used at least once in the Blockchain. A plot of the 11 most frequent pairs is presented at Fig. 4.4.

It appears that the majority, 58.5 %, of transactions have one input and two outputs. This pair can be associated with the regular use case, when someone buys a service or a product and collects the change in a second output.

The transactions with exactly 2 outputs and 2 inputs represent 11.1% percent, while the ones composed with 1 input and 1 output stand for 7.9% of the share.

Data exploitable by the heuristics The proportion of transactions which are concerned with the vanilla multi-input heuristic (Sec. 3.1) (i.e. more than 1 input) is 27.8 %, while the exploitable data by the refined heuristic (i.e. more than 1 input and exactly 1 output) represent only 3.77%. The proportion of transactions which is potentially exploitable (i.e more than 1 output) by the change address heuristic is 88.3 %. The refined version that considers only transactions with exactly 2 outputs and a number of inputs which differs from 2 works potentially applies to 67.1% of the data.

Most of the transactions are composed of a little number of inputs and outputs. For example, only 3.5 % of them are composed with more than 10 inputs or 10 outputs.

To conclude this section, the evolution of the average number of inputs, resp. outputs, is presented at Fig. 4.5b, resp. Fig. 4.5a. The mean is heavily influenced by

[4] txid:52539a56b1eb890504b775171923430f0355eb836a57134ba598170a2f8980c1 blockhash:00000000000000000b8d631639b0dd7ea223f2efc46fc9ab0992f7d612e29c25

[5] txid:dd9f6bbf80ab36b722ca95d9...8e0d4cf0e7d2e28a7a91ab3 blockhash:000000000000000002637cfa48bdc97203a812484ae92def02cb586f6e7d3c03

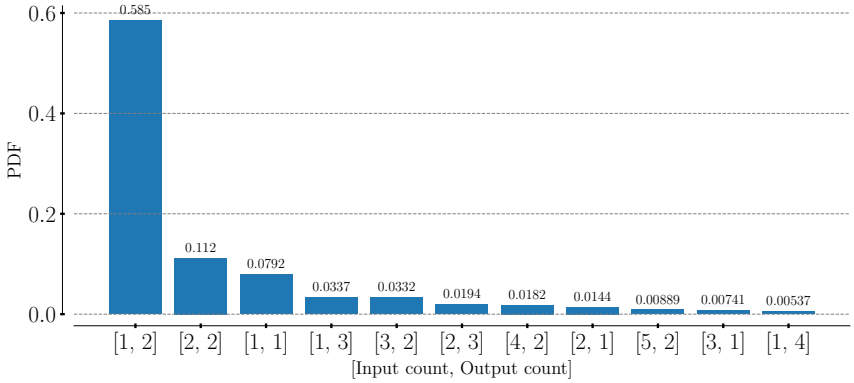
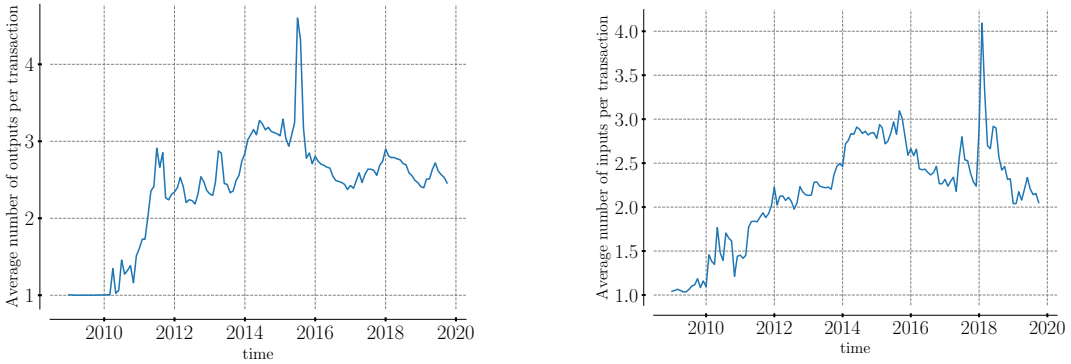


FIGURE 4.4: Top 11 input count-output count pairs.



(A) Average number of outputs over the time

(B) Average number of inputs over the time

FIGURE 4.5: Input and output average count over the time

extreme values. A few transactions with a very large number of ouputs/inputs are possibly responsible for the peaks visible.

4.2.2 Script Types

It can be useful to study the share between the different script types because it is possible to use this information in the context of clustering. For example, it could be relevant to discard the P2MS scripts or P2SH scripts which contain P2MS, because these scripts are sometimes used in custom use cases 2.5.7 that may not fit some heuristics.

A representation of the script share over the time is proposed at Fig. 4.6.

While being the first script in use at the beginning, the P2PK popularity has decreased for the benefit of the P2PKH scripts. Nowadays, its use is negligible, see Table 4.1. Currently, the P2PKH scripts are still the most used, but their use decreases over time. The P2SH and P2WPKH are replacing them. Some of the P2SH are witness scripts nested into P2SH [32]. It is likely that the witness scripts are more widely used than represented in the graph.

TABLE 4.1: Script types share

Date (from : to)	P2PK	P2PKH	P2MS	P2SH	P2WPKH	P2WSH	NULL
2019-10-12: 2019-10-19	0.002	0.417	0.0	0.336	0.137	0.007	0.101

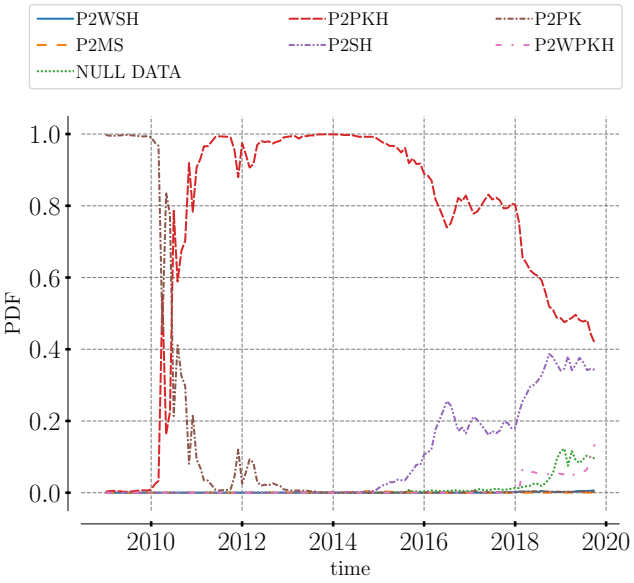


FIGURE 4.6: Script types share over the time

Chapter 5

A CoinJoin Discarding Heuristic

This chapter is dedicated to the techniques that have been employed to cluster the addresses. It starts with the motivations (Sec. 5.1). Then, a new technique based on the multi input heuristic (Sec. 5.2) is introduced. In particular, this method attempts to detect and discard CoinJoin transactions. Finally, an improvement based on a better exploitation of the scripts is proposed 5.3.

5.1 Motivation

The clustering problem in Bitcoin is less about having enough data rather than limiting false positives. A false positive means grouping addresses of different entities into a same cluster. Even a small number of small positives can yield to collapse a large number of addresses into large "super-clusters" that are not actually controlled by a single entity. This problem is well known by the researchers since 2013. [4]

Recently, some refined heuristics aim at reducing this phenomenon without totally eliminating it.[5]

We decided to focus my work on the multi input heuristic for the following reasons. First, it has been proven to be very powerful in practice [4], [5]. Finally, it is also well motivated in theory. Harrigan and Fretter enumerated and analyzed the primary reasons behind the effectiveness. [36]

On the other hand, we were really cautious about using the change address heuristic which is mainly based on an idiom of use.

Using the vanilla multi input heuristic, that exists since 2013, [4] seemed a bad idea considering the methods that exist and can easily defeat it, see Sec. 2.6. The upgraded version of Ermilov et al., increases the reliability of the results, but at the cost of a reduced amount of exploitable data (3.77% of the transactions, Sec. 4.2.1).

We decided to focus on the decentralized mixing techniques because the centralized services do not represent the main thread. In addition to taking a fee, these services are frequently accused of theft and have a bad reputation. It is also likely to see their usage diminish over time.

In many countries, the operators have some legal constraints. For example, one of their frequent responsibility is to deploy an efficient anti-laundry system and report suspicious activities to the authorities. They have sometimes to keep a certain level of information over the users.

The non-respect of these constraints can lead to legal actions against them. In 2019, FinCEN (short for Financial Crimes Enforcement Network) reported a fine of $ 60 million against Helix, one of the most popular mixing services. [15] It can encourage users that seek privacy to use decentralized services.

Furthermore, these centralized mixers involve some technical constraints to provide efficient obfuscation, such as uniform chunk, etc (see Sec. 2.6.1), which are not always really convenient for the users.

Finally, most of the recent solutions to provide anonymity rely on CoinJoin transactions: Wasabi Wallet[1], Samourai Wallet[2], JoinMarket[3], ...

5.2 A Graph Theory Approach

In a white paper published in 2016 [9], Yanovich et al., proposed a method:

- To determine whether a certain bitcoin transaction is a result of a shared send operation.
- To uncover value flows within shared send transactions.

The next subsection is a summary of their work. It starts with a few elementary definitions (Sec. 5.2.1) and a simplification that is applied to the transaction data. Then, the problem of detecting the CoinJoin transactions is formalized and described as a graph problem (Sec. 5.2.2). A final section is dedicated to study the reliability of the results (Sec. 5.2.3).

5.2.1 Definitions

Before digging into the problem, let us remind a few general definitions and introduce the new concepts that are going to be used.

Definition 5.2.1 (Transaction)**.** A transaction can be defined as a triple $t = (A; B; c)$, consisting of:

1. The finite multiset of transaction inputs A, where each input $(a_i; A_i) \in A$ is an ordered pair of the address A_i and the value of the input $a_i > 0$.
2. The finite multiset of transaction outputs B, where each output $(b_i; B_i) \in B$ is an ordered pair of the address B_i and the value of the output $b_i > 0$.
3. The transaction fee c:

$$c \equiv \sum_{(a_i,\cdot)\in A} a_i - \sum_{(b_i,\cdot)\in B} b_i \geq 0$$

Definition 5.2.2 (Sum() & Addr())**.** For any input or output multiset A:

1. Addr(A) denotes the multiset of addresses in A.

$$Addr(A) = \{A : (\cdot, A) \in A\}$$

2. Sum(A) denotes the sum of the values in A.

$$Sum(A) = \sum_{(a,\cdot)\in(A)} a$$

[1] https://www.wasabiwallet.io/
[2] https://samouraiwallet.com/
[3] https://joinmarket.me/

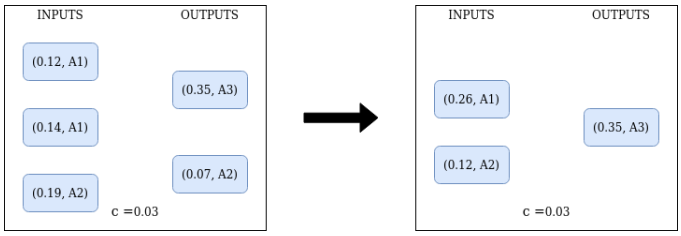


FIGURE 5.1: Example of simplified transaction

Simplification The same address can be used several times in a given transaction, in the inputs as well as in the outputs. Thus, the following simplification can be applied to every transaction without any loss of information:

1. Join all inputs belonging to the same address and all outputs belonging to the same address.
2. Put an address to the set of inputs if the value of the corresponding input is greater that the value of the output associated with the address (assuming the value is zero for addresses not encountered in inputs or outputs); otherwise regard that address as an output.
3. Recalculate the value for each new input and output straightforwardly.

An example of simplified transaction if proposed at Fig. 5.1 and allows us to define the concept of shared send transaction. In the rest of this chapter, we are going to assume the transaction data are simplified.

Definition 5.2.3 (Shared Send Transaction)**.** A transaction $t = (A, B, c)$ is a shared send transaction iif its simplified version $t' = (A', B', c)$ is such that $|A'| > 1$ and $|B'| > 1$.

Note: a shared send transaction is not necessarily the result of mixing action, but any CoinJoin transaction is necessarily a shared send transaction.

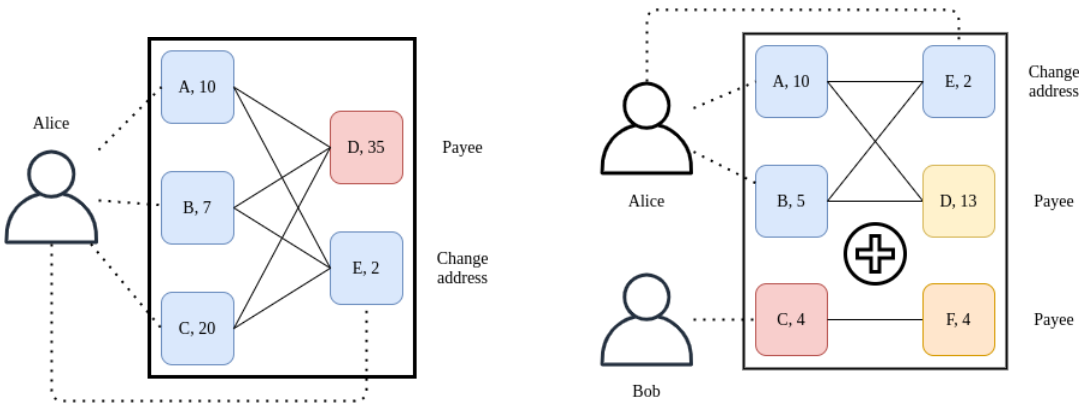
Finally, let us recall the definition of a partition.

Definition 5.2.4 (Partition)**.** A partition of a set X is a collection of sets $\{X_k\}_{k=1}^{K}$ such that the sets X_k are pairwise disjoint and their union equals X. We denote a partition as $X = \cup_{k=1}^{K} X_k$.

5.2.2 A Graph Theory Problem

We can represent the value flows inside a transaction $t = (A; B; c)$ by a bipartite graph $G(t)$ in which the inputs and the outputs of the transaction are the two disjoint and independent sets. This transaction graph is constructed as follows:

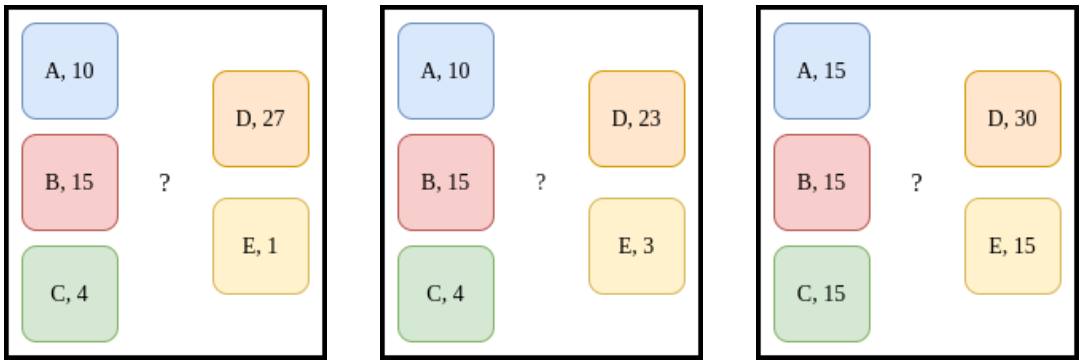
- The vertices represent the inputs and the outputs. Each node is labeled with a positive value which is associated with the value.
- An edge exists between an input and an output if the entity which owns the input has transferred it to the output. The output may belong to the same entity or a different one.



(A) Graph of a non-mixing transaction.

(B) Graph of a Coin Join transaction.

FIGURE 5.2: Graphs of a non-mixing and a CoinJoin transaction



(A) A transaction whose graph is necessarily complete.

(B) A transaction whose graph admits a partition.

(C) A transaction whose graph admits several partitions.

FIGURE 5.3: Graphs of a transaction based on the public data.

More formally,

$$G(t) = (V, E);\ V = A \cup B;\ E \subseteq A \times B$$

The transaction data is public, so we have all the information related to the vertices: the values and the addresses. The edges represent the unknown part of the problem. Recovering the information about the edges is the key to identify mixing transactions or follow some coins in criminal investigations.

In particular, we can notice that the graph is always complete for a non-mixing transaction (Fig. 5.2a). In the other side, it is incomplete for a mixing transaction where there is no financial arrangement between the participants (Fig. 5.2b).

If based on the public data of a given transaction, we can retrieve the edges, we can determine if the graph is complete or if it is separable into independent sub-graphs. For example, the graph at Fig. 5.3a is necessarily complete. There is no other acceptable configuration. On the other hand, the graph at Fig. 5.3b could be complete, but it also admits a valid partition. The transaction can be subdivided into two valid transactions. The first one would be composed of the inputs A and B and the output D. The second one would spend the input C to the output E. Finally, the last graph (Fig. 5.3c) admits several valid partitions.

So, it means that the problem of detecting some mixing transactions is equivalent to the problem of partitioning a graph into sub-graphs. To do so, we need a few more

definitions.

Definition 5.2.5 (Connectivity)**.** Consider a transaction $t = (A; B; c)$. A pair of non-empty sets (A', B'), where $A' \subset A, B' \subset B$, is called *connectable* in t iff the following condition holds:

$$Sum(B') + c \geq Sum(A') \geq Sum(B')$$

Definition 5.2.6 (Acceptable Transaction Partition)**.** Consider a transaction $t = (A, B, c)$. A pair of K-element partitions ($K > 1$) of input and output sets, i.e. $A = \cup_{k=1}^{K} A_k, B = \cup_{k=1}^{K} B_k$ is called *acceptable* iff each pair (A_k, B_k) is connectable. The notation $P = \{(A_1, B_1), ..., (A_k, B_k)\}$ denotes an acceptable partition for t.

Definition 5.2.7 (Minimal Connectable Pair)**.** A connectable pair $(A; B)$ is called minimal if the sets in it cannot be represented as $A = A_1 \cup A_2$,$B = B_1 \cup B_2$ such that pairs $(A_1; B_1)$ and $(A_2; B_2)$ are connectable.

Definition 5.2.8 (Minimal Acceptable Partition)**.** An acceptable partition $P = \{(A_k, B_k)\}_{k=1}^{K}$ is called *minimal* if it cannot be further subdivided into an acceptable partition, i.e., (A_k, B_k) is a minimal connectable pair for each $k = 1; ...; K$.

The problem is equivalent to find all the minimal partitions of a transaction. However, this problem is NP-Complete. [9] We need to restrict the space of research by excluding transactions with a large number of inputs and outputs. We decided to consider the transactions whose the number of inputs and outputs is lower or equal than 10. This seems a good compromise since it concerns 96.5 % of the transactions (Sec. 4.2).

Depending on the results, a shared send can be classified with the following new type.

Definition 5.2.9 (New Transaction Types)**.** The following categorization is proposed for the transactions:

- Simple transaction: A transaction that does not result from the application of the CoinJoin algorithm, see Fig. 5.3a. The remaining types of transactions are referred as tangled transactions.
- Separable transaction: A tangled transaction that admits a unique separation into sub-transactions, see Fig. 5.3b.
- Ambiguous transaction: A tangled transaction that admits several separations into sub-transactions, see Fig. 5.3c.
- Intractable transaction: A transaction whose category cannot be determined because of the computational limitations.

Small Outputs Removal A last improvement has been proposed by Yanovich et al. to classify mixing transactions as *shared send*. Consider a transaction which mixes the funds of several entities and take a mixing fee of 1 %, see Fig. 5.4. If we discard the small outputs, i.e. the ones that represent less than a given percentage of the total value of the inputs, the graph is no longer complete. It can either admit several partitions, either one partition in case of non uniform chunks. The following table gives

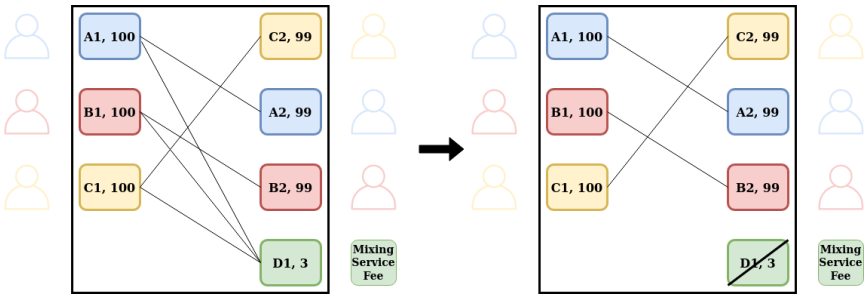


FIGURE 5.4: Mixing transaction performed with mixing fee of 1 % whose small output is discarded.

the mixing fee of several popular services, Table. 5.1 BitMix[4] and Bitcoin Mixer[5] collects a percentage of the value. Samurai Wallet[6] collect a fixed fee which depends on the chunk size. CryptoMixer[7] takes a percentage plus a fixed fee of 0.0005 BTC. Finally, WasabiWallet[8] charges 0.003% times the anonymity set, i.e. the number of users that take part in the mixing operation.

If the outputs whose value is smaller than 5% are discarded, we can be sure that these mixing transactions will be detected as shared transactions. The cost is that some of the non mixing transactions will be discarded as well.

	Mixing fee
CryptoMixer	start at 0.5% + 0.0005 BTC
Wasabi Wallet	0.003% * anonymity set
Samurai Wallet	from 0.0005 BTC to 0.025 BTC
Bitcoin Mixer	from 2 to 5 %
BitMix	from 0.4 to 4%

TABLE 5.1: Mixing fee of several services

As we have seen, it is likely than non mixing transactions are simple transactions, while the shared transactions are more suspicious. The heuristic is applied to the simple and the non-shared transactions to reduce the risk of clustering several addresses belonging to different entities.

5.2.3 Reliability

As described in the previous subsection, the method aims at filtering the mixing transactions. A false positive denotes a tangled transaction which is not the result of a mixing and a false negative denotes a simple or a non-shared transaction which is a CoinJoin transaction.

We can expect the false positives to be very uncommon. Due to the nature of the Bitcoin transactions, all the inputs are consumed in order to be spent in the outputs. However, it is possible that some of the intractable transactions are non-mixing transactions. So, a transaction tends to have a complete graph. It could also happen that a user, in order to reduce the fees, would merge several transactions into one. A

[4] accessed: 27-1-2020: https://bitmix.biz/en/bitcoin-mixing
[5] accessed: 27-1-2020: https://bitcoinmix.org/fr
[6] accessed: 27-1-2020: https://support.samourai.io/article/81-understanding-pools-and-pool-fees
[7] accessed: 27-1-2020: https://cryptomixer.io/
[8] accessed: 27-1-2020: https://www.wasabiwallet.io/

Public Key:	02c6047f9441ed7d6d3045406e95c07cd85c778e4b8cef3ca7abac09b95c709ee5
Script	**Addresses:**
P2PK - P2PKH	1cMh228HTCiwS8ZsaakH8A8wze1JR5ZsP
P2SH(P2PK)	3EuJgd52Tme58nZewZa39svoDtSUgL4Mgn
P2SH(P2PKH)	3D4sXNTgnVbEWaU58pDgBD82zDkthVWazv
P2WPKH	bc1qq6hag67dl53wl99vzg42z8eyzfz2xlkvxechjp
P2WSH(P2PKH)	bc1qehc8zwlgzxhfw9mdmchy2llcu4z57tsacdjt8uny4xcq3hl76euswvnzap

TABLE 5.2: Different addresses for a same public key

such particular case can be seen as a CoinJoin transaction with one entity and is not real false positive.

The false negative can occur in several cases. As it is, it does not handle the case where a financial arrangement occur in a transaction making the graph potentially complete. For example, this technique failed to detect the transactions of JoinMarket as mixing transactions because of this reason. [9]

5.3 Script Exploitation

5.3.1 Public Key Use Cases

To recall, the concept of address is simply a convenient way to indicate how you expect to receive coins. It contains all the information required to create an output. The type of the output can be inferred from the format of the address: P2PKH and P2PK addresses start with a '1', P2SH start with a '3' and P2WSH and P2WPKH start with 'bc'.

It also means that a same public key (i.e. a same identity) can lead to different addresses. In particular, P2SH and P2PKH represent more than 75% of the scripts in use between the 12th and the 19th of October 2019, (see Table. 4.1).

For example, let us consider the following public key in hexadecimal:

$$pubkey : 02c6047f9441ed7d6d3045406e95c07cd85c778e4b8cef3ca7abac09b95c709ee5.$$

It is possible to create an output locked by a P2PKH script or by a P2SH whose expected redeem script is a P2PKH. These scripts will respectively have the addresses

$$1cMh228HTCiwS8ZsaakH8A8wze1JR5ZsP$$

and

$$3D4sXNTgnVbEWaU58pDgBD82zDkthVWazv.$$ [10]

More example can be found in the Table 5.2

In the context of the analysis of clusters, this information is valuable. The public keys would permit to:

- discard the inputs controlled by more than one entity.
- perform the clustering on the public keys instead of the addresses.
- find all the data in the Blockchain relative to a public key.

[9] Example of tx: 7104bae698587b3e75563b7ea7a9aada41d9c787788bc2bf26dd201fd7eca8a2

[10] These addresses have been computed with the Bitcoin Core CLI.

Discarding Inputs To recall, the method produces false negatives when the addresses of several entities end up in the same cluster. The first step is to discard the transactions which contain an input with several addresses, because it is likely these latter belong to several entities. However, only the P2MS inputs contain several addresses.

Unfortunately, this is not enough. A P2SH script can contain a P2MS script and require several signatures, but it will owns only one address. Retrieving the public key(s) of the owner(s) is thus important.

Public Key Clustering To my knowledge, all the developed heuristics rely on the addresses. Using the public keys, could potentially be more powerful, because all the addresses derived from a same key can be considered as equivalent.

To study the difference between the two methods, the following method can be used. Firstly, the address-based heuristic will be applied to produce the clusters. Each cluster will contain its addresses and public keys. Then, another round based on the public keys will be applied on the clusters created the last round.

If no more merger occur, it would mean that the address-based heuristic already managed to collect all the different addresses for a same key into one cluster.

Blockchain parsing The analysis of some metrics of these clusters, such as value owned, number of transactions performed, etc, is based on the parsing of the Blockchain and the retrieval of the data. The public keys could be used to find all the data relative to a public key and every address format derived from it.

It is doubly in our interest to proceed that way because, as discussed in Sec. 4.2.2, the P2SH and P2WPKH are more and more used. It is likely, to see some addresses disappear in the future to be replaced by their equivalent in P2SH or P2WPKH format.

5.3.2 Extracting public keys

An address is the hash of some data that is contained either in the locking script, either in the unlocking script, either in the witness data. So, as it the Blockchain does not allow to extract the public keys from every input or output. Fortunately, the inputs are labeled with the output they redeem (see Sec. 4.1.3, so we can retrieve the public keys for any standard script. The different cases are discussed below.

P2PK This case is the simplest because the public key is always the first constant of the locking script. This means that no more information than the locking script is required and this can be applied to any output.

P2PKH In this scenario, the public key no longer lies in the locking script, but is part of the unlocking script. To retrieve the public key we need both scripts.

P2MS A P2MS script is another very simple case since all the public keys are stored within the locking script as constant data.

P2SH In a P2SH, the address is computed on the hash of a script. The public keys are not available as easily as previously.

- If there are no witness data: one has to deserialize the redeem script contained in the unlocking script. Then, the same rules are applied recursively to this redeem script and the remaining data in the locking script.
- If there are: the script is in fact a witness script nested in a P2SH [32]. The length of the constant in the unlocking script indicates if it is a P2WSH or a P2WPKH.

With this information we can extract the public key(s) from any labeled input, but it is not possible to do so with all the outputs. It would require labelling these latter with the inputs which spend them. Therefore, we didn't use the public keys to retrieve the data relative to a cluster in the Blockchain.

P2WPKH The public key is always the second constant in the witness data. The unlocking script has no valuable information.

P2WSH The redeem script is always the last constant in the witness data. Similarly to the P2SH scripts, we have to deserialize this script and use the remaining data contained in the witness to retrieve the public keys.

Definition 5.3.1 (Formalization)**.** For any locking script, unlocking script and witness data, we will use the following notations:

- $scriptPubKey$ is the locking script and $scriptPubKey.const$ is the array of constants in the script, such that $scriptPubKey.const[i]$ is the $i+1th$ constant in this array and $scriptPubKey.type$ is the type of this script.
- $scriptSig$ is the array which contains the constants in the unlocking script, such that $scriptSig[i]$ is the $i+1th$ constant in this array.
- $witness$ is the array which contains the constants in the witness data, such that $witness[i]$ is the $i+1th$ constant in this array.

To ease the notations: if arr is an array, $arr[-1]$ is the last element of the array, $arr[:]$ is the whole array and $arr[:-1]$ the whole array but the last element.

With these notations, it is possible to give a pseudo algorithm for the extraction of the public keys.

Algorithm 1 Public Keys Extraction

```
1:  function EXTRACT(type, scriptPubKey, scriptSig, witness)
2:      if scriptPubKey.type == P2PK then
3:          return scriptPubKey.const[0]
4:      else if scriptPubKey.type == P2PKH then
5:          return scriptSig[1]
6:      else if scriptPubKey.type == P2MS then
7:          return scriptPubKey.const[:]
8:      else if scriptPubKey.type == P2SH then
9:          if witness.size() == 0 then
10:             redeemScript = deserialize(scriptSig[−1])
11:             return Extract(redeemScript, scriptSig[: −1], witness)
12:         else
13:             if witness[0].size() == 20 then
14:                 return witness[1]
15:             else if witness[0].size() == 32 then
16:                 redeemScript = deserialize(witness[−1])
17:                 return Extract(redeemScript, witness[: −1], [])
18:             else
19:                 raise Error("Malformed nested script");
20:             end if
21:         end if
22:     else if scriptPubKey.type == P2WPKH then
23:         return witness[1]
24:     else if scriptPubKey.type == P2WSH then
25:         redeemScript = deserialize(witness[−1])
26:         return Extract(redeemScript, witness[: −1], [])
27:     else
28:         raise Error("Non standard script");
29:     end if
30: end function
```

Chapter 6

Clustering Results

This chapter presents the main results of the application of our heuristic on the transactions up to the 19th of October 2019. The data flow is represented in Fig. 6.1. The first step is the inference of the transaction types. So, we start with an analysis of the transaction types obtained by the graph theory approach described in 5.2. In particular, the distribution of the types over time is analysed in Sec. 6.1.1 and the distribution over all the transactions in Sec. 6.1.2.

Then, the effects of our choices to filter some transactions are discussed in Sec. 6.2. Specifically, the removal of the small outputs is studied in Sec. 6.2.1. Finally, the Section 6.2.2 is dedicated to the transactions which have been discarded due to the presence of several addresses or public keys. After the filtering, 87.1e6 of the 129.8e6 transactions remain (Fig. 6.1).

In a first time, the multi-input heuristic is then applied on the addresses (Fig. 6.1). The analysis of the cluster size and its evolution is proposed in Sec. 6.3. This analysis is performed on the clusters obtained through the address clustering (Fig. 6.1). These results are compared with the clusters obtained through the public key clustering in Sec. 6.4.

This chapter concludes with the analysis of some metrics of the clusters obtained through the public key clustering. The cluster's lifespan is studied in Sec. 6.5.1 and the cluster's financial balance in Sec. 6.5.2.

6.1 Transaction Type

The clustering technique being based on non-shared and simple transactions, it is important to dedicate a section on the analysis of the transaction types. Furthermore, the inference of the transaction types is the first step in our heuristic, see Fig. 6.1. In

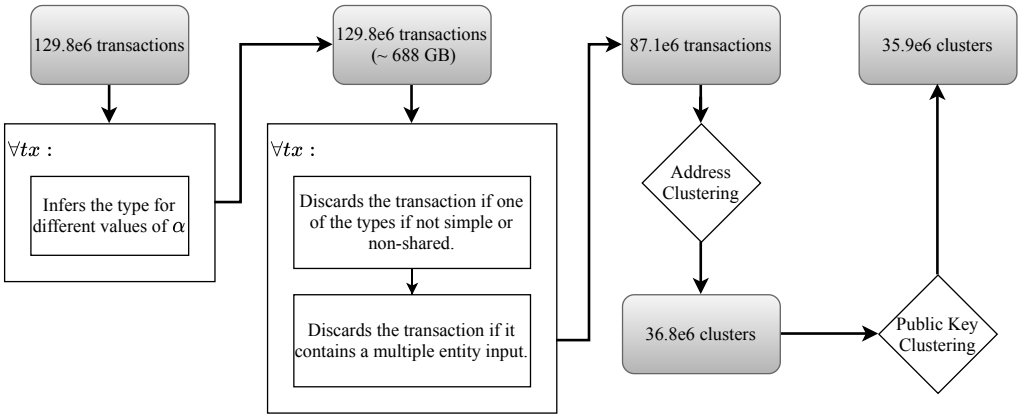


FIGURE 6.1: Data flow.

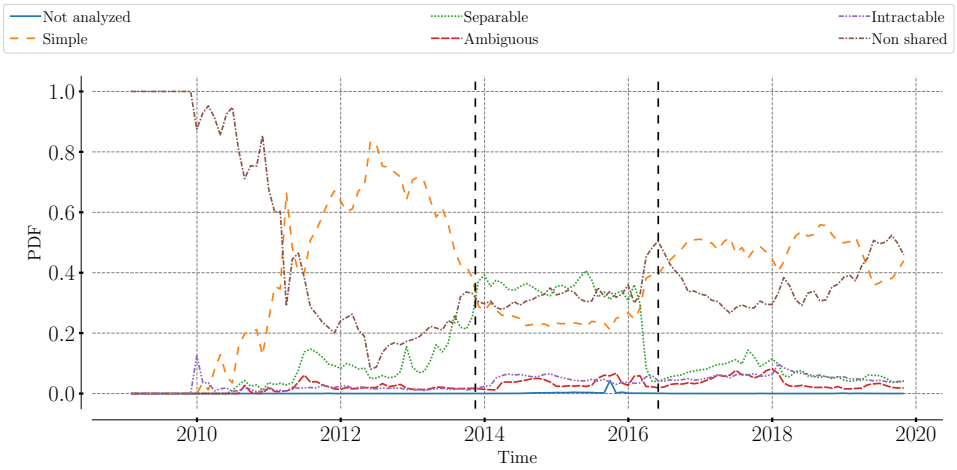


FIGURE 6.2: Evolution of the transaction type over time

particular, the evolution of the transaction types over time is presented at Fig. 6.2 and the share of these types is presented ad Fig. 6.6. To recall, the analysis has been performed on the transactions with several inputs (see Sec. 4.1.3) and without any outputs removal. The purpose was to compare with the results of the white paper [9]. In the case of the time analysis, the transaction types are collected and aggregated per month. Finally, there is another type of transaction displayed in the figures: 'NOT_ANALYZED'. This type corresponds to the transactions for which the algorithm could not be applied. This happens in the case where an input or an output of the transaction has several addresses or the script is not standard.

6.1.1 Evolution over time

This section analyzes in depth the Figure 6.2. There are 3 interesting periods of time to consider:

- Period 1: from the creation of the Bitcoin to October 2013.
- Period 2: from November 2013 to May 2016.
- Period 3: from June 2016 to October 2019.

Period 1 (Fig. 6.3) In the beginning of the Bitcoin network, it appears that most of the transactions were non-shared. Their share decreased fast as the average number of inputs and outputs increased, see Sec. 4.2.1. According to Google Trends, the first research for the terms 'bitcoin mixer' and 'bitcoin mixing' start from the date of early 2011.[1] [2] It seems to match the negligible amount of other transactions than non-shared or simple before 2011. In fact, the discarded transactions are probably false positives.

After 2011, the number of non-shared transactions keep decreasing to the profit of simple and separable transactions. This 1st period ends in October 2013, the last month in which the number of simple transactions is greater than the number of separable transactions.

[1] https://trends.google.com/trends/explore?date=all&q=bitcoin%20mixer
[2] https://trends.google.com/trends/explore?date=all&q=bitcoin%20mixing

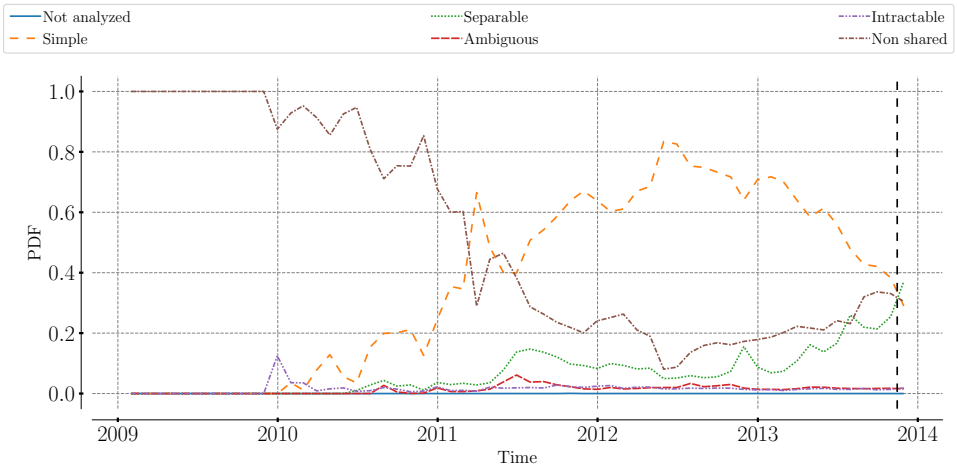


FIGURE 6.3: Evolution of the transaction type over time (1st period)

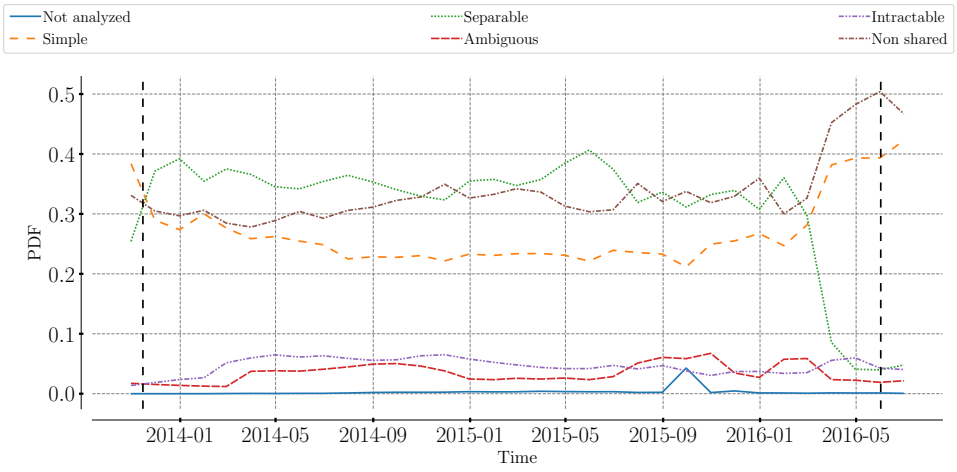


FIGURE 6.4: Evolution of the transaction type over time (2nd period)

Period 2 (Fig. 6.4) This period is particular because of the prominence of separable transactions and their sudden drop at its end. This fall indicates a brutal change in the transaction patterns and the proportion of separable transactions reaches its local minimum in May 2016 (3.97%). During this period, the number of ambiguous and intractable transaction increased. Combined with the separable transactions, it also means that about 40% of the transactions are discarded.

At last, there is a peak of 4.2% transactions which were not analyzed in September 2015.

Period 3 (Fig. 6.5) This period is the one with the most stable results. The simple and non-shared transactions represent the large majority of the transactions. The proportion of simple transactions have doubled compared to the previous period.

The proportions of ambiguous and intractable transactions are similar to the previous period.

Overall To conclude, it seems that the method applied to discard CoinJoin heuristic is highly dependent on the popular transaction patterns. It is hard to determine whether the large proportion of separable transactions in the second period is due to

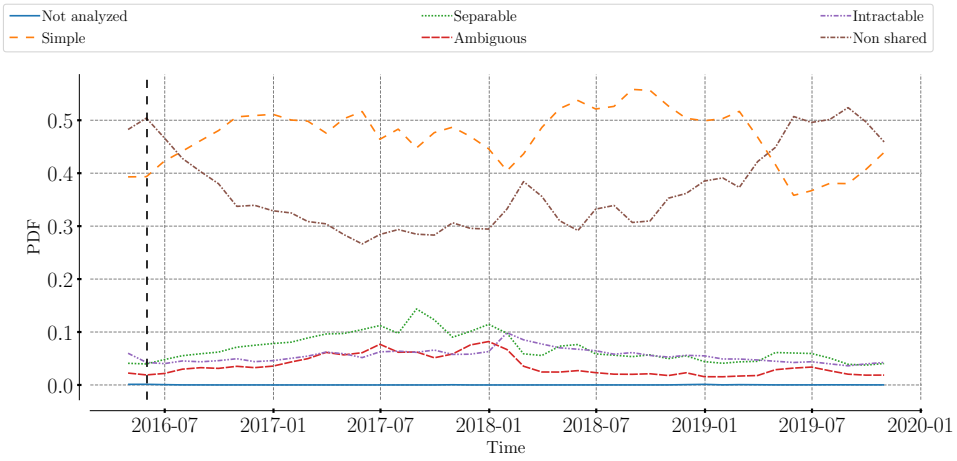


FIGURE 6.5: Evolution of the transaction type over time (3rd period)

the presence of false positives or whether the simple transaction share has increased in the 3rd period because of a raise of the number of false negatives. But, this fluctuation reminds us that we should be cautious about idioms of use.

6.1.2 Transaction type share

The figure 6.6 is the representation of the transaction types over the 129.8 million multi-inputs transactions. This is the basis for the comparison with the results presented by Yanovich et al. [9]

To recall the main results, they analyzed the transactions between May 27th and July 11th, 2016, which corresponds to the 3rd period. They filtered the transactions to keep only the shared ones. On these transactions, about 82% are simple, 9.6% are separable, 6.3% are ambiguous and the remaining 2.5% of shared send transactions are intractable.

Over the three periods of time, the non-shared transactions stand for 34.8% and the ones discarded by the analysis stand for 0.1%. On the 65.1% of shared transactions, 63.1% of them are simple, 23.8% of them are separable, 5.5% are ambiguous and 7.5% are intractable.

The chasm that occurs between the results for the simple and separable transactions comes from the instability of the results over time (Sec. 6.1.1). My results are heavily influenced by the second period. Due to the relative stability of the proportion of ambiguous transactions over time, our results are similar. Finally, my results present a higher number of intractable transactions. We have no information about the metrics they used to classify as such a transaction. Since there is no evolution in the share of intractable transaction over time, we can only assume they ran their algorithm with a different parameter.

6.2 Transaction Filtering

The comparison with the results presented by the authors [9] being done, we can focus on running the algorithm to filter and get the transactions that will be used for the cluster computation. To recall, a transaction will be used for the clustering algorithm under the following conditions:

- The transaction is simple or non-shared.

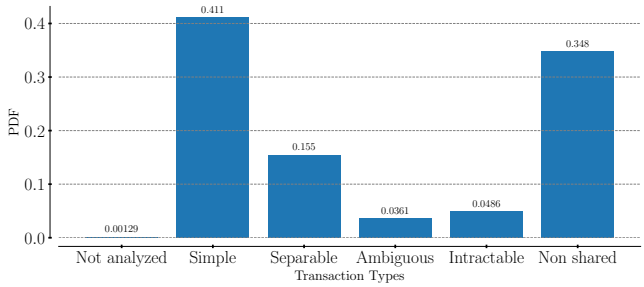


FIGURE 6.6: Types of the 129.8 million multi-input transactions

Parameter	# of transactions
0.00	87.5e6
0.02	101.0e6
0.05	103.5e6
and	87.1e6

TABLE 6.1: Number of non-shared or simple transaction according to the choice for α

- The transaction does not contain an input associated with several addresses (P2MS).
- The transaction does not contain an input associated with several public keys.

The next section 6.2.1 discusses the effects of the small output removal on the filtering. Then the section 6.2.2 is dedicated to the second and third condition.

6.2.1 Small Output Removal

The choice of the parameter α, the proportion of the total output value, is crucial to determine what is considered as a small output and whether it should be removed. The algorithm has been run 3 times for each transaction with the following values: $0.0, 0.02, 0.05$.

Contrary to my expectations, increasing this parameter doesn't lower the number of transactions exploitable. It has the reverse effect, see Table 6.1. The reason behind is that many transactions after the removal of the small outputs, have only one output at the end of the process. These transactions normally labeled as shared transactions with a low value for α are labeled as non-shared. An example of such transaction is presented at Fig. 6.7

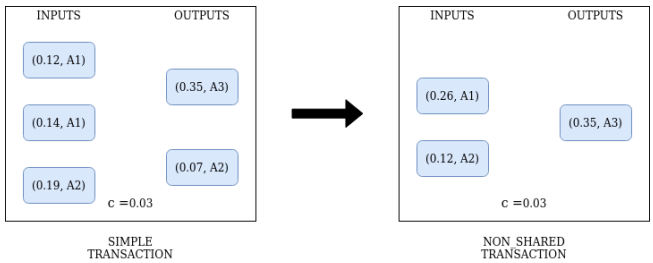


FIGURE 6.7: Example of output removal that leads to non-shared classification. The values are given in BTC. Transaction id: $'442a16294926b83e8ac67727a69ef970f426c2100fb5cc6b596aa239c2a61077'$

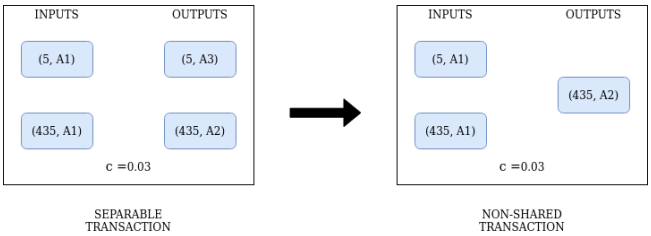


FIGURE 6.8: Example of output removal that leads to non-shared classification. The values are given in BTC. Transaction id: $'db8793340893da6533d6e717903d5c7f109b3e3264c8afddce355ff268bdd230'$

We decided to apply a simple logical AND between the different results. A transaction will be used in the clustering algorithm if and only if the transaction is labeled as simple or non-shared transaction by the 3 executions. This solution combines the best of each approach. A high value for α potentially discards dangerous transaction patterns but increases substantially the number of non-shared transactions. Combined with the results for a low or a null value for α, it discards the non-shared transactions with a suspicious pattern. The Fig. 6.8 is an example of such situation. The removal of the small output convert the transaction into a non-shared one, while the transaction was separable previously.

The 87.1 million of transactions represent 18.7% of the transactions (466 million), which is an improvement compared to the 3.77% that are exploitable by the refined multi-input heuristic. Considering that 27.8% of the transactions have more than 1 input, it is also means that the technique discards almost a third of them.

6.2.2 Multiple Entity Input

A multiple entity input is an input which is associated with several addresses or with several public keys.

Over the 129.8 million transactions, 14.0 million contain an input with one address and several public keys and about 106,000 have an input with several keys and several addresses.

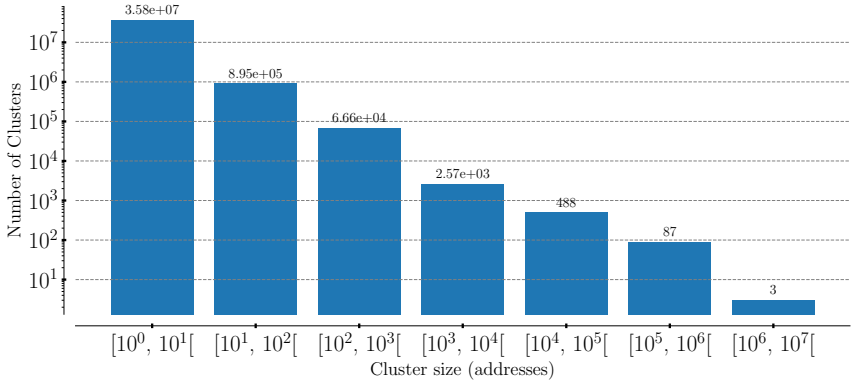
It means that 10.9% of the transactions with several inputs are controlled by more than one entity, which represents 3% of all the Bitcoin transactions. 78 transactions containing a non-standard script were found. At last, our algorithm has failed to extract the public keys of about 150,000 transactions. There are 2 reasons for a failure:

- The script is not standard.
- The redeem script (in the case of P2SH or P2WSH) is not standard.

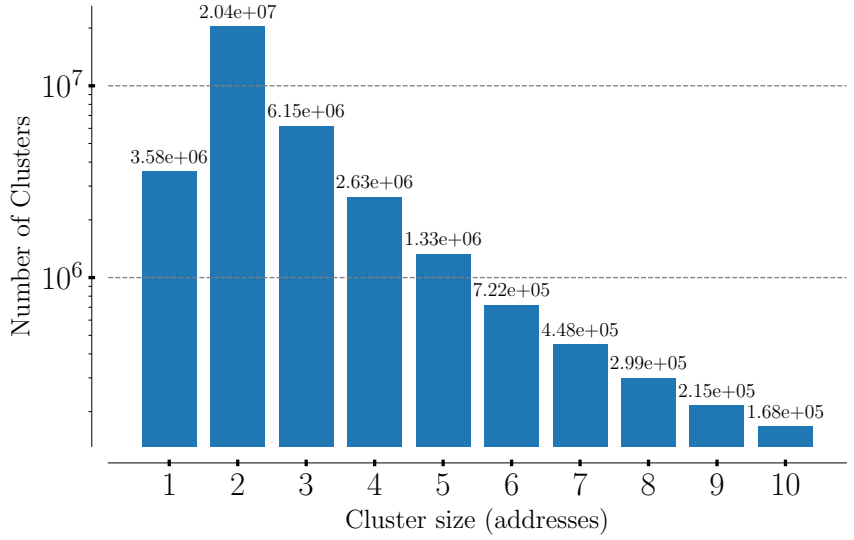
The 106,000 are transactions containing P2MS scripts, because it is the only kind of script that contains several addresses. The 14.0 million are transactions that contain P2SH or P2WSH scripts. These scripts are always associated with one address but their redeem script may require the signatures of several entities.

6.3 Cluster Size and Evolution

This section is dedicated to study the size of the computed clusters. To recall, theses clusters are based on the simple and non-shared transactions. The address-based heuristic produced 36.8 million clusters, see Fig. 6.1.



(A) Distribution of the cluster size.



(B) Distribution of the cluster size between 1 and 10 addresses.

FIGURE 6.9: Cluster size

The largest cluster was composed of 20,416,815 addresses. However, such a cluster size is infrequent, as illustrated in Fig. 6.9. 36.0 million clusters have a size between 1 and 10. Furthermore, the number of clusters seems to decrease linearly with the number of addresses. There are only about 3000 clusters with more than 10,000 addresses.

Note that, these large clusters are not necessarily the result of false negatives. For example, Mt Gox (an old exchange service) was known to be bound to a cluster with more than 10 million addresses. [36] The size was due to the possibility for the users to import their private key into the wallet online. The services that provide a similar feature may also be responsible for the creation of large clusters. Sadly, there is no ground truth in the Bitcoin domain to assert the validity of these huge clusters.

If we look in details the first category, the clusters which are composed between 1 and 10 addresses, it appears that most of the entities own 2 addresses. Moreover, the number of clusters with more than 2 addresses decrease drastically with the number of addresses.

Another interesting metric to study is the cluster size evolution. If we merge 3 clusters $A, B, C, D, C, E, F, B, F, G$, the size increases are said to be 2 and 1 because the new addresses are added to the largest cluster of the group. Similarly to what observed M. Harrigan and C. Fretter, [36], it appears that large increases in address

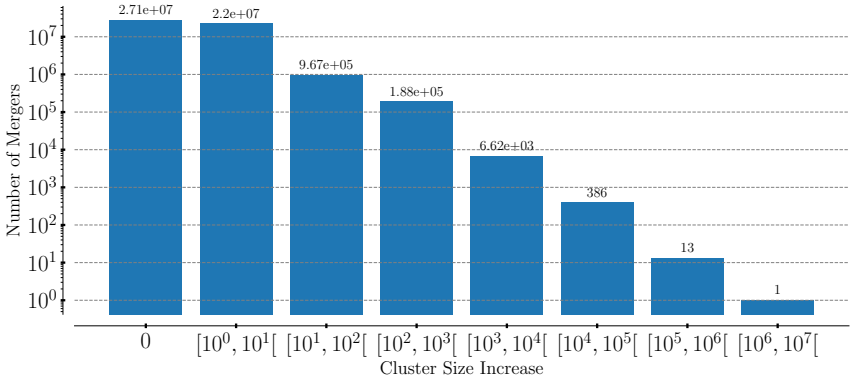


FIGURE 6.10: Cluster size increase

cluster sizes are rare (Fig. 6.10). However, the presence of a large merger is a plausible sign of false negatives. It is also worth to mention that the majority of the mergers between two clusters does not increase the size of the resulting merged cluster.

We finish this section with a brief discussion of 2 other metrics. Each time a cluster is read by our algorithm, it checks if some collisions, i.e. a set of addresses in common, occur with the clusters that are already in memory. If it does, the clusters that have addresses in common are merged. The Fig. 6.11a illustrates the number of cluster merged and the Fig. 6.11b the sizes of the clusters added to the largest. The shapes of the cluster size increases and the size of the merged clusters are very similar.

The number of clusters merged at each iteration is unsurprisingly low. With the exception of a merger of 1432 clusters, the number of clusters is below 434. Actually, 96.7% of the iterations produced the merger of 2 clusters.

6.4 Public Key Clustering

To evaluate, the gain of the clustering on keys rather on the addresses, the clustering algorithm is run on the clusters produced in the Sec. 6.2.1.

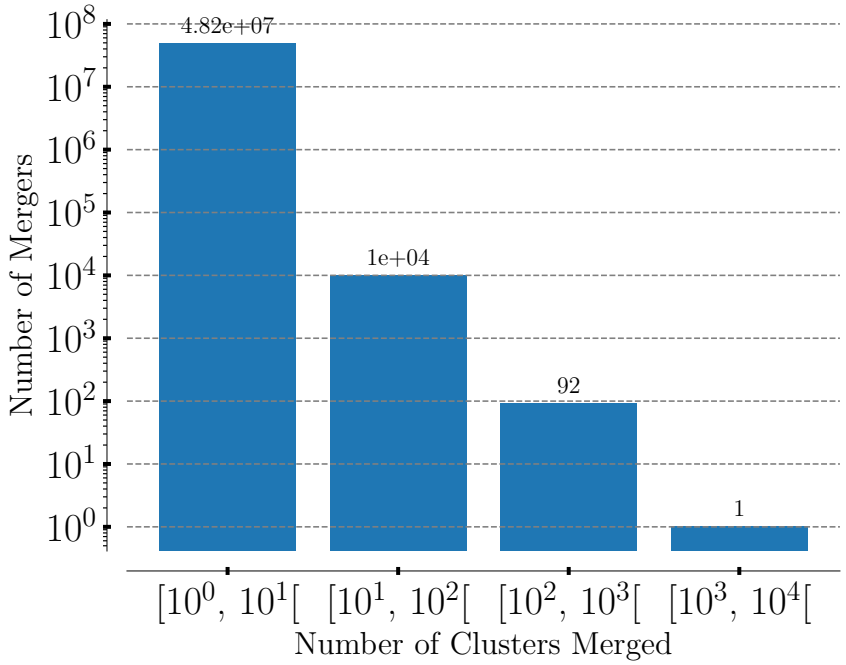
The 36.8e6 clusters are merged into 35.9e6 which represents a decrease of 2.5%, see Fig. 6.1. The mergers indicate that some clusters were not merged the last round because their address sets are disjoint but some of these addresses correspond to a same public key. This is because the different formats that may exist for a same public key.

As the use of P2PKH decreases fast over time (see Sec. 4.2.2) for the benefit of other kinds of scripts, this phenomenon should increase.

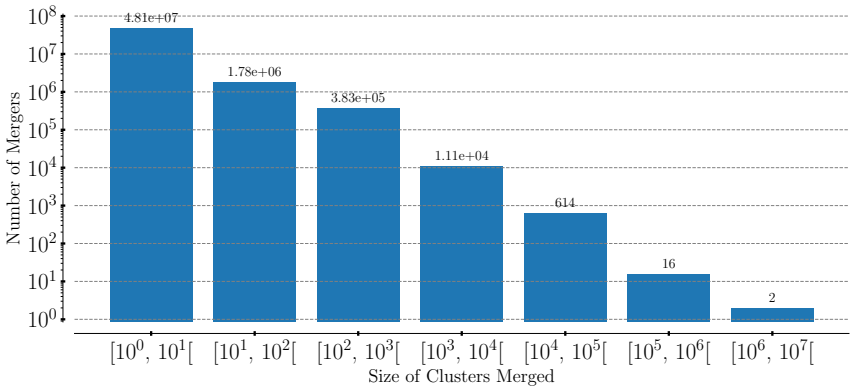
The similarity between the distributions of the cluster sizes (Fig. 6.12a, Fig. 6.9a) seems to indicate a stability in the results. The cluster size raise figure (Fig. 6.12b) would indicate also that direction.

6.5 Cluster Analysis

This last section focuses on the analysis of some characteristics of the clusters as their financial balance (Sec. 6.5.2) or their lifespan (Sec. 6.5.1). The transactions in Blockchain have been parsed to collect the data relative to the clusters. The activity of a cluster is the presence of one of his addresses in a transaction. As the number of transactions is potentially huge for the clusters, the data are aggregated per month.

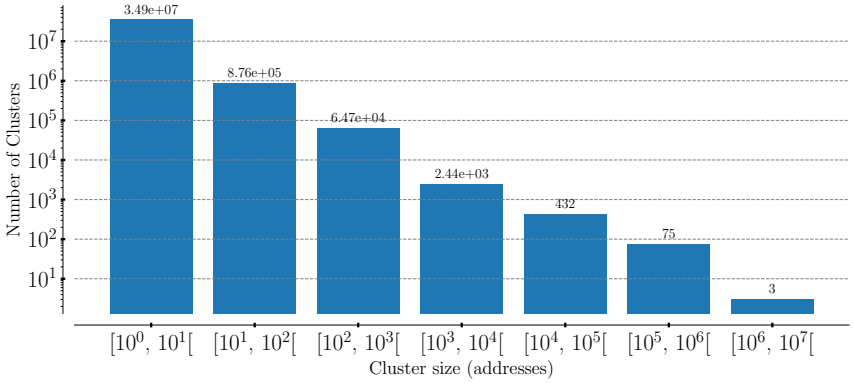


(A) Number of merged clusters

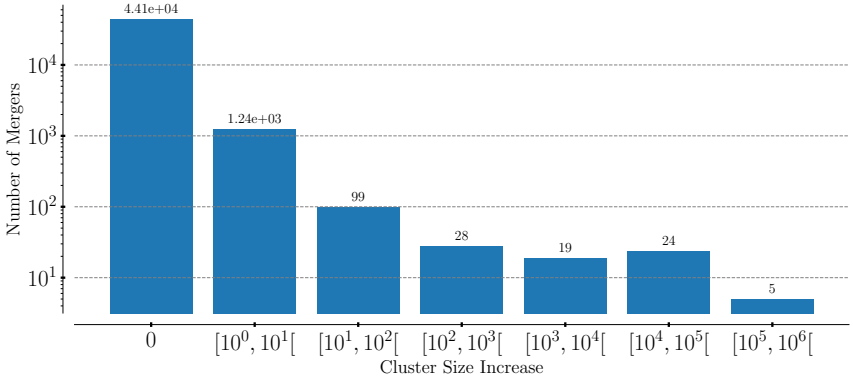


(B) Size of the merged clusters.

FIGURE 6.11: Size and number of clusters that are merged during the execution of the algorithm.



(A) Size of the clusters.



(B) Size increase of the clusters.

FIGURE 6.12: Size and size increase of the clusters based on the public keys.

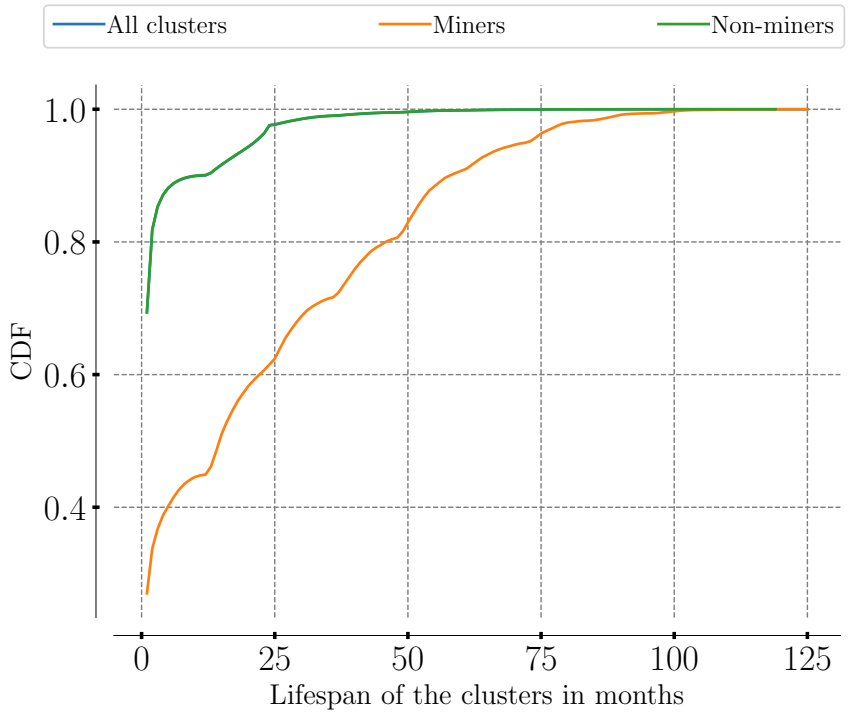


FIGURE 6.13: Lifespan of the Clusters.

Each cluster is associated with one or several records. Each record is the summary of its activity during a month.

Then, the clusters are split into 2 categories: the ones that are responsible for the generation of new coins and the others. For each of this category, the analysis is repeated. This separation revealed 48,356 mining entities and $35,8e^6$ other clusters.

6.5.1 Lifespan Analysis

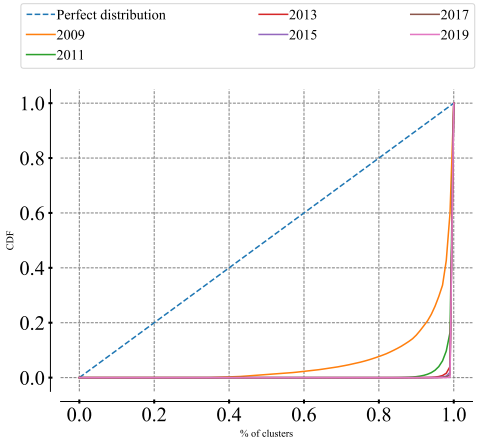
The lifespan of a cluster is defined as the number of months during which the cluster has been active. Thus, a cluster which only sent or received money once in its history is said to be active during a month. This loss in accuracy could be prevented with a finer granularity at the cost of a greater number of records.

Almost 70% of the clusters have a lifespan of a month and more than 90% are active less than a year, see Fig. 6.13. The non-miners represent 0.999% of the clusters, so the curve is identical to the curve for the non-miners.

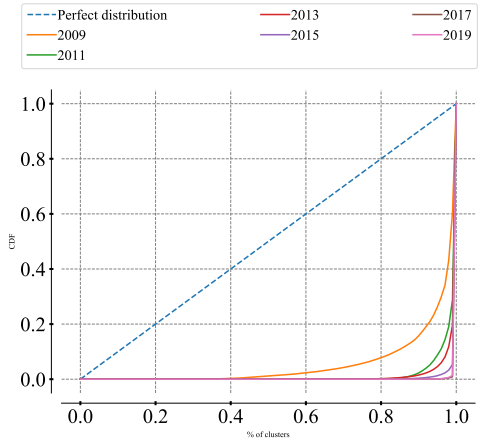
The study of the miners reveals that these entities are less ephemeral. Nonetheless, more than 40% of the clusters last less than 1 year. Considering the cost to deploy the architecture to mine new coins, this seems unlikely. This could be a sign of the limitation of our clustering algorithm. The oldest miner is active for about 10 years.

6.5.2 Financial Analysis

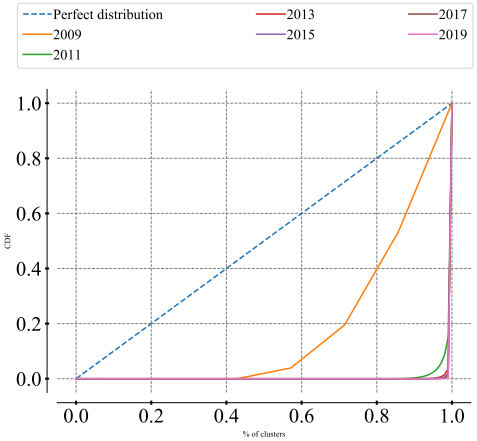
In economics, the Lorenz curve is a graphical way to represent the distribution of wealth inside a population. The Lorenz curve is computed every 2 years to show the evolution over time. The Gini indices for each of those years are presented in Table 6.2.



(A) Lorenz Curve of the Clusters



(B) Lorenz Curve of the Miners



(C) Lorenz Curve of the Non-Miners

FIGURE 6.14: Lorenz Curves of the Clusters.

Firstly, the Lorenz curves of the whole set of clusters are presented at Fig. 6.14a. The curves for the miners and non-miners are presented at Fig. 6.14b and Fig. 6.14c.

It appears that the wealth has always been owned by a minority of entities. But over time, the wealth is less and less distributed. The shape of the curve is heavily influenced by the clusters that are not responsible for the generation of new coins except for the year 2009. It is explained by the fact that most of the coins are held by the non-miners entities, see Fig. 6.15. The year 2009 is an exception because it was the beginning of the Bitcoin history.

6.6 Take-Home message

To sum up this chapter, we can consider the following points.

Firstly, it appears that the Bitcoin network is highly volatile; the transaction patterns can vary over the time and change suddenly. Any technique based on these patterns should be aware of this characteristic.

Secondly, the exploit of the knowledge of the public keys in the transactions allows to remove the transactions forged by several entities. We found that 10.9% of the transactions fall into this case. The filtering of those reduces the risk of incorrectly merging clusters.

Year	All	Miners	Non-Miners
2009	0.8900	0.8882	0.6382
2011	0.9811	0.9682	0.9822
2013	0.9886	0.9763	0.9887
2015	0.9894	0.9863	0.9894
2017	0.9897	0.9896	0.9897
2019	0.9900	0.9894	0.9900

TABLE 6.2: Gini Indices relative to the Lorenz Curves

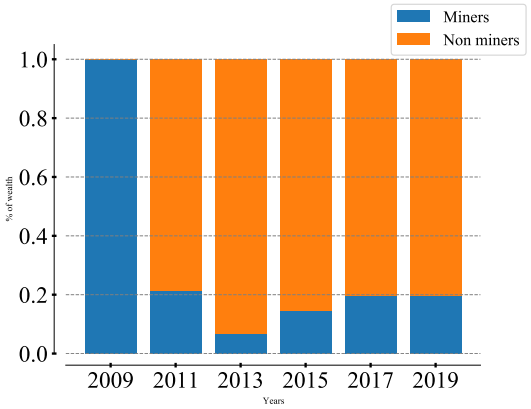


FIGURE 6.15: Distribution of wealth between the miners and the other entities.

Then, The clustering techniques based on the public key improves slightly the efficiency. It is capable to cluster some addresses and public keys that were not merged into a same cluster with the clustering based on the addresses.

Finally, the analysis of the clusters revealed that most of them are ephemeral. Almost 70% of them are active during a month. It also disclosed that the mining clusters are less ephemeral than the non-mining ones. The financial analysis revealed that distribution of wealth is highly concentrated amongst a minority of clusters and mostly owned by non-mining clusters.

Chapter 7

Conclusion

Nowadays, the Bitcoin is still the most popular cryptocurrency and its popularity does not seem to decrease. The ease with which the peers involved in a transaction can exchange currency and the pseudo anonymity of the Bitcoin attracted all sorts of criminal activities. To recall only a few: Helix was a Bitcoin mixer involved into money laundry [15] and Silk Road was a market proposing illicit products and services and used the Bitcoin. [16] It is critical in many investigations to be able to link the financial activities between the entities. The clustering is one of the techniques that permits to retrieve and link the transactions that are emitted by a same entity.

First, this thesis provided a survey on the current Bitcoin application layer. It also showed that some metrics, such as the number of inputs and outputs or the script type distribution, are unstable over time.

Then, a new approach has been proposed and implemented to tackle the issue of coinjoin transactions that counter the multi-input heuristic. This approach is composed of a graph-theory formalization of the transaction semantic and the extraction of the public keys from the inputs. This approach led to several conclusions. i) The transaction's popular patterns vary widely over time; the heuristics based on an idiom of use should be avoided as far as possible.

ii) At least 10.9 % of the multi-input transactions have an input controlled by several entities, the large majority of these transactions is bound with one address and several public keys. These transactions are responsible for incorrect clustering and we know thanks to previous papers [4] that even a small number of them can lead to the creation of super clusters.

It seems that the cluster's generation is such that their size growths smoothly and is similar to what M. Harrigan and C. Fretter observed. [36] The analysis of the clusters revealed that, iii) most of them are ephemeral. Almost 70 % of the clusters obtained are active during a month and 90 % of them less than 1 year. iv) Finally, and the wealth distribution is very unequal. The Gini index for the year 2019 is 0.99.

7.1 Further Work

Public Keys and Blockchain Parsing So far, the collection of the data relative to the clusters is based on the address. The set of addresses for each cluster is obtained from the clustering of the multi-input transactions. If some addresses derived from the public keys of the cluster do not belong to this set, a part of the data relative to this cluster is not used.

If the consumed outputs were labeled with their corresponding input, it would become possible to extract the public keys from these pieces of information.

Clusters to Real World Entities The topic of trying to link some clusters to known entities in the world is a problem tackled by some papers.

- Meiklejohn et al. [4] proposed to make transactions to with several services in order to tag some of their addresses with their real identity.
- Ermilov et al. [5] decided to use a different approach to tag addresses. The passive approach is based on the web crawling of several of public forums, user profiles or online markets. The active approach requires the manual analysis of the Bitcoin companies and data actualization procedures. For example, many service providers include always the same prefix in their addresses. In its most basic form, the address is the hash of a public key, they just keep on generating public keys until they find one with the right prefix. For example, the gambling website Satoshi Bones had users send money to addresses containing the string "bones" in positions 2–6, such as 1bonesEeTcABPjLzAb1VkFgySY6Zqu3sX. [18, Chapter 4.1]
- Finally, many miners choose to include some data into the coinbase transactions in order to be identified.

Besides increasing the knowledge on the clusters, this topic could also help to study the results and the defects of the employed clustering heuristic. Spotting the transactions that led to a wrong clustering could help to improve the heuristic afterward.

Bibliography

[1] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: *Cryptography Mailing list at https://metzdowd.com* (Mar. 2009).

[2] Vicky Ng. *Why haven't we all bought cryptocurrency yet?* June 2018. URL: https://www.finder.com/why-people-arent-buying-cryptocurrency. (last consultation: 04.01.2021).

[3] Ivan Mercanti, Stefano Bistarelli, and Francesco Santini. "An Analysis of Non-standard Bitcoin Transactions". In: June 2018.

[4] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon Mccoy, Geoffrey Voelker, and Stefan Savage. "A fistful of bitcoins: characterizing payments among men with no names". In: *Proc. ACM Conference on Internet Measurement Conference (IMC)*. Oct. 2013.

[5] Dmitry Ermilov, Maxim Panov, and Yury Yanovich. "Automatic Bitcoin Address Clustering". In: *Proc. IEEE International Conference on Machine Learning and Applications (ICMLA)*. Dec. 2017.

[6] Vinnie Monaco. "Identifying Bitcoin users by transaction behavior". In: *Proc. SPIE DSS: Biometric and Surveillance Technology for Human and Activity Identification XII*. Apr. 2015.

[7] Malte Moser, Rainer Bohme, and Dominic Breuker. "An inquiry into money laundering tools in the Bitcoin ecosystem". In: *Proc. IEEE APWG eCrime Researchers Summit*. Sept. 2013.

[8] Younggee Hong, Hyunsoo Kwon, Jihwan Lee, and Junbeom Hur. "A Practical De-mixing Algorithm for Bitcoin Mixing Services". In: *Proc. ACM ASIA CCS: Asia Conference on Computer and Communications Security*. May 2018.

[9] Yuriy Yanovich, Pavel Mischenko, and Aleksei Ostrovskiy. *Shared Send Untangling in Bitcoin*. Tech. rep. The Bitfury Group, Aug. 2016. URL: https://bitfury.com/content/downloads/bitfury_whitepaper_shared_send_untangling_in_bitcoin_8_24_2016.pdf.

[10] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua Kroll, and Edward Felten. "Mixcoin: Anonymity for Bitcoin with Accountable Mixes". In: *Proc. International Conference on Financial Cryptography and Data Security*. Mar. 2014.

[11] Ian Miers, Christina Garman, Matthew Green, and A.D. Rubin. "Zerocoin: Anonymous Distributed E-Cash from Bitcoin". In: *Proc. IEEE Symposium on Security and Privacy*. May 2013.

[12] Eli Ben-sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *Proc. IEEE Symposium on Security and Privacy (SP)*. May 2014.

[13] gmaxwell. *I taint rich! (Raw txn fun and disrupting 'taint' analysis; >51kBTC linked!)* Jan. 2013. URL: https://bitcointalk.org/?topic=139581. (last consultation: 09.10.2020).

[14] gmaxwell. *CoinJoin: Bitcoin privacy for the real world*. Aug. 2013. URL: https://bitcointalk.org/?topic=279249. (last consultation: 09.10.2020).

[15] Titus. *Bitcoin Mixer Helix in need of explanation: $ 60 million fine imposed*. Oct. 2020. URL: https://medium.com/the-capital/bitcoin-mixer-helix-in-need-of-explanation-60-million-fine-imposed-3980e39e94c0. (last consultation: 22.10.2020).

[16] U.S. Attorney's Office. *United States Files A Civil Action To Forfeit Cryptocurrency Valued At Over One Billion U.S. Dollars*. Nov. 2020. URL: https://www.justice.gov/usao-ndca/pr/united-states-files-civil-action-forfeit-cryptocurrency-valued-over-one-billion-us. (last consultation: 04.01.2021).

[17] Congcong Ye, Guoqiang Li, Hongming Cai, Yonggen Gu, and Akira Fukuda. "Analysis of Security in Blockchain: Case Study in 51%-Attack Detecting". In: *Proc. IEEE International Conference on Dependable Systems and Their Applications (DSA)*. Sept. 2018.

[18] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[19] Joachim Breitner and Nadia Heninger. "Biased Nonce Sense: Lattice Attacks Against Weak ECDSA Signatures, in Cryptocurrencies". In: *Financial Cryptography and Data Security*. Ed. by Ian Goldberg and Tyler Moore. Cham: Springer International Publishing, Sept. 2019.

[20] Wikipedia Community. *Original Bitcoin client*. [Last edition 17-August-2015]. URL: https://en.bitcoin.it/wiki/Original_Bitcoin_client. (last consultation: 05.10.2020).

[21] Eric Lombrozo and Pieter Wuille. *Bip 144*. Jan. 2016. URL: https://github.com/bitcoin/bips/blob/master/bip-0144.mediawiki. (last consultation: 06.10.2020).

[22] Satoshi Nakamoto (2009-2010) and The Bitcoin Core developers (2009-2019). *Standard transaction test - Source Code*. [Last commit on 13-Oct-2020]. URL: https://github.com/bitcoin/bitcoin/blob/master/src/policy/policy.cpp. (last consultation: 05.10.2020).

[23] Sergio Demian Lerner. *A Bitcoin transaction that takes 5 hours to verify*. Jan. 2017. URL: https://bitslog.com/2017/01/08/a-bitcoin-transaction-that-takes-5-hours-to-verify/. (last consultation: 05.10.2020).

[24] David A. Harding. *What's the use of not relaying non-standard transactions if anyone can still use them using P2SH?* June 2018. URL: https://bitcoin.stackexchange.com/questions/76541/whats-the-use-of-not-relaying-non-standard-transactions-if-anyone-can-still-use. (last consultation: 05.10.2020).

[25] Wikipedia Community. *Script - Bitcoin Wiki*. [Last edition 25-July-2020]. URL: https://en.bitcoin.it/wiki/Script#Reserved_words. (last consultation: 05.10.2020).

[26] Wikipedia Community. *Script - Bitcoin Wiki*. [Last edition 25-July-2020]. URL: https://en.bitcoin.it/wiki/Script. (last consultation: 05.10.2020).

[27] Wikipedia Community. *Technical aspects of btc addresses*. [Last edition 28-March-2019]. URL: https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses. (last consultation: 05.10.2020).

[28] gavinandresen. *Bitcoin version 0.9.0 released*. Mar. 2014. URL: https://github.com/bitcoin/bitcoin/blob/master/doc/release-notes/release-notes-0.9.0.md. (last consultation: 05.10.2020).

[29] The Bitcoin Core developers. *Bitcoin Core version 0.9.0 released*. Mar. 2014. URL: https://bitcoin.org/en/release/v0.9.0#opreturn-and-data-in-the-block-chain. (last consultation: 05.10.2020).

[30] Gavin Andresen. *Bip 11*. Oct. 2011. URL: https://github.com/bitcoin/bips/blob/master/bip-0011.mediawiki. (last consultation: 06.10.2020).

[31] Gavin Andresen. *Bip 16*. Jan. 2012. URL: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki. (last consultation: 06.10.2020).

[32] Eric Lombrozo, Johnson Lau, and Pieter Wuille. *Bip 141*. Dec. 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki. (last consultation: 06.10.2020).

[33] Pieter Wuille. *Bip 62*. Mar. 2014. URL: https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki. (last consultation: 06.10.2020).

[34] The Bitcoin Core developers. *Segwit Benefits*. Jan. 2016. URL: https://bitcoincore.org/en/2016/01/26/segwit-benefits/. (last consultation: 06.10.2020).

[35] Pieter Wuille and Greg Maxwell. *Bip 173*. Mar. 2017. URL: https://github.com/bitcoin/bips/blob/master/bip-0173.mediawiki. (last consultation: 06.10.2020).

[36] Martin Harrigan and Christoph Fretter. "The Unreasonable Effectiveness of Address Clustering". In: *Proc. IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. July 2016.

[37] Gavin Andresen. *Bip 13*. Oct. 2011. URL: https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki. (last consultation: 06.10.2020).

Appendix A

Technical Annex

A.1 Address Format for version 1 Bitcoin Address

The following method describes precisely how to generate Bitcoin address from a public key (from step 1) or a public key hash (from step 3). [27] [37]

1. Perform SHA-256 hashing on the public key
2. Perform RIPEMD-160 hashing on the result of SHA-256
3. Add a version byte in front of RIPEMD-160 hash
4. Perform SHA-256 hash on the extended RIPEMD-160 result
5. Perform SHA-256 hash on the result of the previous SHA-256 hash
6. Take the first 4 bytes of the second SHA-256 hash. This is the address checksum
7. Add the 4 checksum bytes from stage 6 at the end of extended RIPEMD-160 hash from stage 3. This is the 25-byte binary Bitcoin Address.
8. Convert the result from a byte string into a base58 string using Base58Check encoding. This is the most commonly used Bitcoin Address format.

The version byte to add in the step 3 is:

- 0x00 for P2PK, P2PKH, P2MS (main network).
- 0x05 for P2SH.

It will results in different address formats: the script addresses will start with a 3 while the others with a 1.

A.2 Address Format for Segregated Witness

A new format has been specified for the Segwit addresses in the BIP 173 to replace the previous format in the BIP 142, that is to say a bit more than a year after the standardization of Segwit scripts. [35]

A segwit address is a Bech32 encoding of:

- The human-readable part "bc"[7] for mainnet, and "tb"[8] for testnet.
- The data-part values:
  - 1 byte: the witness version

- A conversion of the 2-to-40-byte witness program (as defined by BIP141) to base32: Start with the bits of the witness program, most significant bit per byte first. Re-arrange those bits into groups of 5, and pad with zeroes at the end if needed. Translate those bits to characters using the table above.

Such Bech32 addresses (in the main network) always start with the prefix "bc1".